UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION,

Petitioner,

v.

BRADIUM TECHNOLOGIES LLC,

Patent Owner.

CASE: IPR2017-01818

Patent No. 9,641,645 B2

DECLARATION OF PROF. WILLIAM R. MICHALSON IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 9,641,645 B2

I hereby declare that all the statements made in this Declaration are of my own knowledge and true; that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

I declare under the penalty of perjury that all statements made in this Declaration are true and correct.

Executed July 19, 2017 in Douglas, Massachusetts.

/William R. Michalson/
William R. Michalson

TABLE OF CONTENTS

Page

LIST OF APPENDICES IV
I. INTRODUCTION 1
II. SUMMARY OF OPINIONS 3
III. QUALIFICATIONS AND EXPERIENCE 6
A. Education and Work Experience 6
B. Compensation 9
C. Documents and Other Materials Relied Upon 10
IV. STATEMENT OF LEGAL PRINCIPLES 11
A. Claim Construction 11
B. Anticipation 11
C. Obviousness 12
V. LEVEL OF ORDINARY SKILL IN THE ART 13
VI. TECHNOLOGY BACKGROUND OF THE '645 PATENT 17
A. Data Communications Over the Internet 19
B. Data Communications in Wireless Mobile Systems 22
C. Image Tiles and Image Pyramids 23
D. Compression of Image Tiles 33
E. Progressive Image Resolution Enhancement 35
F. Three-Dimensional Graphics 37
1. Overview of 3D Computer Graphics principles 37
2. Texture 43
3. Virtual Reality Modeling Language (VRML) 47
G. Mip-Maps 48
H. Storage of image data 54
VII. OVERVIEW OF THE '645 PATENT 56

TABLE OF CONTENTS
(continued)

Page

VIII. IDENTIFICATION OF THE PRIOR ART AND SUMMARY OF OPINIONS 62

A. Reddy 62

B. Woods 64

C. Chiarabini 64

D. Fuller 67

IX. CLAIM CONSTRUCTION 68

A. "Wireless Portable Device" in All Claims Except Claims 6, 10, 18, 22, 30, and 34-36 69

B. "Thereby Enabling Efficient Use of Network Bandwidth in Conditions of Network Latency" in Claims 8, 20, and 32 70

C. "Configure[d/s] ... as a server to provide access to [the] at least some image parcels [received by the wireless portable device]" in Claims 7, 19, and 31 71

D. "Image Parcel" in Claims 1-4, 7-8, 12-16, 19-20, 25-28, 31-32, and 36 74

E. All Remaining Claim Terms 76

X. UNPATENTABILITY OF CLAIMS 1-36 OF THE '645 PATENT 77

A. GROUND 1: CLAIMS 1-7, 9-11, 13-19, 21-23, 25-31, AND 33-35 ARE UNPATENTABLE UNDER 35 U.S.C. § 103(a) AS BEING OBVIOUS OVER REDDY IN VIEW OF WOODS 77

1. Overview of Asserted References 78

2. Motivations to Combine Reddy and Woods 98

3. Independent Claim 1 Is Obvious 106

4. Independent Claim 13 Is Obvious 123

5. Independent Claim 25 Is Obvious 126

6. Dependent Claims 2, 14, and 26 Are Obvious 129

7. Dependent Claims 3, 15, and 27 Are Obvious 134

8. Dependent Claims 4, 16, and 28 Are Obvious 136

TABLE OF CONTENTS
(continued)

Page

9. Dependent Claims 5, 17, and 29 Are Obvious 137
10. Dependent Claims 6, 18, and 30 Are Obvious 140
11. Dependent Claims 7, 19, and 31 Are Obvious 144
12. Dependent Claims 9, 21, and 33 Are Obvious 145
13. Dependent Claims 10, 22, and 34 Are Obvious 151
14. Dependent Claims 11, 23, and 35 Are Obvious 152

B. GROUND 2: CLAIMS 8, 20, AND 32 ARE UNPATENTABLE UNDER 35 U.S.C. § 103(a) AS BEING OBVIOUS OVER REDDY IN VIEW OF WOODS AND CHIARABINI 155

1. The Reddy-Woods-Chiarabini Combination 155
2. Motivations to Combine 156
3. Dependent Claims 8, 20, and 32 are Obvious 157

C. GROUND 3: CLAIMS 12, 24, AND 36 ARE UNPATENTABLE UNDER 35 U.S.C. § 103(a) AS BEING OBVIOUS OVER REDDY IN VIEW OF WOODS AND FULLER 166

1. The Reddy-Woods-Fuller Combination 166
2. Motivations to Combine Reddy, Woods, and Fuller 167
3. Dependent Claims 12, 24, and 36 Are Obvious 169

XI. CONCLUSION 170

LIST OF APPENDICES

Appendix A	Curriculum Vitae of William R. Michalson
Appendix B	Excerpt of Hanan Samet, *The Design and Analysis of Spatial Data Structures*, University of Maryland (1989)
Appendix C	U.S. Patent No. 5,263,136 (DeAguiar et al)
Appendix D	U.S. Patent 4,972,319 (Delorme)
Appendix F	International Telegraph and Telephone Consultative Committee ("CCITT") Recommendation T.81, September 1992
Appendix G	Ken Cabeen & Peter Gent, *Image Compression and the Discrete Cosine Transform*
Appendix H	M. Antonini, *Image Coding Using Wavelet Transform* , IEEE Transactions on Image Processing, Vol. 1, No. 2, April 1992.
Appendix I	U.S. Patent No. 5,321,520 (Inga et al)
Appendix J	U.S. Patent No. 6,182,114 (Yap et al.)
Appendix K	U.S. Patent No. 5,179,638 (Dawson et al)
Appendix L	Lance Williams, *Pyramidal Parametrics*, Computer Graphics, vol. 17, no. 3, July 1983
Appendix M	OpenGL Standard Version 1.1, March 1997, available: https://www.opengl.org/documentation/specs/version1.1/glspec1.1/node84.html#SECTION00681100000000000000
Appendix N	H. Hoppe, *Progressive Meshes*, SIGGRAPH '96: Proceedings of the 23rd annual conference on computer graphics and interactive techniques, pp. 99-108
Appendix O	U.S. Patent 5,798,770 (Baldwin)
Appendix P	U.S. Patent No. 5,987,256 (Wu et al)

Appendix R	Boris Rabinovich & Craig Gotsman, *Visualization of Large Terrains in Resource-Limited Computing Environments* (1997)
Appendix S	*User Datagram Protocol (UDP) (Windows CE 5.0,* Microsoft, Available: https://msdn.microsoft.com/en-us/library/ms885773.aspx [Accessed April 28, 2015]
Appendix T	OpenGL Standard Version 1.2.1, April 1999, available: https://www.opengl.org/documentation/specs/version1.2/opengl 1.2.1.pdf
Appendix X	George H. Forman and John Zahorjan, "The challenges of mobile computing," *Computer* vol. 27, no. 4, pp. 38, 47 (April 1994)
Appendix Y	K. Brown and S. Singh, *A Network Architecture for Mobile Computing*, INFOCOM '96, Fifteenth Annual Joint Conference of the IEEE Computer Societies, Networking the Next Generation, Proceedings IEEE vol. 3, pp. 1388-139
Appendix Z	Kreller, B. et al "UMTS: a middleware architecture and mobile API approach," Personal Communications, IEEE, vol. 5, no. 2, pp. 32-38 (April 1998)
Appendix AA	Hansen, J. et al, "Real-time synthetic vision cockpit display for general aviation," AeroSense '99, International Society for Optics and Photonics, 1999
Appendix BB	U.S. Patent No. 5,760,783 to Migdal et al ("Migdal")
Appendix EE	Theresa-Marie Rhyne, *A Commentary on GeoVRML: A Tool for 3D Representation of GeoReferenced Data on the Web*, International Journal of Geographic Information Sciences, issue 4 of volume 13, 1999
Appendix GG	GeoTIFF Format Specification Revision 1.0
Appendix HH	TIFF Revision 6.0, dated June 3, 1992.
Appendix II	FlashPix Format Specification v1.0, dated September 11, 1996

Appendix KK	The Virtual Reality Modeling Language ISO/IEC 14772-1:1997
Appendix LL	Marc H. Brown and Robert A. Shillner, "DeckScape: an experimental Web browser," Computer Networks and ISDN Systems 27 (1995) 1097-1104
Appendix NN	IPR2016-00448 and IPR2016-00449, transcript of April 18, 2017 hearing (non-confidential portions)

I. INTRODUCTION

1. My name is William R. Michalson. I am a professor of electrical and computer engineering at Worcester Polytechnic Institute in Massachusetts.

2. I have been engaged by Microsoft Corporation ("Microsoft") to investigate and opine on certain issues relating to U.S. Patent No. 9,641,645 B2 (the "'645 Patent") entitled "Optimized Image Delivery Over Limited Bandwidth Communication Channels" in Microsoft's Petition for *Inter Partes* Review of the '645 Patent ("Microsoft IPR Petition") which requests the Patent Trial and Appeal Board ("PTAB") to review and cancel all claims of the '645 Patent—claims 1-36 ("Challenged Claims").

3. I have also been engaged by Microsoft to investigate and opine on certain issues relating to six other patents that are related to the '645 Patent—U.S. Patent Nos. 7,908,343 B2 ("the '343 Patent"), 7,139,794 B2 ("the '794 Patent"), 8,924,506 B2 ("the '506 Patent"), 9,253,239 B2 ("the '239 Patent"); 9,641,644 B2 ("the '644 Patent"); and 9,635,136 B2 ("the '136 Patent")—in additional petitions for *inter partes* review by Microsoft. I understand that Bradium Technologies LLC ("Bradium") has asserted the '794, '343, '506, and '239 Patents against Microsoft in an on-going patent infringement lawsuit, No. 1:15-cv-00031-RGA, filed in the U.S. District Court for the District of Delaware on January 9, 2015 and

amended to add the '239 Patent on March 14, 2016. I understand that the '644, '645, and '136 Patents have not yet been asserted in litigation against Microsoft.

4. I understand that the '645 Patent was purportedly assigned to Bradium. Bradium is therefore referred to as the "Patent Owner" in this declaration.

5. In this declaration, I will first discuss the technology background related to the '645 Patent and then provide my analyses and opinions regarding claims 1-36 of the '645 Patent. The discussion of the technology background includes an overview of that technology as it was known before December 2000, which I understand as the earliest priority date claimed by the '645 Patent. This overview provides some of the bases for my opinions with respect to the '645 Patent.

6. This declaration is based on the information currently available to me. To the extent that additional information becomes available, I reserve the right to continue my investigation and study, which may include a review of documents and information that may be produced, as well as testimony from depositions that may not yet be taken.

7. In forming my opinions, I have relied on information and evidence identified in this declaration, including the '645 Patent, the prosecution history of the '645 Patent, and prior art references listed as Exhibits to the Microsoft IPR

Petition and listed as appendices of this declaration. The Appendices to this declaration include a number of references known to those in the art to describe technical concepts relevant to the subject matter of this declaration, and include (for example) patents, technical publications, and industry standards. In my opinion, an expert or a person of ordinary skill in the art in the subject matter relevant to this declaration would consider each of the Appendices to this declaration relevant to the subject matter of this declaration and would reasonably rely on such materials to form an opinion as to the state of the art prior to December 27, 2000, the interpretation of the prior art references relied upon in Microsoft's petition, and the obviousness of the claims challenged in the petition. I have also relied on my own personal experience in the field of computer graphics, which includes the design and development of computer graphic hardware, software, and display systems.

II. SUMMARY OF OPINIONS

8. Claims 1-36 of the '645 Patent relate to a system and method for dynamic visualization of image data transferred through a communications channel. For the reasons explained below, none of the features described in Claims

1-36 of the '645 Patent were novel as of either October 1999 or December 2000,[1] nor does the '645 Patent teach a novel and non-obvious way of combining these known features.

9. Claims 1-36 of the '645 Patent relate to well-known technologies in the computer industry such as multi-resolution hierarchical maps, image compression, packetized data transmission, and three-dimensional (3D) graphics rendering. No element of Claims 1-36 is novel, and Claims 1-36 do not bring these elements together in a way that brings any benefit beyond what a person of ordinary skill in art would expect from the known functions of the individual

[1] I understand that the inventors alleged during the prosecution of U.S. Patent No. 7,644,131 that "the herein invention was first defined in October 1999." *See, e.g.* IPR2016-00448, Ex. 2064. However, this statement related to a different application and no corroboration was provided for the assertion of this date. I refer to this date only because it is the earliest invention date which I am aware of 3DVU or Bradium having asserted. Nothing in this declaration should be taken as an admission that the subject matter claimed in the '645 Patent was actually invented on this date, and I reserve the right to offer rebuttal testimony if Bradium seeks to argue or present evidence of an invention date prior to the effective filing date for any claim.

components. Claims 1-36 describe techniques that were well-known in the field, and combine them in ways that would have been readily apparent to a person of ordinary skill in the art with predictable results.

10. It is my opinion that each of Claims 1-36 is invalid under the patentability standard of 35 U.S.C. § 103 as I understand it and as explained to me by Microsoft's counsel. Within this declaration I discuss specific grounds of invalidity of Claims 1-36; however, my opinion that Claims 1-36 are invalid under 35 U.S.C. § 103 is not limited to these specific grounds, and indeed, it is my opinion that Claims 1-36 would have been invalid in light of the general knowledge of a person of ordinary skill in the art at the time of the alleged invention.

11. For purposes of my analyses in this declaration only, I provide my proposed construction of certain terms in Claims 1-36 in detail in a later part of this declaration.

12. The subsequent sections of this declaration will first provide my qualifications and experience and then describe details of my analyses and observations.

III. QUALIFICATIONS AND EXPERIENCE

A. Education and Work Experience

13. I received a Ph.D. degree in Electrical Engineering in 1989 and a Master of Science degree in Electrical Engineering in 1985 from the Worcester Polytechnic Institute. I received a Bachelor of Science degree in Electrical Engineering from Syracuse University in 1981.

14. I have more than twenty years of experience in the fields of electrical engineering, computer systems, navigation systems, and communications systems. My experience includes the design, implementation and use of geographic information systems ("GIS"), as well as the design, implementation and use of navigation systems relying on GPS and other positioning system technologies. I also have extensive experience in computer communication and data processing systems as well as systems for the efficient transmission of digital images and other data. Additionally, I have experience in the design and implementation of hardware and software systems used to render image data for display.

15. I have published 16 papers in technical journals and 97 papers in technical conferences. I hold eight U.S. patents in the fields of handheld GPS (Global Positioning System), portable geolocation devices, and communication networks. I have also authored one book chapter relating to optical interconnect

networks for massively parallel computers. I became a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE) in 2003.

16. My experience spans from product designs and R&D in industry, teaching, research and development in an educational and research institution to technology consulting to industry. I was an engineer at Raytheon Company for ten years from 1981 to 1991. During this period, I worked on projects related to computer display hardware for various applications, including air traffic control applications.

17. After leaving Raytheon Company, I joined the Worcester Polytechnic Institute and became a full-time faculty member there in 1991. My research at WPI focuses on navigation systems and related technologies. I am the director of WPI's Robot Navigation and Control Laboratory.

18. My research projects at WPI cover various technologies and include (1) a system using tracking and communications technologies to track shipping containers, (2) an automotive based system that combined GPS and map data in an automotive environment, (3) a remote hazard detection system using GPS and radio communications, and (4) a differential GPS system that combined GPS and radio technologies to determine the precise path of vehicles operating off-road during forest operations.

19. I have worked as a consultant in the navigation and communication systems fields, *e.g.*, in the context of space shuttle docking operations, transfer of traffic information to GPS devices, combinations of GPS and cellular communications for tracking purposes, and map-based handheld tracking devices.

20. I am familiar with numerous GIS and mapping products that existed in the market since the late 1980s, including systems and software developed by Etak, Microsoft, DeLorme, and others. In the conduct of my research and other work, I have routinely used commercially available GIS and mapping products and have developed mapping and visualization software for specialized applications. Additionally, I have used and incorporated database systems such as Microsoft Access, Borland Paradox, Oracle, SQL and others in my research and have incorporated database systems into other hardware and software systems for use in storing and retrieving GIS-related data.

21. I have done extensive research work in communications and networking system design, and have worked with all of the digital, analog and software components needed to build communications and navigation systems. My work with communications and networking protocols began in the mid-1980s with TCP/IP over packet radio. I have used these and other communications and networking protocols extensively in conducting my research. In addition, my work on GPS and navigation systems involved implementing low-latency

communications to support differential techniques that allow a GPS receiver to provide more accurate positioning information.

22. I have extensive experience with the development and maintenance of server computers, including the installation and maintenance of web servers and file servers, as well as the design, development, test, and maintenance of web based applications. These applications typically employ C/C++, Java, JavaScript, PHP, HTML, MySQL, and etc. I am also experienced with server-client systems where the client computer exchanges navigation and/or geographical information with server computer through a wired and/or wireless network.

23. My curriculum vitae, which provides a detailed summary of my education, work experience, publication, teaching history, and etc. is attached to this declaration as Appendix A.

B. Compensation

24. I am being compensated for the services I am providing in this and other Microsoft IPR petitions. The compensation is not contingent upon my performance, the outcome of this *inter partes* review or any other proceedings, or any issues involved in or related to this *inter partes* review or any other proceedings.

C. Documents and Other Materials Relied Upon

25. The documents on which I rely for the opinions expressed in this declaration are documents and materials identified in this declaration, including the '645 Patent, patents related to the '645 Patent, the prosecution histories for the '645 Patent and other patents related to the '645 Patent, the prior art references and information discussed in this declaration, including the references attached as exhibits to the IPR Petition for the '645 Patent: *TerraVision II: Visualizing Massive Terrain Databases in VRML* by M. Reddy *et al*., IEEE Computer Graphics and Applications, March/April 1999 (Ex. 1004), U.S. Patent 5,956,039 to Woods *et al.* ("Woods") (Ex. 1003), U.S. Patent 7,324,228 B2 to Chiarabini *et al.* ("Chiarabini") (Ex. 1006); *The MAGIC Project: From Vision to Reality* by Barbara Fuller *et al.*, IEEE Network, May/June 1996 ("Fuller") (Ex. 1011), and any other references specifically identified in this declaration, in their entirety, even if only portions of these documents are discussed here in an exemplary fashion. I have also considered certain arguments made by Bradium and its hired experts, including Dr. Peggy Agouris, in IPRs of related patents, which do not change my opinion that the claims of the '645 Patent are obvious.[2]

[2] I understand that Microsoft's burden of proof for institution of an IPR does not extend to rebutting every possible counter-argument, so I will not discuss every

IV. STATEMENT OF LEGAL PRINCIPLES

A. Claim Construction

26. Microsoft's counsel has advised that, when construing claim terms of an unexpired patent, a claim subject to *inter partes* review receives the "broadest reasonable interpretation in light of the specification of the patent in which it appears."

B. Anticipation

27. Microsoft's counsel has advised that in order for a patent claim to be valid, the claimed invention must be novel. Microsoft's counsel has further advised that if each and every element of a claim is disclosed in a single prior art reference, then the claimed invention is anticipated, and the invention is not patentable according to pre-AIA 35 U.S.C. § 102 effective before March 16, 2013. In order for an invention in a claim to be anticipated, all of the elements and limitations of the claim must be shown in a single prior reference, arranged as in the claim. A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art

---

argument previously made by Bradium or its expert in previous IPRs. However, I reserve the right to offer testimony in rebuttal to any arguments or evidence submitted by Bradium in this proceeding.

reference. In order for a reference to inherently disclose a claim limitation, that claim limitation must necessarily be present in the reference.

C. Obviousness

28. Microsoft's counsel has also advised me that obviousness under pre-AIA 35 U.S.C. § 103 effective before March 16, 2013 is a basis for invalidity. I understand that where a prior art reference does not disclose all of the limitations of a given patent claim, that patent claim is invalid if the differences between the claimed subject matter and the prior art reference are such that the claimed subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the relevant art. Obviousness can be based on a single prior art reference or a combination of references that either expressly or inherently disclose all limitations of the claimed invention. In an obviousness analysis, it is not necessary to find precise teachings in the prior art directed to the specific subject matter claimed because inferences and creative steps that a person of ordinary skill in the art would employ can be taken into account.

29. I understand that obviousness is not driven by a rigid formula, but is a flexible inquiry that reflects the fact that a person of ordinary skill in the art exercising ordinary creativity may find a variety of reasons to combine the teachings of different references. I understand that a non-exclusive list of possible factors that may give a person of ordinary skill in the art a reason to combine

references includes combining elements according to known methods to yield predictable results; simple substitution of known elements to obtain predictable results; use of known techniques to improve similar devices in the same way; applying known techniques to known devices ready for improvement to yield predictable results; choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success; known work in one field of endeavor prompting variations of it for use in the same field; and teaching in the prior art that would have led one of ordinary skill to combine prior art reference teachings to arrive at the claimed invention.

V. LEVEL OF ORDINARY SKILL IN THE ART

30. I understand from Microsoft's counsel that the claims and specification of a patent must be read and construed through the eyes of a person of ordinary skill in the art at the time of the priority date of the claims. I have also been advised that to determine the appropriate level of a person having ordinary skill in the art, the following factors may be considered: (a) the types of problems encountered by those working in the field and prior art solutions thereto; (b) the sophistication of the technology in question, and the rapidity with which innovations occur in the field; (c) the educational level of active workers in the field; and (d) the educational level of the inventor.

31. The "Background" section of the '645 Patent describes a "well recognized problem" of how to reduce the latency for transmitting full resolution images over the Internet on an "as needed" basis, particularly for "complex images" such as "geographic, topographic, and other highly detailed maps." Ex. 1001 at 1:55-63.

32. To solve this problem and to address some perceived issues in the existing art, the '645 Patent discloses a system capable of "optimally presenting image data on client systems with potentially limited processing performance, resources, and communications bandwidth." *Id.* at 3:63-67. The '645 Patent states that the disclosed technology can achieve faster image transfer by (1) dividing the source image into parcels/tiles, (2) processing the parcels/tiles into a series of progressively lower resolution parcels/tiles, and (3) requesting and transmitting the parcels/tiles needed for a particular viewpoint in a priority order, generally lower-resolution tiles first.

33. In light of the disclosed technology of the '645 Patent, a person of ordinary skill in the art for the '645 Patent would need education or work experience in computer network communications. Because a "common application" of the '645 Patent is to transmit "geographic, topographic, and other highly detailed maps," (*id.* at 1:58-60), a person of ordinary skill in the art would

require some knowledge and experience with geographic information systems ("GIS").

34. Based on the above considerations and factors, it is my opinion that a person having ordinary skill in the art should have a Master of Science or equivalent degree in electrical engineering or computer science, or alternatively a Bachelor of Science or equivalent degree in electrical engineering or computer science, with at least 5 years of experience in a technical field related to geographic information system ("GIS") or the transmission of image data over a computer network. This description is approximate and additional educational experience could make up for less work experience and vice versa.

35. I understand that in IPR proceedings involving related patents, Bradium and its expert have offered a definition of the level of ordinary skill in the art which differ from mine. *See, e.g.* IPR2016-00448, Paper 20 at 8 and Ex. 2003, ¶¶ 15-19. As I explained in those related proceedings, Bradium's position regarding the level of ordinary skill in the art was incorrect.

36. For example, Bradium argued that my proposed level of ordinary skill in the art was incorrect because one of the named inventors, Mr. Levanon, would not have met my definition of the level of ordinary skill in the art. Bradium also asserted that the other named inventor, Mr. Lavi, also would not qualify as a person of ordinary skill in the art. I disagree.

37. First of all, I understand that a person of ordinary skill in the art is not a specific individual, but a hypothetical individual at the time of the alleged invention who is familiar with the relevant art in the field and is capable of making reasonable inferences from that art, in addition to being a person of ordinary creativity. If an alleged invention is not in fact novel but simply applies principles that were well-known in the art with predictable results, as is the case with the '645 Patent, it is certainly possible that the named inventors might have less education and experience than a hypothetical person of ordinary skill in the art.

38. Additionally, based on my review of Mr. Levanon's linkedin profile (Ex. 1015), it does not appear as though Mr. Levanon would meet Bradium and Dr. Agouris' proposed definition of a person of ordinary skill in the art, because Mr. Levanon does not have a four-year degree or equivalent in any of the fields of art identified by Bradium. *See also* Ex. 1019 (Levanon Deposition Transcript) at 31:19-22.

39. However, having considered the proposed level of ordinary skill previously offered by Bradium and Dr. Agouris, it is my opinions that I offer in this declaration would not change if Bradium's proposed level of ordinary skill were applied.

40. My conclusions would not change if the level of ordinary skill in the art were assessed in October 1999, which was the earliest invention date asserted

by Bradium during the prosecution of related patents, or in December 2000, when the earliest applications to which the '645 Patent claims priority were filed.

VI. TECHNOLOGY BACKGROUND OF THE '645 PATENT

41. It is my opinion that the '645 Patent recites an obvious and predictable combination of elements that were well-known in the art at the time the '645 Patent was filed and at the time of alleged invention. In this section of my declaration, I provide an overview of some general principles that were understood in the art at the time of filing of the '645 Patent, and therefore would be within the knowledge of a person of ordinary skill in the art. I use certain references (including both patents and non-patent literature) to illustrate the background knowledge of a person of ordinary skill in the art, but the knowledge of a person of ordinary skill in the art at the time regarding the claimed features would not have been limited to these specific references.

42. The '645 Patent recites that the "preferred operational environment of the present invention is generally shown in Fig. 1" and links a network server with a client system "through a communications network, such as the Internet 14 generally and various tiers of Internet service providers (ISPs) including a wireless connectivity provider." Ex. 1001, 5:44-64; Fig. 1:

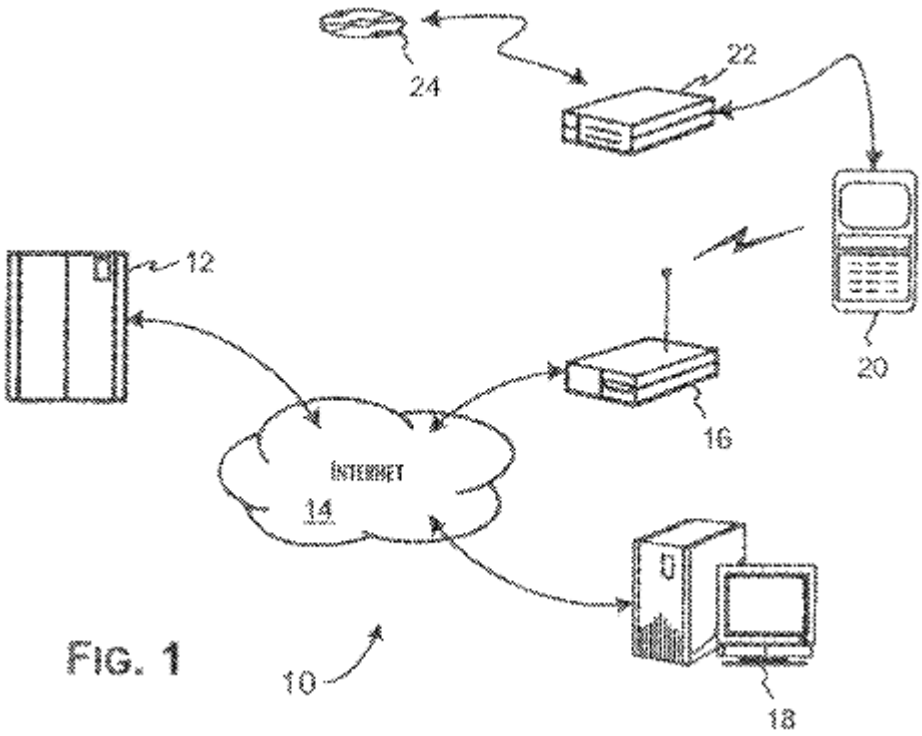


FIG. 1

43. Based on my review of the entire specification of the ’645 Patent, it appears to me that the inventors describe a system that relied on conventional network connections, including conventional wireless networking methods, and that the underlying means of transmitting data over the Internet or over a wireless network are not emphasized as a point of novelty. In other words, in order to implement the alleged invention in the ’645 Patent, a person of ordinary skill in the art would have to rely on existing methods already known in the art of connecting to the Internet and sending data over a wireless connection, since the ’645 Patent does not provide any novel teachings about this aspect of the alleged system. This fact is particularly relevant to certain claim limitations which relate to, *e.g.*, the

bandwidth of the communications channel, whether the communications channel is wireless, and the type of client device which operates the '645 Patent's user software, because the ability to connect to the Internet, or connect to the Internet via a wireless channel or on a "small" client device such as a PDA, is something that the '645 Patent assumes that a person of ordinary skill in the art would already know how to do. I considered this relevant to my analysis later in this declaration that these claim limitations are obvious over the references discussed and that a person of ordinary skill in the art would have a reasonable expectation of success implementing the system described by Reddy (which itself describes a laptop computer) on, for example, a Personal Digital Assistant (PDA) or via a wireless connection.

44. I provide below a general description of the underlying technology of transmitting data over the Internet and via wireless connections as it existed in 2000 and before.

A. Data Communications Over the Internet

45. The predominant computer networking technology and set of communications protocols used for most online communications today and prior to the filing of the application for the '645 Patent is known as the Internet Protocol (IP) suite including TCP/IP, named after its two main component protocols: the Transmission Control Protocol and the Internet Protocol. While other protocols,

such as the User Datagram Protocol, or UDP, are also part of the IP suite of protocols, the '645 Patent teaches at 8:28-49 that its preferred embodiment uses TCP/IP to send data packets. In this declaration I do not provide a detailed description of all characteristics of the very well-known TCP/IP protocols, but focus on a few specific aspects of TCP/IP that are pertinent to the claims at issue in the '645 Patent. TCP/IP transmits data between computers in a network using data packets, which are formatted units of data carried by the network as suitably sized blocks. Packets are composed of a header and a payload. The "payload" is the information which the packet is actually intended to convey. The header refers to supplemental data placed at the beginning of a block, and contains information in a standard format such as the sender's and recipient's Internet Protocol addresses, a sequence number indicating where in a sequence of packets being transmitted the packet falls, an offset (how far into the data the payload begins) and the protocol governing the format of the payload. The addresses are used to route the packet to its destination, although unlike a circuit-switched connection, different packets going between the same sender and recipient at the same time may take different routes over the network. A rough analogy for data packets is that the header is the "envelope" which contains the address used to deliver the packet, while the payload is the contents of the envelope. The destination computer uses information in the header to place the data packet in its proper place in order, from

which the original data contained in multiple packets can be reassembled. When data segments arrive in the wrong order TCP/IP buffers the out-of-order data until all data can be properly re-ordered and delivered to the application.

46. Before data is transmitted using TCP/IP, the sender and the destination exchange a short series of messages confirming a connection. The connection in this case simply means that the sender and the destination exchange messages to confirm that they are able to exchange messages via the network. When the destination computer receives a packet, it sends a short confirmation message to the sender that the packet has been received. If the confirmation is not received within a certain time period, the sender re-sends the packet. This method avoids losing data in transmission if the transmission of a single packet fails.

47. A common consideration in building online systems is how large to make the data packets. Among other trade-offs, smaller packets may be more likely to reach their destination without loss or error; however, because the header size is similar for a large packet and a small packet, the amount of bandwidth taken up by header overhead increases with the use of smaller packets. In general, there are a variety of packet sizes that may be used in the transfer of a TCP/IP packet from source to destination since lower-level network protocols may have limitations on packet length based on the physical layer or other requirements.

Typically, the network protocols used at any given layer in a protocol stack set the minimum and maximum packet lengths that may be transferred.

B. Data Communications in Wireless Mobile Systems

48. By the late 1990s, it was well-known in the art that digital data could be transmitted by TCP/IP over wireless networks. For example, Appendix X, "The challenges of mobile computing," Computer vol. 27, no. 4, pp. 38, 47 (April 1994) provides an overview of methods for implementing internet connections on mobile devices as of 1994, noting that while wireless networks typically deliver lower bandwidth than wired networks, cellular telephone products of the time could already achieve transmission rates of 9-10 kilobits per second.

49. In another example in 1996, K. Brown and S. Singh, *A Network Architecture for Mobile Computing*, INFOCOM '96, Fifteenth Annual Joint Conference of the IEEE Computer Societies, Networking the Next Generation, Proceedings IEEE vol. 3, pp. 1388-1397 (Appendix Y) describes technologies that integrate wireless or mobile networks with existing fixed data networks such as the Internet. This integration was described by using mobile data protocols to interact and be compatible with the TCP/IP structure of the Internet. This paper described how the Universal Mobile Telecommunications System (UMTS) under development in Europe was expected to offer average mobile data rates of between 1-2 megabytes per second (Mbps) per mobile user. Among other features, "Mobile

users will be able to access their data and other services such as… map services." App. Y at 2.

50. Appendix Z, Kreller, B. et al "UMTS: a middleware architecture and mobile API approach," Personal Communications, IEEE, vol. 5, no. 2, pp. 32-38 (April 1998) describes the development of third-generation (3G) mobile networks offering "high-bit-rate data services, guaranteed on-demand bandwidth, and low delays." *Id.* at 32. To illustrate the development of frameworks to connect mobile telephone networks with existing fixed networks, the authors use the example of a mapping service called "City Guide," which allows a mobile device to request and download map imagery and other data from a server via hypertext transfer protocol (HTTP) to "provide access to maps describing the current surroundings." *Id.* at 33. The CityGuide could use JPEG compression and decompression, and could achieve bandwidth of up to 9.6 kbps using the then-existing Global System for Mobile Communications (GSM) cellular data standard. *Id.* at 36, 37. Further, the CityGuide system is an early example of the use of "[w]eb pages to allow instant access to further information about a particular location" by using a browser on a mobile device. *Id*. At 33.

C. Image Tiles and Image Pyramids

51. The '645 Patent describes sub-dividing a high resolution source image into a regular array of image parcels (a.k.a. image tiles), and pre-processing the

image into a series of derivative images of progressively lower resolutions. Ex. 1001 at 6:23-28; Fig. 2. Preferably, the resolution decreases by a factor of four for each derivative image in the series. *Id.* at 6:34-38. Fig. 2 of the six provisional applications to which the '645 Patent claims priority (which is identical in all six provisional applications) best illustrates this image tiling and image pyramid scheme. Ex. 1021 at Fig. 2.

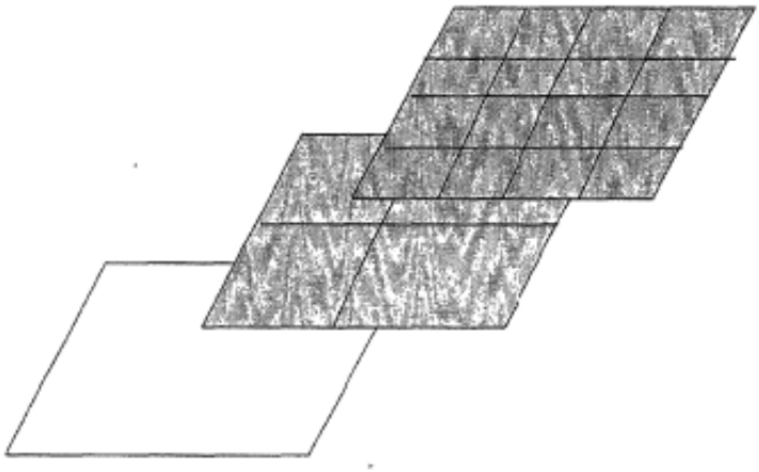


FIG.2

52. This image processing scheme, however, had been developed and widely used long before the '645 Patent's priority date. For example, Hanan Samet's book *The Design and Analysis of Spatial Data Structures* discloses generating an image "pyramid" from a $2^n x 2^n$ image array, where the pyramid is "a sequence of arrays {A(i)} such that A(i-1) is a version of A(i) at half the scale of A(i)." App. B, Hanan Samet, *The Design and Analysis of Spatial Data Structures*

at 12 (1989, Reprinted with corrections in January, 1994). Fig. 1.7 in the Samet book is virtually the same as Fig. 2 of the '645 Patent's provisional application.

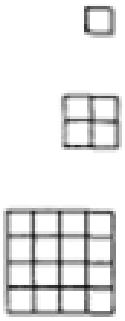
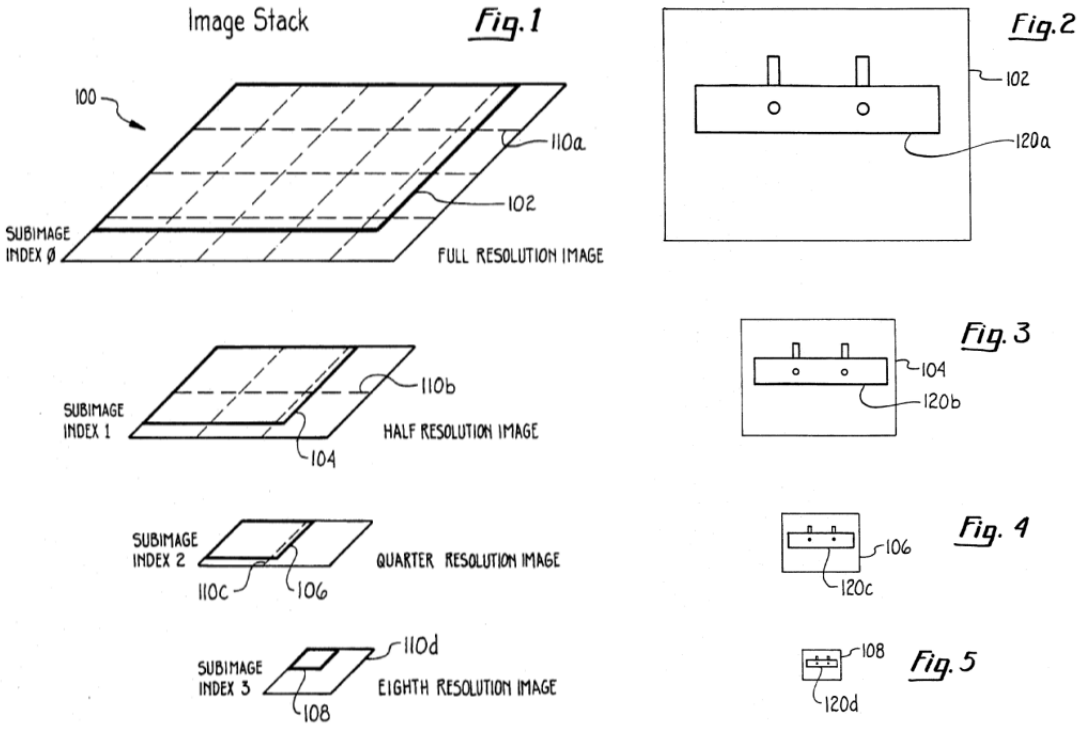


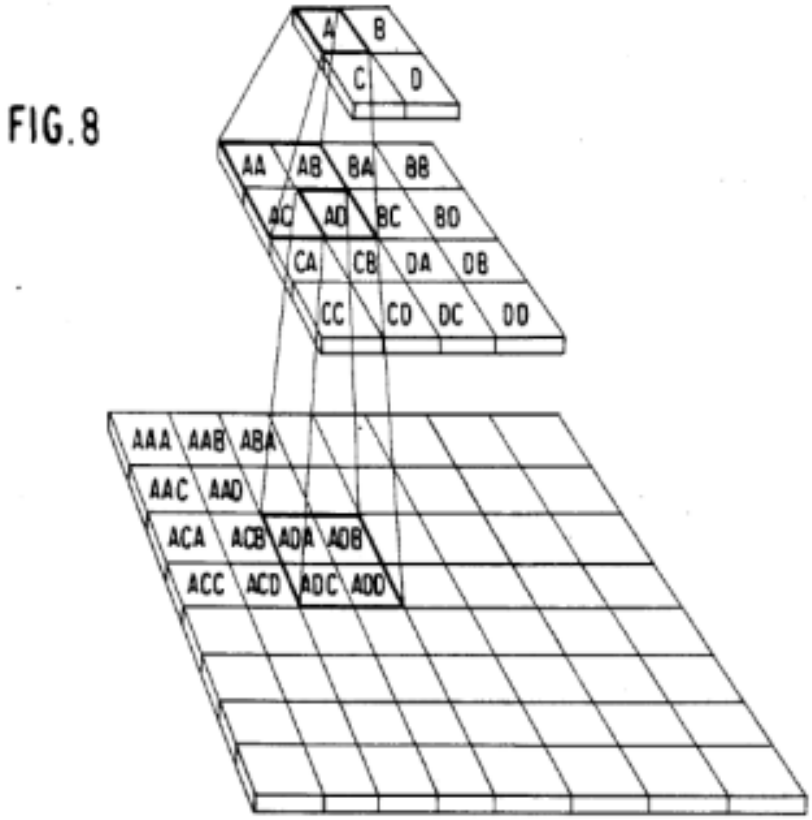
Figure 1.7 Structure of a pyramid having three levels

53. In another example, U.S. Patent No. 5,263,136 (DeAguiar *et al*) filed on April 30, 1991 and issued on November 16, 1993, entitled *System for Managing Tile Images using Multiple Resolutions*, discloses an "image memory management system for tiled images," where "each source image is stored as a full resolution image and a set of lower-resolution subimages." App. C at Abstract; Figs. 1 and 2. Suitable applications of the DeAguiar patent's image tiling and image pyramid scheme include "electrical schematics, topographical maps, satellite images, heating/ventilating/air conditioning (HVAC) drawings, and the like." *Id.* at col. 6:65-7:2.



54. U.S. Patent 4,972,319 to Delorme, filed on Sept. 25, 1987 and issued on Nov. 20, 1990, also showed that image tiling and image pyramids can be used in mapping applications. Specifically, the Delorme patent discloses a "global mapping system which organizes mapping data into a hierarchy of successive magnitudes or levels for presentation of the mapping data with variable resolution, starting from a first or highest magnitude with lowest resolution and progressing to a last or lowest magnitude with highest resolution." Ex. 1065 at Abstract. A pyramid of successively lower resolution image tiles is shown in Fig. 8 of the Delorme 319 patent. *Id.* at Fig. 8.

FIG. 8



55. In yet another example, a 1996 article entitled "The MAGIC Project: From Vision to Reality" by Barbara Fuller and Ira Richer ("Fuller") also shows the image tiling and image pyramid scheme for mapping applications. Ex. 1011 at Fig. 3.

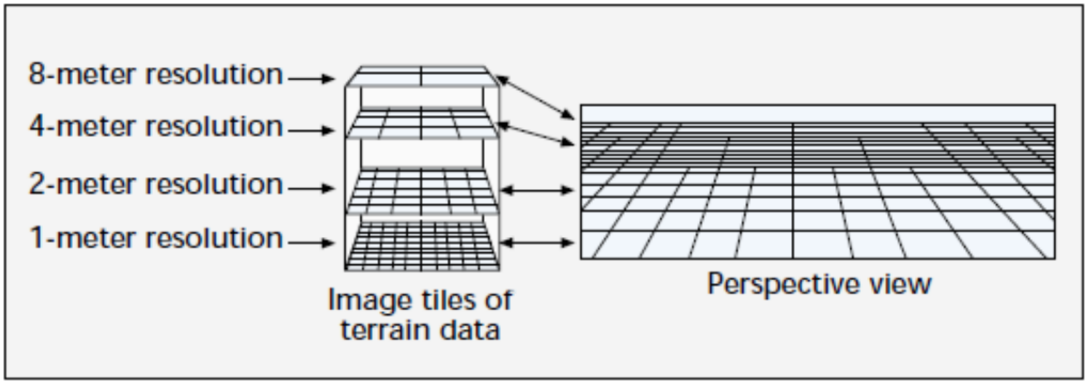


Figure 3. *Relationship between tile resolutions and perspective view. (Source: SRI International)*

56. Microsoft itself used a multi-resolution tiling system in an online mapping application, TerraServer, starting in the mid 1990s. *See, e.g.* Barclay *et al*, "Microsoft TerraServer: A Spatial Data Warehouse," Microsoft Technical Report MS-TR-99-29, June 1999, Revised February 2000 (Ex. 1030) ("Barclay").[3] TerraServer stored "aerial, satellite, and topographic images of the earth in a SQL database available via the Internet." Barclay, Abstract. TerraServer stored image data in the database in JPEG or GIF format in a "pyramid" of image tiles at varying resolutions. *Id.*, § 2.1. Source imagery is first cut into high-resolution tiles using a "TerraCutter" program:

[3] Also available online at https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/msr_tr_99_29_terraserver.pdf (accessed September 29, 2016).

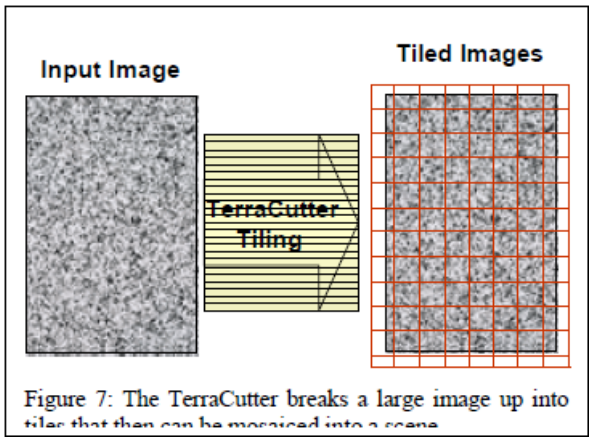


Figure 7: The TerraCutter breaks a large image up into tiles that then can be mosaiced into a scene

57. Barclay, Fig. 7. Once the high-resolution tiles have been generated, "[f]our higher resolution tiles are sub-sampled onto one lower resolution tile," which provess is "repeated for the number of levels in the image hierarchy." *Id.* § 2.1. Tiles are requested using HTTP protocol by a "thin-client" graphical web browser. Fig. 9 of Barclay shows how TerraServer down-samples image tiles into a hierarchy or image pyramid of tiles at varying resolutions:

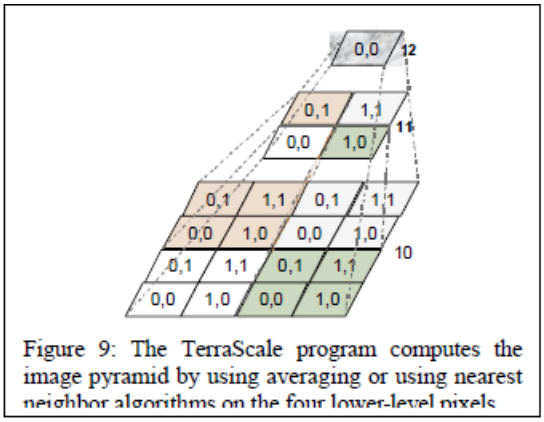


Figure 9: The TerraScale program computes the image pyramid by using averaging or using nearest neighbor algorithms on the four lower-level pixels

See also id., § 3.4.

58. I was personally familiar with the TerraServer system around the time that it was released in 1998 because I personally used the system at the time and it

was widely discussed in the GIS community after it became very popular very quickly. The description in Ex. 1030 is consistent with my own recollection at the time of how the TerraServer operated. I have also considered a number of additional contemporaneous documents describing TerraServer, including the original 1999 version of Barclay (Ex. 1032), a version of Barclay that was published in the proceedings of the ACM SIGMOD (Association for Computing Machinery's Special Interest Group on Management of Data) in 2000 (Ex. 1035), an earlier 1998 Microsoft White Paper on Terra Server (Ex. 1034, with Cornell University Library submission record as Ex. 1033), a 1998 Microsoft White Paper on TerraServer (Ex. 1036) and several Internet Archive captures from the late 1990s and 2000 of the technical information on the TerraServer page (Exs. 1037-1042). All of this information is consistent with and supports my understanding of the operation of the TerraServer program which I discussed in regard to Ex. 1030. Additionally, all of these documents were contemporaneously generated descriptions of TerraServer that long pre-date the litigation in this case. Therefore, it is my opinion that a reasonable person in the field of computer science, whether an expert or a person of ordinary skill in the art, would rely on these exhibits as

indicative of what was known in the art of online Geographic Information Systems prior to the alleged invention date and effective filing date of the '645 Patent.[4]

59. Bradium also previously mischaracterized a portion of Ex. 1030 in other IPRs relating to other Bradium patents. For example, in a "Motion for Observations" on my deposition testimony in IPR2016-00448, Bradium quoted a portion of Ex. 1030 saying that the solution described in Ex. 1030 "had not been attempted before" and that "many people felt it was impossible without using an object-oriented or object-relational system" as somehow relevant to Bradium's "argument that VRML is essentially a set of objects that are linked to one another." IPR2016-00448, Paper 44, Observation No. 3; *see also* Appendix NN at 52:1-12. This argument is such a complete non sequitur that it is difficult to even make sense of what Bradium was trying to argue. Simply put, the quoted portion of Ex. 1030 had nothing to do with VRML. Since Bradium's position here was incoherent, it is more straightforward for me to explain what the quoted portion of Ex. 1030 actually said. Microsoft's TerraServer project was originally developed by a research group focused on "scaleable servers" who wanted to demonstrate a

[4] Naturally, some of these exhibits were *accessed* from the Internet more recently and contain additional markings reflective of when they were accessed, but the documents themselves are much older than that.

"large Internet server with a large database and heavy web traffic," and in order to generate such traffic they "needed to build an application that would be interesting to millions of web users." One of the reasons that the authors describe for utlizing a large scale geographic database as the test case is that:

> The solution as we defined it - a wide-area, client/server imagery database application stored in a commercially available SQL database system - had not been attempted before. Indeed, many people felt it was impossible without using an object-oriented or object-relational system.

Ex. 1030 at 2-3.

60. Read in context, this portion of the specification is discussing how creating a large-scale commercially available application based on commonly used SQL databases made TerraServer a good application for demonstrating scalable server concepts. This has nothing to do with VRML. Of course, it is also true that VRML at the time was capable of using mip-mapped textures, including geographically linked images (indeed, this is a key teaching of Reddy, which is illustrated in Figures 1 and 3, for example), but that is not specifically what Ex. 1030 was discussing. To the extent that Bradium may raise similar arguments in this proceeding, particularly if it does so in a more coherent manner, I may choose to respond in more detail.

D. Compression of Image Tiles

61. The ’645 Patent discusses that the image tiles can preferably be compressed, *e.g.*, for a fixed compression ratio of 4:1. Ex. 1001 at 6:39-44. Numerous methods existed, however, long before the ’645 Patent’s priority date, to compress images for either a variable or fixed ratio.

62. One widely used method of digital image compression is JPEG compression, which is based on the Discrete Cosine Transform (“DCT”), and is described in the International Telegraph and Telephone Consultative Committee (“CCITT”) Recommendation T.81 published in September 1992 (App. F). JPEG compression includes the following main steps: “1. The image is broken into 8x8 blocks of pixels. 2. Working from left to right, top to bottom, the DCT is applied to each block. 3. Each block is compressed through quantization. 4. The array of compressed blocks that constitute the image is stored in a drastically reduced amount of space. 5. When desired, the image is reconstructed through decompression, a process that uses the Inverse Discrete Cosine Transform (IDCT).” App. G at 1, Ken Cabeen & Peter Gent, *Image Compression and the Discrete Cosine Transform*.

63. Another widely used method of digital image compression is based on the wavelet transform. For example, Marc Antonini et al.’s 1992 paper *Image Coding Using Wavelet Trans*form discloses a scheme for image compression using

the wavelet transform. *See generally* App. H. In addition, the Antonini paper shows that image compression using wavelet transform not only achieves a good image quality (*id*. at 217-18), but is also suitable for a progressive transmission scheme to "allow the receiver to recognize a picture as quickly as possible at minimum cost." *Id.* at 218-19.[5]

64. The JPEG 2000 image compression standard, which was designed as the next version of the JPEG Standard to address its identified problems, uses discrete wavelet transform.

[5] I understand that Bradium has mischaracterized my discussion of the use of wavelet transformation in previous IPRs as somehow teaching away from the use of progressive resolution enhancement (a term which does not appear in the claims of the '645 patent) by transmitting successive resolution levels. Simply put, this is wrong because the two things have little or nothing to do with each other. Many forms of compression were well known prior to the alleged invention and effective filing dates of the '645 Patent which could be used either to compress individual images or "tiles," or to progressively send differential images to form a complete image, and neither taught away from the other.

E. Progressive Image Resolution Enhancement

65. The progressive image resolution enhancement technique described and claimed in the '645 Patent is one of the "conventional" solutions that have been used to reduce the latency of transmitting complex images over a communications network, as admitted in the "Background of the Invention" section of the '645 Patent. Ex. 1001 at 2:5-15 ("Different conventional systems have been proposed to reduce the latency affect by transmitting the image in highly compressed formats that support **progressive resolution build-up of the image within the current client field of view**. . . . **Progressive image resolution transmission**, typically using a differential resolution method, permits an approximate image to be quickly presented with image details being continuously added over time.") (emphasis added).

66. For example, U.S. Patent No. 5,321,520 to Inga et al., filed July 20, 1992 and issued June 14, 1994, discloses a "Progressive Image Enhancement" ("PIE") method, where a "'crude' image is presented to the subscriber" first and then the method "progressively enhance[s] the quality of the presented image" over time. App. I at col. 12:65-13:1. "The longer the user observes a selected image, the 'better' the image becomes in the sense of pixel resolution and quantity of gray levels." *Id.* at col. 13:1-3.

67. U.S. Patent No. 6,182,114 to Yap et al. was filed January 9, 1998 and issued January 30, 2001. The "Background of the Invention" section of the '645 Patent mentions Yap. The Yap Patent recognizes that "progressive transmission" is an existing approach to solve the problem of "realtime visualization of large scale images over a 'thinwire' model of computation," *i.e.*, over a "low bandwidth line." App. J at 1:47-65. In addition to the traditional progressive transmission method, where the higher resolution of the entire image will be eventually transmitted, the Yap Patent discloses an improved version of progressive transmission, where "resolution is also varied over the physical extent of the image." *Id.* at 2:4-17. Specifically, the Yap Patent discloses that "high resolution data is transmitted at the user's gaze point but with lower resolution as one moves away from that point." *Id.* at 2:18-20. The same scheme is used in the '645 Patent.

68. Early web browsers supported a form of progressive resolution enhancement as well. For example, Netscape supported a "lowsrc" attribute for HTML <IMG> tags that allowed web pages to specify low resolution images to be downloaded and displayed temporarily until the browser fetched higher resolution images for the page. *See, e.g.*, Ex. 1053 at 135-136; Ex. 1054 at 791-792.

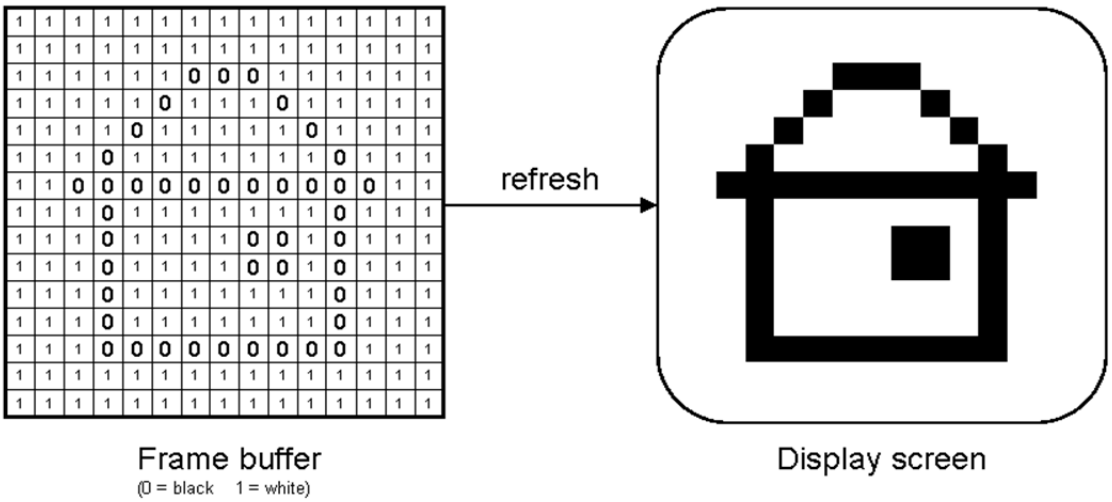
F. Three-Dimensional Graphics

1. Overview of 3D Computer Graphics principles

69. The "field" section of the '645 Patent notes that the claimed invention is designed "to support presentation of high-resolution images subject to dynamic viewing frustums." Ex. 1001, 1:46-51. The term "frustum" is used in computer graphics to refer to the field of view of a three-dimensional image, and is analogous to the view through a viewfinder of a real-world camera, except that the "camera" is notional and is simply the basis for the calculations that the computer does to render the image. In order to understand this aspect of the '645 Patent and how the prior art relates to it, I discuss below certain concepts in the field of computer graphics. Computer graphics is the art and science of drawing pictures on a display screen using a computer. A picture generated using computer graphics is created from numerical data describing the objects to be drawn. Computer graphics is generally divided into 2D ("two-dimensional") graphics that only depict images in two dimensions and 3D ("three-dimensional") graphics that depict images in three dimensions, although by way of representing them on a 2D screen.

70. An image that shows up on a computer display typically corresponds to a large, rectangular, two-dimensional array of values in a computer memory called a frame buffer. An individual location in the frame buffer can hold a color

value corresponding to one "dot" or picture element, or pixel for short, on the display screen. In the simple example shown below, the values in the frame buffer at each pixel are either 0 or 1, which get displayed as black and white, respectively, on the screen.

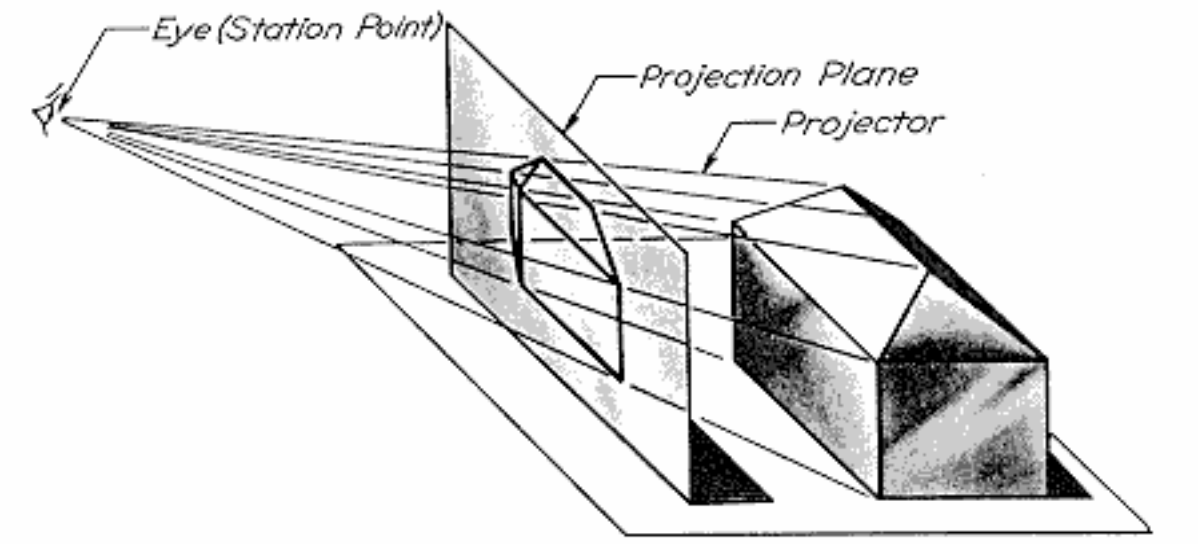


71. In display systems, color values at each pixel are usually either represented by a single number representing shades of gray, or by 3 numbers, R, G, and B, corresponding to red, green, and blue intensity values, for each location on the screen. The computer display is generated by repeatedly "scanning out" the array of numerical pixel values from the frame buffer memory in successive rows (at a rate, for example, of 60 frames/second), which produces the actual colors seen at each location on the screen.

72. When the computer changes an image displayed on the screen, it updates the corresponding values in the frame buffer. Simply writing a new

number into the frame buffer at a given location results in a new color appearing at that position on the screen starting with the next refresh cycle.

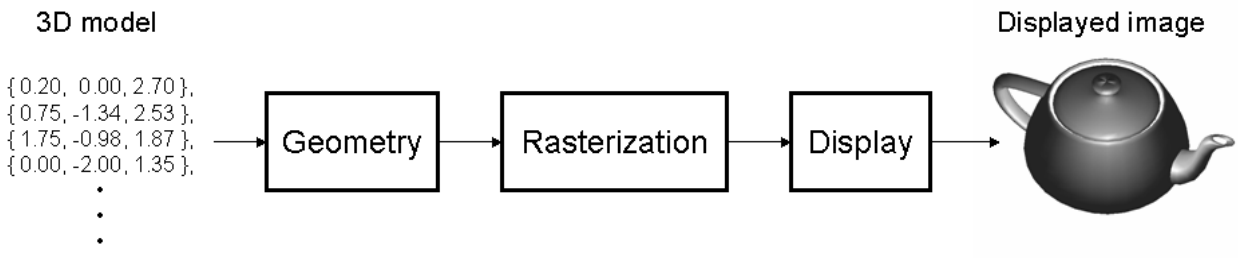
73. Creating an image of a 3D scene involves taking a mathematical description of the objects in a scene, looking at it from a given point of view, and figuring out what colors to draw at all the pixel locations in the frame buffer to create the corresponding image on the screen, as shown in the following example of a 3D house being drawn on a 2D display. All 3D points in the scene are mapped to the corresponding pixels on the screen by projecting along lines of sight, as though the scene was being photographed by a camera onto film. The "projection plane" below is another term for the "viewing frustum" as the term is used in the '645 Patent.



74. The mathematical description of the scene is known as a 3D model. Each 3D object in the 3D model is typically represented using a collection of

geometric “primitives” such as points, lines, and polygons (usually triangles) that make up the object surfaces. In the simple example above, the house might be modeled using 4 polygons for the walls and 4 more polygons for the roof. Each polygon is defined by its vertices or corners. Typically, each vertex is specified using 3D numerical coordinates, X, Y, and Z, for its location in 3D space and R, G, and B values for its color. A mathematical process called rendering is used to model a virtual "camera" looking at the 3D scene from a particular point of view, mathematically project all the 3D polygons into the corresponding 2D pixels in the display, and assign the appropriate colors to them in the frame buffer.

75. Various 3D computer graphics systems are built around the concept of a graphics pipeline. Acting like an assembly line, the graphics pipeline takes in the "raw materials" consisting of the data for the underlying 3D model and processes these through a series of computational steps to produce the image displayed on the 2D screen. In its simplest form, a graphics pipeline is described as having a series of three general phases, geometry, rasterization, and display, as shown in the diagram below:



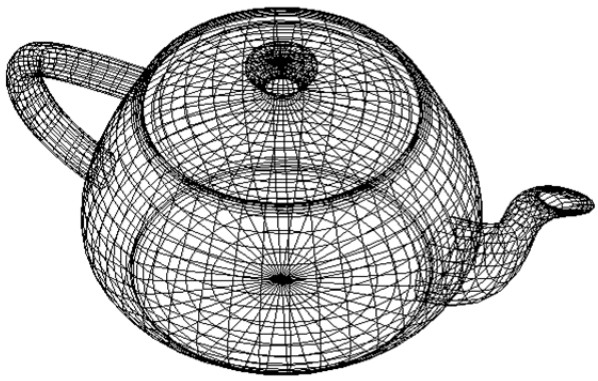
76. In the above diagram, the 3D model represents the X, Y, Z coordinates and R, G, B color values for all the 3D polygons that make up the objects in the desired scene, along with certain other information such as the location of the camera, light sources, display boundaries, etc. Geometry refers to the calculations performed to mathematically transform the 3D coordinates of all the polygons in the 3D model into corresponding screen coordinates given the location and orientation of the imaginary camera viewing the scene. Rasterization refers to the computations that determine all of the 2D pixel locations that will be visible within each 3D polygon and the colors those pixels should have. Display is then the process of writing the 2D pixel color values into the frame buffer and thereby causing the corresponding image to be displayed on the screen.

77. Rasterizing a polygon generally involves three main tasks: determining which pixels fall within the polygon (scan conversion), determining which of these pixels are visible on the screen (visible-surface determination), and determining what color to assign each visible pixel (shading).

78. The process of determining which pixels within the scan converted polygons will actually be visible on the screen is known as visible-surface determination. Depending on the direction from which the scene is viewed by the virtual camera, certain polygons (or portions thereof) may be occluded by other polygons and not visible on the screen, such as the back wall of the house in the

example shown above. One common way to solve the visible surface problem is to write the RGB value for a pixel into the frame buffer only if its 3D position is closer to the camera than what may have been previously written into that same location by another polygon, much as an oil painting is painted in layers from back to front. This involves what is known in the art as depth buffering or z-buffering, that is, keeping track of the depth or "Z" value (distance from the viewer's eye) currently residing at each pixel.

79. Once the scan conversion and rasterization processes are complete, the graphics program must assign colors to each visible pixel, a process that has evolved substantially over the history of computer graphics and depends on the level of realism desired in the resulting image. In the simple example below, only the scan converted pixels that make up the edges of each polygon are drawn with black pixels.



Example of a Wireframe Image

80. The resulting collection of polygons approximating the three-dimensional object is sometimes referred to as a "mesh" or "polygon mesh." The use of meshes, including foundational work performed by Microsoft to represent three-dimensional objects at varying levels of detail, is described in detail in Hughes Hoppe's 1996 paper "Progressive Meshes," (App. N), which was published in the SIGGRAPH '96: Proceedings of the 23rd annual conference on computer graphics and interactive techniques, pp. 99-108, and is also available online at http://research.microsoft.com/en-us/um/people/hoppe/pm.pdf.

2. Texture

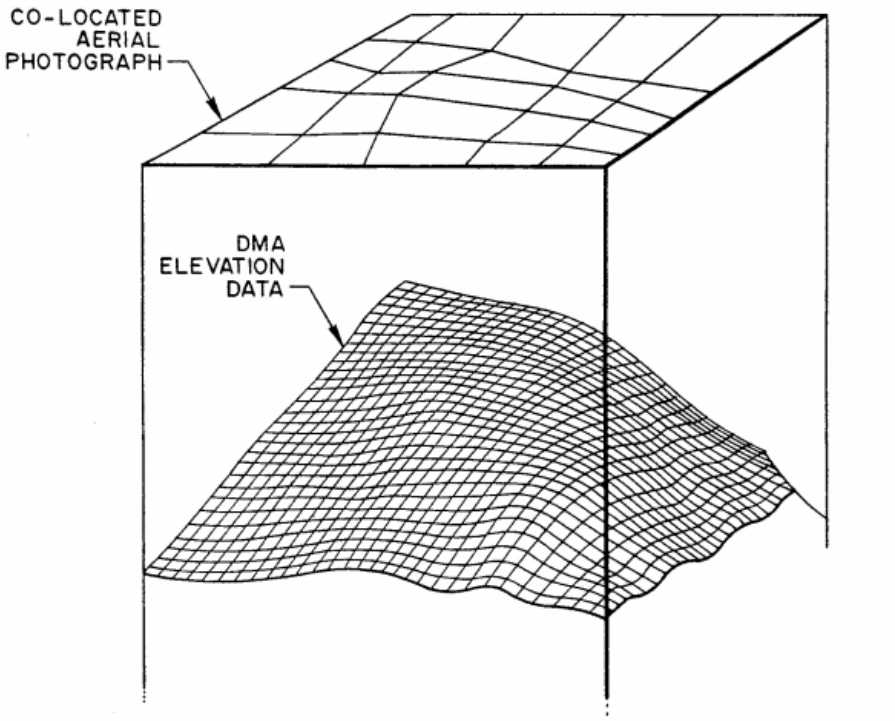
81. In 1974, "texture mapping" was developed as a further improvement in adding detail to objects or images. Texture mapping involves applying a 2D image or function approximating some real-world material like wood, bricks, fabric, marble, or a checkerboard, to the surface of polygons (*i.e,* the mesh) as in the image shown below. The "pixels" of a texture map are often referred to as texture elements or texels to distinguish them from the pixels of the resulting image. Texture mapping is like applying wallpaper or a decal to a surface. It is possible to construct a brick wall by carefully drawing many 3D bricks, which takes a lot of work, or one can simply paste a photograph of a brick wall onto an otherwise flat wall, which is easier and looks like a brick wall if you don't look too close. Texture mapping has become standard in 3D graphics systems to use

texture mapping to quickly fill in realistic detail for many of the objects in a 3D scene, especially floors, walls, sky, and other background areas. Textures can be either color images, or can be monochrome images used to modulate the untextured color of the polygon.

82. Some textures may be generic- for example, a 3D graphics rendering program might re-use a "wood" texture for all objects represented as wood. In this scenario, the texture is essentially "wallpaper" with a repeating pattern applied to certain objects or surfaces within the field of view.

83. Textures may also be unique, or specific to a particular surface or object. This is often the case when photographs are used as textures. For example, when satellite or aerial photographs are used in a 3D rendering of a landscape, the specific portions of the imagery that correspond to a particular location are mapped onto the terrain at that location. For example, U.S. Patent No. 5,179,638 to Dawson et al., assigned to Honeywell, Inc., (App. K) illustrates how an aerial photograph can be used as "texture data" and mapped onto co-located digital elevation data, as shown in Fig. 2:

Fig.-2



84. This technique of using aerial imagery as a texture applied to a three-dimensional model of terrain is also known as a "synthetic view." Synthetic view technology can be used in aviation to provide a pilot operating at night or bad weather with a synthesized view of the terrain around them based on actual position (*e.g*. derived from GPS). Appendix AA, Hansen, J. et al, "Real-time synthetic vision cockpit display for general aviation," AeroSense '99, International Society for Optics and Photonics, 1999, describes such a system. In the figure below, the bottom portion of the figure shows a wire-frame diagram illustrating the three-dimensional model of terrain, while the top image shows the synthetic view created by rendering satellite imagery on the terrain model:

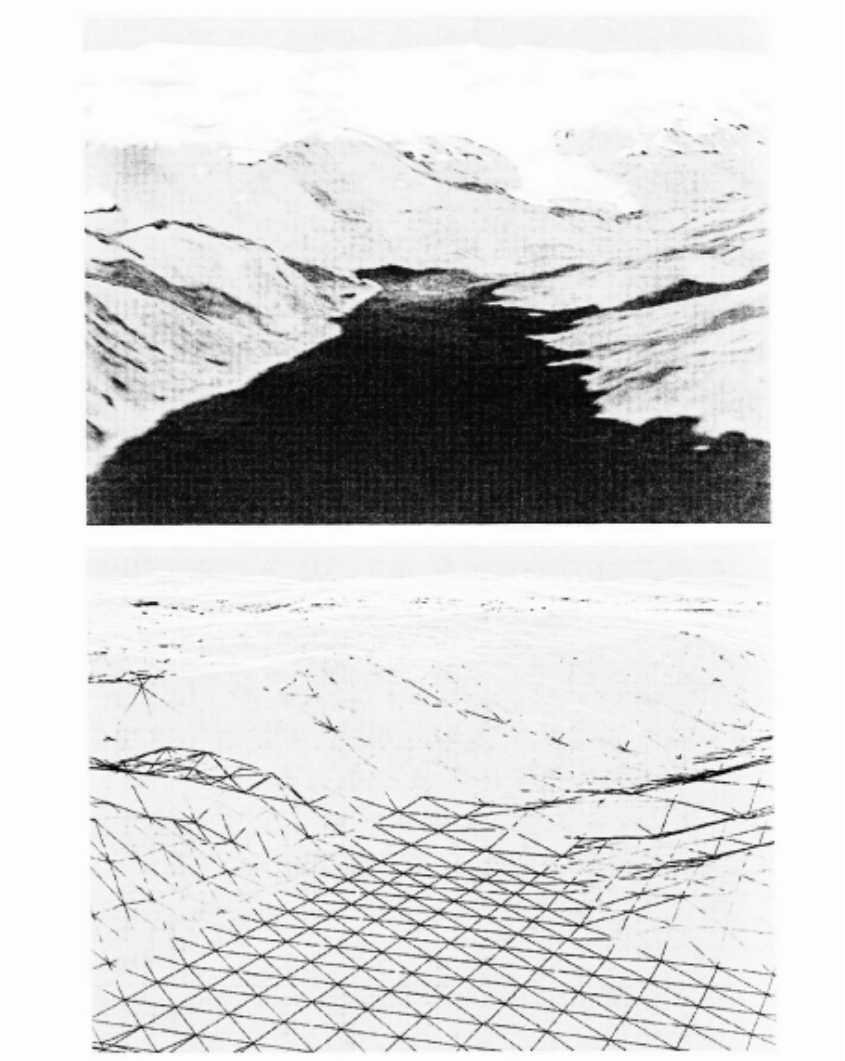
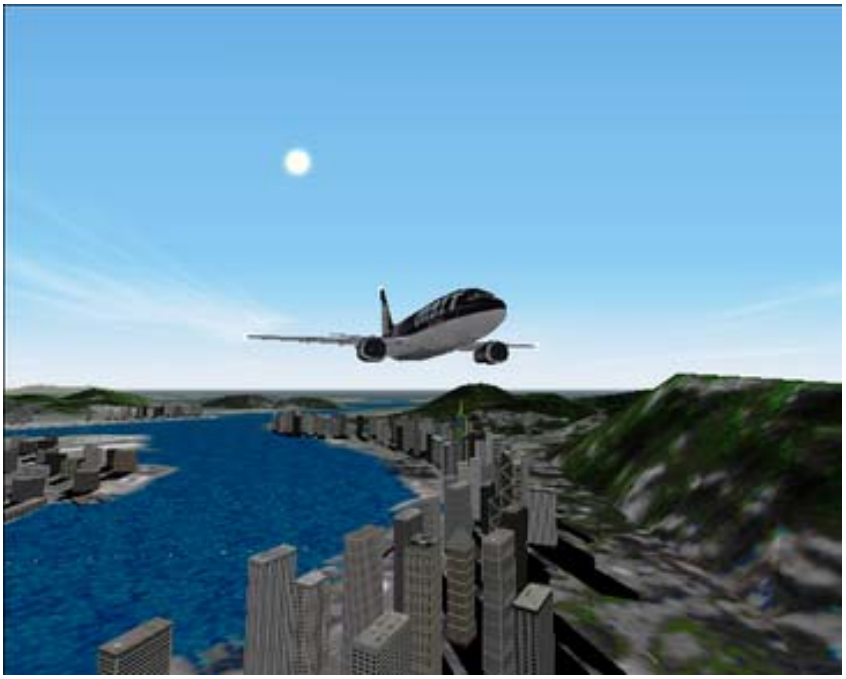


Figure 5. Terrain with extreme amounts of structure can be accommodated with high fidelity. The bottom graphic is a wire frame image of the Alaskan coastline. On the top is the fully rendered scene with imagery.

85. Microsoft used a similar technique with its popular Flight Simulator series of computer games, starting with Flight Simulator 1995. Flight Simulator utilized a real-time 3D rendering of terrain features with textures generated from a variety of sources, including satellite imagery. The figure below illustrates a 3D perspective view generated in Flight Simulator 2000, which was actually released in late 1999:



3. Virtual Reality Modeling Language (VRML)

86. VRML was a well-known industry standard that defined a file format for describing interactive 3D objects and worlds (App. KK). The 1997 release version of the VRML standard, commonly known as VRML97, specified a file format by which visual information, *e.g.*, the texture and 3D modeling discussed herein, can be stored on a computer and viewed.

87. A user wishing to view visual information stored in the VRML format could typically use a VRML viewer program, a VRML browser or a plug-in that worked with an off-the-shelf web browser. Because VRML was designed to operate in a client-server system, *e.g.*, conveying visual information via web pages, a typical network-based VRML application would rely on an underlying Internet

mechanism of hypertext data requests to fetch VRML data from a server to a user computer. One advantage of VRML file format was that it allowed for hyperlinking to other media such as text and images, and thus provided a mechanism by which a user could navigate through a visual scene being viewed, while in the background, the VRML application would fetch and/or render additional data.

G. Mip-Maps

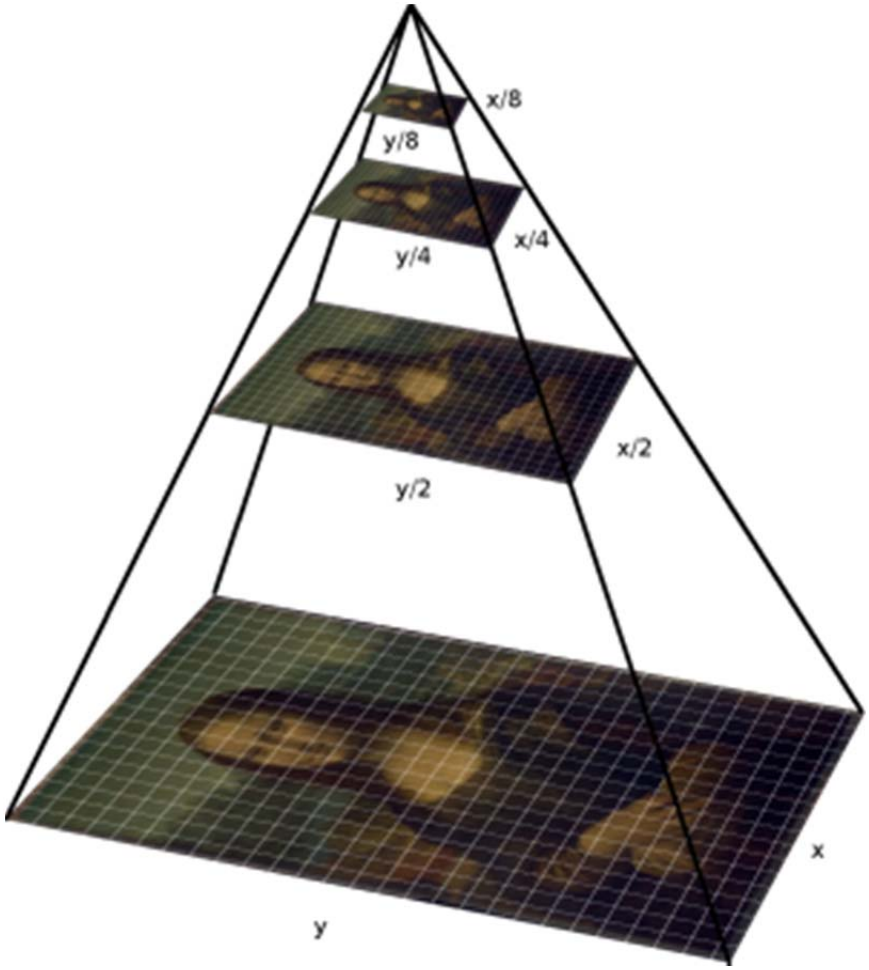
88. The provisional applications that the '645 Patent claims priority to described the use of "mip-maps" as "surface textures when rendering a two-dimensional representation of a three-dimensional scene." *See, e.g.*, Ex. 1066 at 7-9. This mip-mapping technology, however, has been used for rendering surface textures since 1979, more than two decades before the filing date of the provisional applications to which the '645 Patent claims priority. App. L at 2, Lance Williams, *Pyramidal Parametrics*, Computer Graphics, vol. 17, no. 3, July 1983.

89. The term "mip" derives from the Latin phrase "multum in parvo" meaning "many things in a small place." The term was adopted by Lance Williams in his 1983 paper, which indicated that "the mip-mapping technology has been used successfully to bandlimit texture mapping . . . since 1979." *Id.* Mip-mapping has been adopted in several versions of the OpenGL Standard prior to the filing date of the '645 Patent, including OpenGL 1.1 released in March 1997 and

OpenGL 1.2.1 released in April 1999. App. M, "Mipmapping" section of the OpenGL 1.1 Standard;[6] App. T at 129-131. OpenGL (GL stands for "graphics library") is a 3D graphics standard originally designed and released by Silicon Graphics, Inc. of Mountain View, CA. As defined in OpenGL 1.1, a mipmap is "an ordered set of arrays representing the same image; each array has a resolution lower than the previous one. If the texture has dimensions $2^n \times 2^m$, then there are $\max\{n, m\} + 1$ mipmap arrays. The first array is the original texture with dimensions $2^n \times 2^m$. Each subsequent array has dimensions $2^{(k-1)} \times 2^{(l-1)}$ where $2^k \times 2^l$ are the dimensions of the previous array." *Id.*

90. An illustration of the mipmap pyramid is shown below. *See* Photospector Blog, http://photospector.com/gigapixels/.

[6] *Available online at* https://www.opengl.org/documentation/specs/version1.1/glspec1.1/node84.html#SECTION00681100000000000000



91. By the late 1990s, mipmaps were commonly used in 3D graphics applications, among other purposes, to render object textures at varying levels of detail based on the proximity of the object to the simulated viewpoint. For example, it would ordinarily be preferable to display an object in close proximity to the viewpoint at a high level of detail where the display is capable of showing a high level of texture detail, whereas for more distant objects a lower level of resolution is preferable because the display screen is unlikely to be capable of displaying the highest-resolution texture at a great distance, and because lower-

resolution textures require far less system resources and bandwidth to retrieve, load and render. U.S. Patent No. 5,760,783 to Migdal et al ("Migdal," App. BB) is a patent from Silicon Graphics which describes how mipmaps may be used to render textures- including satellite or aerial photographs used as terrain textures for large maps, such as a flight simulator application. App. BB, 9:5-17, 10:14-19. Migdal illustrates how mipmaps at higher levels of detail may be used for points closer to the viewpoint and lower levels of detail for more distant objects. For example, Fig. 4C of Migdal illustrates a perspective view of regions of three different level of detail maps aligned with a center line from an eyesight location:

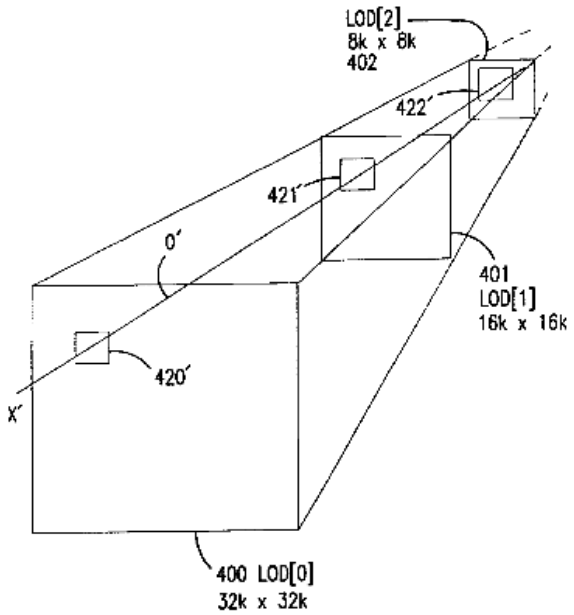


FIG.4C

92. Migdal teaches that Fig. 4C illustrates that the clip-map "contains sufficient texel data to cover larger minified areas in the background of a display where coarser texture detail is appropriate." *Id.* at 10:3-5. In my opinion, this teaching of Migdal is representative of what was already well-known in the art long before the earliest priority date claimed by the '645 Patent: that 3D graphics applications could use "mipmaps" or similar level-of-detail pyramids, to render objects closer to the viewpoint at a higher resolution and objects more distant from the viewpoint at a lower resolution.

93. Fuller (Ex. 1011) also illustrates this principle of using mipmaps to display closer objects at higher resolution and more distant objects at lower resolution. Fuller describes an online system for 2D and 3D visualization of map data, which creates a 3D perspective representation of a landscape using a digital elevation model (DEM), then uses mipmaps of aerial images as the textures. Ex. 1002, Fig. 4:

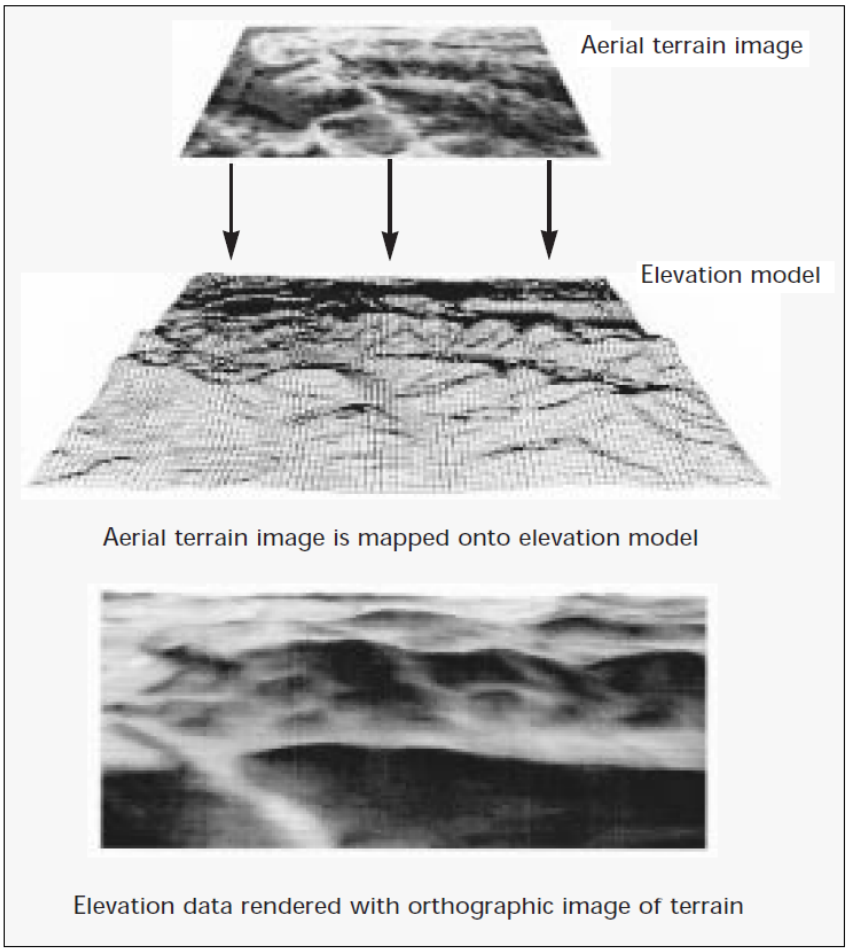


Figure 4. *Mapping an ortho-image onto its digital elevation model. (Source: SRI International)*

94. Fig. 3 of Fuller shows that higher resolution images are mapped onto portions of the terrain nearest the viewer, while lower resolution images are mapped onto more distant portions of the terrain:

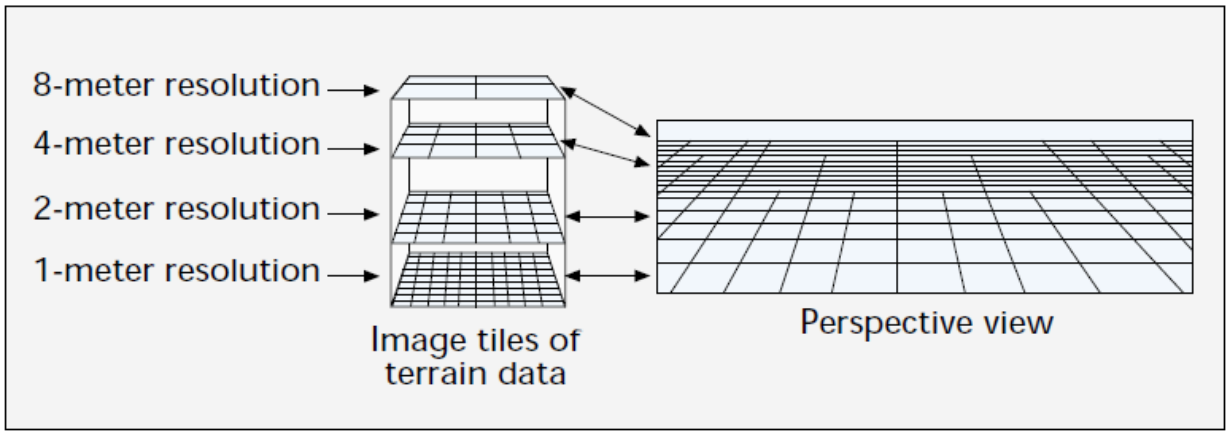


Figure 3. *Relationship between tile resolutions and perspective view. (Source: SRI International)*

H. Storage of image data

95. In practical systems, the visual information in the form of image tiles, image pyramids, mip-maps, 3D graphics, meshes, and so on, is stored on a computer from which users can access the visual information.

96. The '645 Patent describes that a network image server system stores the source image data. Ex. 1001 at 6:8-22. The source image data is typically a high-resolution bit-map raster map and/or satellite imagery of geographic regions. *Id*. The '645 Patent describes that this source image data is pre-processed to obtain a series of derivative images, and is also subdivided into a regular array such that each resulting image parcel has a same resolution. The '645 Patent then describes that each image parcel is "preferably" stored in a file such that any image parcel can be located by specification of a K_D, X and Y value.

97. Many formats for storing images, and the trade-offs for storage and retrieval efficiency, were well-known for several years before the earliest priority date claimed in the '645 Patent. The formats and trade-offs related to the color space used for image representation, the tile size used in case of multi-resolution images, whether different portions of an image are stored in one file or multiple files, the number and details of headers used to indicate information about the image, and so on.

98. For example, GeoTIFF is a file format (Appendix GG) for storing geographical image files using the well-known Tagged Image File Format (TIFF). The GeoTIFF spec defines a set of TIFF tags provided to describe all "Cartographic" information associated with TIFF imagery that originates from satellite imaging systems, scanned aerial photography, scanned maps, digital elevation models, or as a result of geographic analyses. Its aim is to allow means for tying a raster image to a known model space or map projection, and for describing those projections. The 1.0 release version of the GeoTIFF file format builds upon the then available revision 6.0 of TIFF.

99. TIFF (Appendix HH) defines a file format for storing image data. In particular, TIFF 6.0 provides for storage of multiple images in a single TIFF file, each image corresponding to a subfile. These images could be, e. g., a full resolution image and its reduced resolution representations.

100. FlashPix by Kodak was another well-known format for storing images (Appendix II). A FlashPix file stored image data in a hierarchy of resolutions from the highest available for an image, down to the lowest defined format (§2.1). Thus, depending upon an implementer's choice, a FlashPix file either contained a single resolution image or an entire multi-resolution hierarchy (§2.2, FIG. 2.3). For convenient access, each resolution image was organized in tiles, which represented rectangular pixel arrays of the image. §2.3. FIG. 2.4.

101. These techniques, and other obvious variations, for storing image data, including multiple resolution versions of an image, provided multiple implementation choices to an implementer. Depending on design criteria such as the amount of storage available, tolerance to header overhead, transmission latency when the images are to be transferred over a network, portability and compatibility with client viewing programs, an implementer would be able to select a storage method for storing images in a single file, with each tile in its separate file, or with all tiles of one resolution in a single file, and so on.

VII. OVERVIEW OF THE '645 PATENT

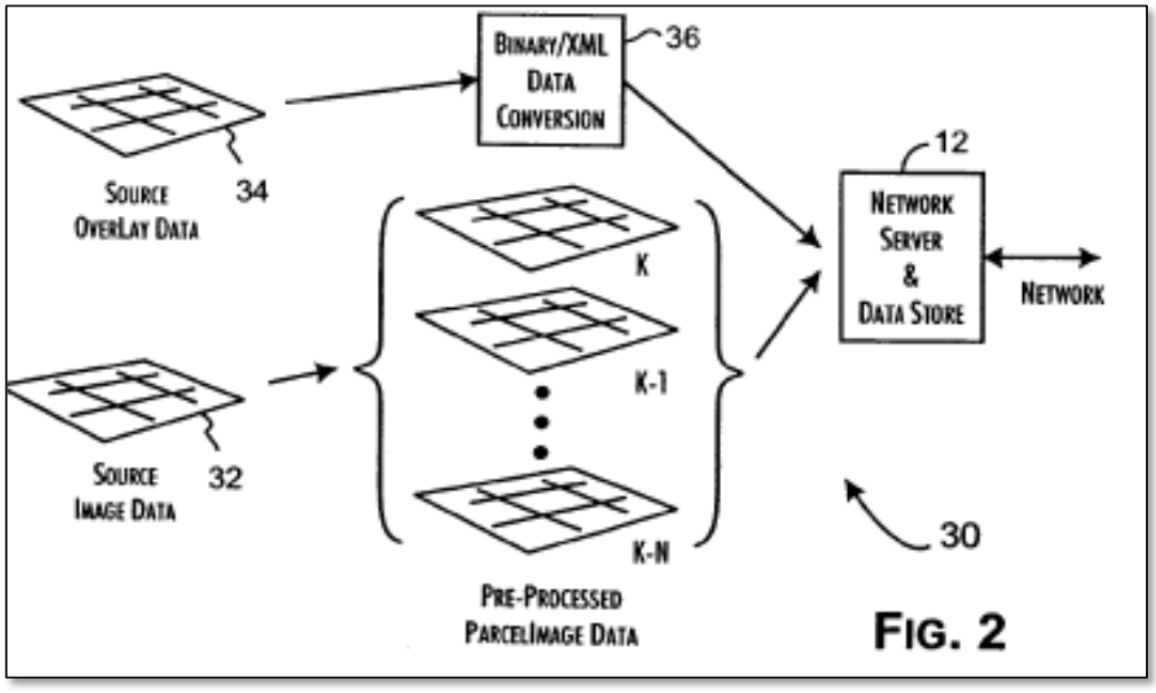
102. The '645 Patent describes a system in which "[l]arge-scale images are retrieved over network communications channels for display on a client device by selecting an update image parcel relative to an operator controlled image viewpoint to display via the client device." Ex. 1001 at Abstract.

103. The "Background" section of the '645 Patent describes a "well recognized problem" of how to reduce the latency for transmitting full resolution images over the Internet on an "as needed" basis, particularly for "complex images" such as "geographic, topographic, and other highly detailed maps." Ex. 1001 at 1:55-2:5. The '645 Patent states that solutions already in existence included "transmitting the image in highly compressed formats that support progressive resolution build-up of the image within the current client field of view." *Id.* at 2:5-9. The '645 Patent also states that such "conventional" solutions, like the ones described in U.S. Pat. Nos. 4,698,689 (Tzou) and 6,182,114 (Yap), usually "presume that client systems have an excess of computing performance, memory and storage" and are "generally unworkable for smaller, often dedicated or embedded" clients. *Id.* at 2:16-3:28. According to the '645 Patent, the conventional solutions do not work well under "limited network bandwidth" situations. *Id.* at 3:29-54.

104. To address these perceived issues in the existing art, the '645 Patent discloses a system purportedly capable of "optimally presenting image data on client systems with potentially limited processing performance, resources, and communications bandwidth." *Id.* at 3:63-67.

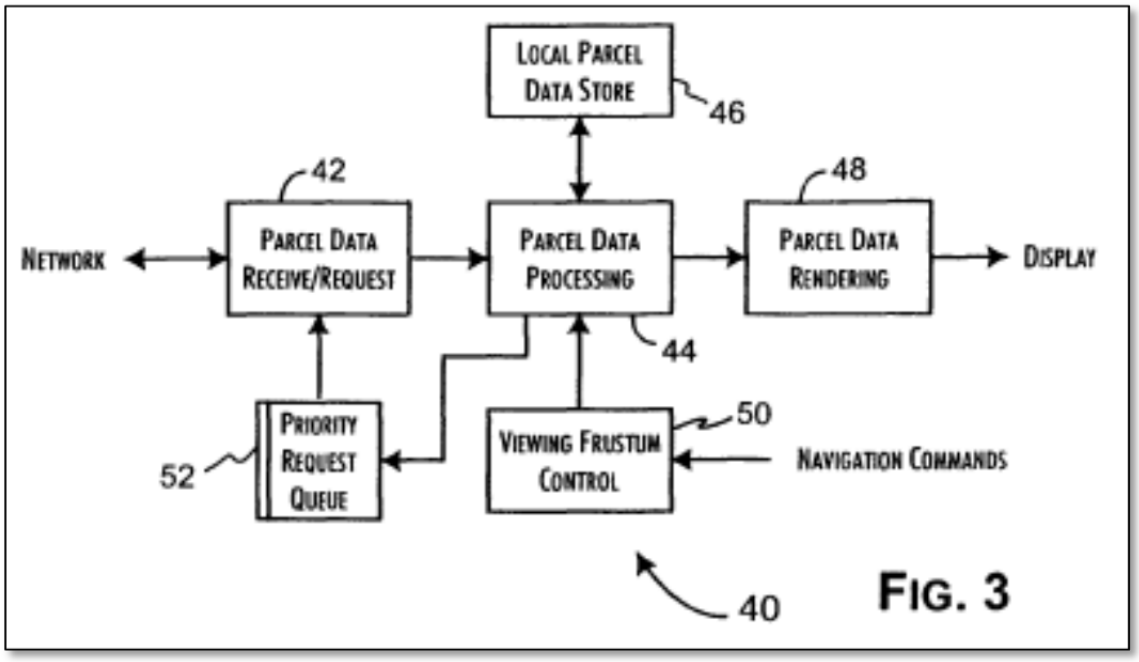
105. Specifically, the '645 Patent describes an image distribution system having a network image server and a client system, where a client can input

navigational commands to adjust a 3D viewing frustum for the image displayed on the client system. *Id.* at 5:44-6:7. High-resolution source image data is pre-processed by the image server into a series K_{1-N} of derivative images of progressively lower image resolution. *Id.* at 6:23-38, Fig. 2:



106. The source image is also subdivided into a regular array of 64 by 64 pixel resolution image parcels (a.k.a. image tiles), and each image parcel may be compressed to fit into a single TCP/IP packet for faster transmission. *Id.* at 6:23-44; 8:28-49.

107. The client system in the '645 Patent has a "parcel request" subsystem to request image parcels from the server, a "control block" that directs the transfer of received image parcels and overlay data to a local parcel data store. *Id.* at 7:26-48. The control block also decompresses the image parcels and directs a "rendering engine" to render them. *Id.* at 7:49-51; Fig. 3.



108. When the viewing point is changed in response to a user navigation command, the control block "determines the ordered priority of image parcels to be requested from the server . . . to support the progressive rendering of the displayed image." *Id.* at 8:8-11. A number of image parcel requests are then placed in a request queue, to be issued by the parcel request subsystem according to each request's assigned request priority. *Id.* at 8:11-16; 9:24-36. Although various factors may affect the priority assigned to a parcel request, *e.g.*, the "resolution of

the client display" (9:54-10:5) or whether the image parcel is "outside of the viewing frustum" (10:27-29), generally speaking, "image parcels with lower resolution levels will accumulate greater priority values," so "a complete image of at least low resolution will be available for rendering" in a fast manner (11:10-21). In addition, a control parameter used in calculating the priority can be set in a way that gives "higher priority for parcels covering areas near the focal point of the viewer" to make sure that image parcels are requested "based on the relative contribution of the image parcel data to the total display quality of the image." *Id.* at 11:22-40. The '645 Patent acknowledges that storing and retrieving image tiles in a manner that facilitated retrieving them based on their position and resolution level was known in the art, *e.g.* in Yap. Ex. 1001 at 2:32-58. The '645 Patent also acknowledges that zoom and pan functions for user image navigation were well known (*id.* at 1:64-2:2) as the Tzou Patent taught selective transmission of low resolution image data and subsequent updating of the prior transmission with succeeding refined images (*id.* at 2:16-21), the Yap Patent suggested updated transmission of image data parcels based on the user gaze point as the user-controlled image viewpoint (*id*. at 2:42-58). Woods further discloses using URLs to request VRML resources (e.g., textures) which are prioritized based on criteria such as the position and orientation of the viewpoint. Ex.1003 at 7:27-36, 7:66-8:8.

109. In the '645 Patent, after the needed parcels are requested and received, an algorithm is used to select the image parcel for rendering and display. *Id.* at 9:37-42. Overlay data may also be added to the display if its image coordinates match the current image parcel location. *Id.* at 9:46-51. The '645 Patent discloses that two-dimensional image parcels are displayed in a three-dimensional space using projection transform. *Id.* at 5:65-6:7; 7:65-8:7; 9:37-43; 11:22-34; 11:60-12:2. In my opinion, there is no disclosure in the specification of the '645 Patent that teaches or suggests that the images displayed are mapped onto an elevation model. The '645 Patent does not mention an elevation model or surface model or any other surface geometry data onto which the imagery would be texture mapped. The '645 Patent specification suggests that an overlay may include 3D objects (*Id.* at 6:17-22), but a POSITA would understand such "overlays" to be displayed on top of the imagery like other overlays such as "icons, building and landmark names." The specification of the '645 Patent effectively discloses a view that is "three-dimensional" in the sense that it generates a viewing perspective that contains position, rotation, and height components, but operating over a flat plane of terrain imagery. *Id.* at 5:65-6:7. Although the images rendered may represent an image or scene in 2D or 3D space, they are always rendered onto a 2D screen of the user computing device. *Id.* at 2:59-3:13, 5:65-6:22, 7:65-8:7, 9:37-53, 10:43-11:9, 11:60-12:15.

110. The '645 Patent states that its disclosed technology can achieve faster image transfer by (1) dividing the source image into parcels/tiles, (*id.* at 6:23-38), (2) processing the parcels/tiles into a series of progressively lower resolution parcels/tiles, (*id.*) and (3) requesting and transmitting the parcels/tiles needed for a particular viewpoint in a priority order, generally lower-resolution tiles first. *Id.* 3:63-4:62.

VIII. IDENTIFICATION OF THE PRIOR ART AND SUMMARY OF OPINIONS

111. As explained below, it is my opinion that the prior art references cited in this Declaration disclose all technical features in Claims 1-36 of the '645 Patent, thus rendering them unpatentable.

112. Based on my analysis of the prior art references, Claims 1-7, 9-11, 13-19, 21-23, 25-31, and 33-35 of the '645 Patent are rendered obvious by Reddy in view of Woods; Claims 8, 20, and 32 are rendered obvious by Reddy in view of Woods and Chiarabini; and Claims 12, 24, and 36 are rendered obvious by Reddy in view of Woods and Fuller.

A. Reddy

113. "TerraVision II: Visualizing Massive Terrain Databases in VRML" by Martin Reddy et al. ("Reddy") (Ex. 1004) was published in the March/April 1999 issue of IEEE Computer Graphics and Applications. From my 25+ year

experience as an IEEE member, I am very familiar with the IEEE, which is the world's largest association of technical professionals and publishes a number of well-respected peer-reviewed periodical journals. The IEEE Computer Graphics and Applications was a well-known publication. In my opinion, persons of ordinary skill in the art in the field of computer graphics would be familiar with the IEEE and its publications, consider IEEE publications to be established, reliable sources of information accessible to those of skill in the art, and would rely on the publication and copyright dates indicated on the face of an article in an IEEE publication as a reliable indication of the actual publication date of the article. Reddy bears such publication dates on each page and a 1999 copyright date on the first page of the article (page 30 of the journal). In my opinion, the publication dates indicated on the face of Reddy are a type of information that persons in the field of computer graphics would reasonably rely upon. Reddy includes a footnote "0272-1716/99/$10.00 © 1999 IEEE." Based on my experience, this footnote means that Reddy was published sometime in the year 1999. Further, it was common practice to release IEEE journals to the public in the beginning of the stated period of the journal. For example, based on my experience receiving numerous IEEE journals over the past 25 years, the March-April 1999 issue would have been sent to subscribers in late February or early March 1999. Therefore, Reddy was published more than one year prior to December 27, 2000.

B. Woods

114. U.S. Patent No. 5,956,039 to Woods et al ("Woods") (Ex. 1003) is a patent that was filed on Jul. 25, 1997 and issued on Sep. 21, 1999. I understand from Microsoft's counsel that Woods is prior art based both on its 1999 issue date which pre-dates Bradium's provisional applications and also based on its even earlier 1997 filing date.

C. Chiarabini

115. U.S. Patent No. 7,324,228 to Chiarabini et al. ("Chiarabini") (Ex. 1006) is a patent that issued from an application filed on August 24, 2001. Therefore, Chiarabini was filed before the earliest non-provisional application to which the '645 Patent claims priority, which was filed on December 24, 2001. I discuss Chiarabini below as prior art against Claims 8, 20, and 32, which recite that the number of parallel requests is determined based on network response latency and available system resources. I have been informed that the mere fact that the '645 Patent claims priority to earlier provisional applications filed in 2000 does not necessarily mean that the claims are automatically entitled to that priority date. I have further been informed that a claim is not entitled to a priority date from an earlier application unless the disclosure in that earlier application, and every other application in the chain of priority, supports every element of the claim.

116. In my opinion, none of the provisional applications to which the '645 Patent claims priority contain any support for determining the number of parallel requests based on network response latency and available system resources as recited by Claims 8, 20, and 32. None of the provisional applications mentions "latency" or "system resources." Nor is there any description of this subject matter using different terms.

117. For example, Provisional Application No. 60/258,465 (Ex. 1066) states at page 8 lines 22 through 25 that "[d]ownloading is asynchronous; the renderer maintains a priority queue of download requests, and separate threads are downloading images. Whenever a download is complete, another download is initiated immediately, based on the highest-priority request."

118. In my opinion, while this disclosure does mention "separate threads," it says nothing about how many threads or requests are active at one time, or about adjusting the number of threads or requests that are used based on network latency and available system resources. Having reviewed the entirety of the provisional applications, I found no other disclosures that support this claim limitation.

119. Therefore, it is my opinion that the December 2000 provisional applications to which the '645 Patent claims priority do not support claims 8, 20, and 32. Therefore, Chiarabini is prior art against these claims.

120. In addition, it is my opinion that if Claims 8, 20, and 32 are construed to require a *runtime* determination of the number of parallel requests,[7] then they should not receive a priority date earlier than Sept. 29, 2016 when the '645 Patent was filed. While it is true that the statement that "[t]he number of pool threads is determined as a balance between the available system resources and the network response latency" (Ex. 1001 at 8:64-67) was included in the nonprovisional application filed on Dec. 24, 2001, a person of ordinary skill in the art would understand this statement to describe a *design-time* decision made by the system designer(s). This is clear from the preceding statement that "a pool of four network request threads" is used "[i]n the preferred embodiments" (*id.* at 8:62-64), and also the subsequent suggestion that the number four was determined based on "[e]mpirical[]" data suggesting it works well "for many wireless devices" (*id.* at 8:67-9:3). If the thread pool was adjusted automatically at runtime, there would not be a preferred pool size or a need to resort on empirical data to determine an appropriate pool size. Furthermore, there is no discussion in the '645 Patent of actually adjusting the number of request threads at runtime or any indication as to what "available system resources" would warrant adjusting the number of request threads at runtime.

[7] For example, Claim 32 appears to recite the "determining ..." limitation as an additional step of the method recited in Claim 25.

D. Fuller

121. "The MAGIC Project: From Vision to Reality," by Barbara Fuller and Ira Richer ("Fuller") (Ex. 1011) was published in the May/June 1996 issue of IEEE Network. I explained my familiarity with IEEE and its publications, as well as the widespread knowledge in the art of IEEE publications, above in regard to Reddy, and in my opinion the same analysis would apply to Fuller.

122. In my opinion, persons of ordinary skill in the art in the field of computer graphics would be familiar with the IEEE and its publications, consider IEEE publications to be established, reliable sources of information accessible to those of skill in the art, and would rely on the publication and copyright dates indicated on the face of an article in an IEEE publication as a reliable indication of the actual publication date of the article. Fuller bears such publication dates ("IEEE Network * May/June 1996") on each page and a 1996 copyright date on the first page of the article (page 15 of the journal). In my opinion, the publication dates indicated on the face of Fuller are a type of information that persons in the field of computer graphics would reasonably rely upon. In my experience, the publication dates indicated in Fuller indicates that Fuller was published sometime in 1996. Further, it was common practice to release IEEE journals to the public in the beginning of the stated period of the journal, For example, based on my experience receiving numerous IEEE journals over the past 25 years, the May/June

1996 issue would have been sent to subscribers in late April or early May 1996. Therefore, Fuller was published more than one year prior to December 27, 2000.

IX. CLAIM CONSTRUCTION

123. In conducting my analyses of the asserted claims of the '645 Patent, I have applied the legal understandings I set out below regarding claim constructions consistent with the "broadest reasonable interpretation" (BRI) standard described above, and offer them only for this *Inter Partes* Review. The claim constructions do not necessary reflect the appropriate claim constructions to be used in litigation proceedings, such as litigation in a district court, where a different standard applies. For example, I have been informed that if Bradium argues for a narrow claim construction or otherwise relies on a narrow interpretation of a particular term in an IPR, Bradium may be precluded from relying on a broader interpretation or construction in litigation.

124. I understand that, under the BRI claim construction, claim terms are given their ordinary and customary meaning as would be understood by one of ordinary skill in the art in the context of the entire disclosure. An inventor may rebut that presumption by providing a definition of the term in the specification with reasonable clarity, deliberateness, and precision. In the absence of such a definition, limitations are not to be read from the specification into the claims.

A. "Wireless Portable Device" in All Claims Except Claims 6, 10, 18, 22, 30, and 34-36

125. In my opinion, this term does not require further construction and should be given its plain and ordinary meaning. "Wireless" is a common, well-understood term, as is "portable." Therefore, in my opinion, a "wireless portable device" is simply a device that is wireless and portable. The term "wireless portable device" does not specifically appear in the specification of the '645 Patent, and having reviewed the specification, I did not see any evidence that the inventors acted to specifically define a "wireless portable device" as anything other than its ordinary meeting.

126. I understand that in IPR2016-01897, Bradium proposed that the term "mobile device" in the related '239 Patent be construed as "a portable small client such as a mobile phone, smart phone, or personal digital assistant (PDA) that is constrained to limited bandwidth." I further understand that in its Decision to institute IPR of the '239 Patent, the Board rejected Bradium's proposed limiting construction of "mobile device" and determined that the term needed no further construction. IPR2016-01897, Paper 17 at 9-10 (April 5, 2017). In general, Bradium's prior arguments have attempted to conflate the size, mobility, processing power, type of network connection, and bandwidth features described in the specification of the Bradium patents, in my view erroneously because these

are different things. For example, while the specification of the '645 Patent gives various examples of a "small client" (*see, e.g.*, Ex. 1001 at 3:5-10), "small client" is not the claim language, nor is it appropriate in my view to limit the construction of the term based solely on representative examples.

B. "Thereby Enabling Efficient Use of Network Bandwidth in Conditions of Network Latency" in Claims 8, 20, and 32

127. In my opinion, this phrase of the dependent Claims 8, 20, and 32 should be construed to have no limiting effect. I have been informed by counsel that "[a] 'whereby' clause that merely states the result of the limitations in the claim adds nothing to the patentability or substance of the claim" and that merely "laudatory" descriptions of the results of a process step are not given patentable weight when such a description "simply expresses the intended result of a process step positively recited." In my opinion, the term "efficient," is vague in this context and could have a wide variety of meanings depending on the specific application. Therefore, in my opinion, a person of ordinary skill in the art would understand this phrase only as stating the general goal for more efficient use of bandwidth, rather than a clear defining statement of the scope of the claims. As such, the "thereby" phrase in these claims does nothing more than state a "laudatory" description of the preceding language, *e.g.* "number of parallel requests by the wireless portable device for image parcels of the series is

determined based at least in part on network response latency and available system resources" (Claim 8). Nevertheless, it is my opinion that this claim language is taught by the prior art references discussed in this declaration even if it is construed to have a limiting effect, as I discuss further below.

C. "Configure[d/s] ... as a server to provide access to [the] at least some image parcels [received by the wireless portable device]" in Claims 7, 19, and 31

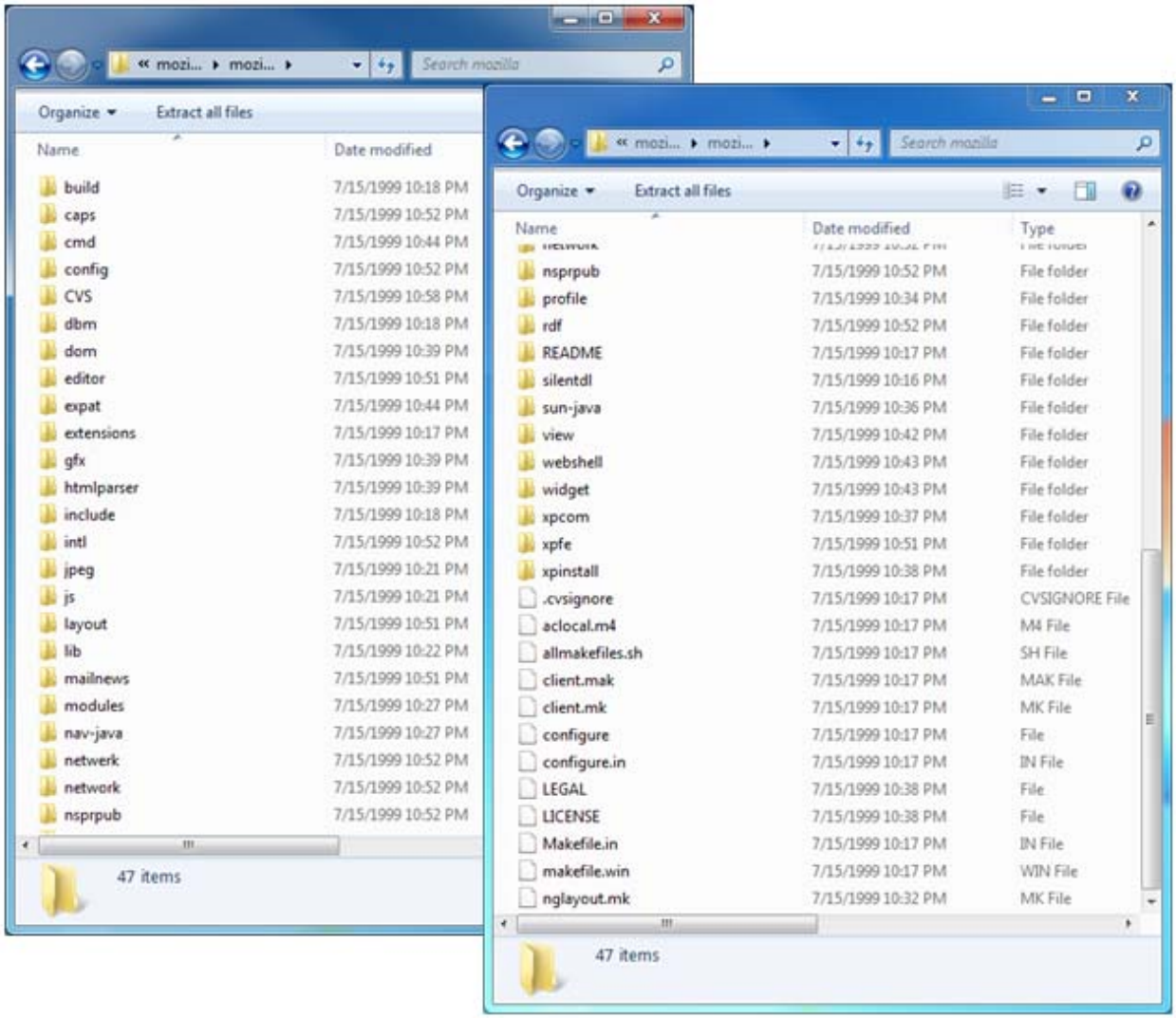
128. In my opinion, under the broadest reasonable interpretation, these phrases includes configuring the local store to provide parcels needed for display. Neither this claim language nor anything like it appears in the specification of the '645 Patent. The '645 Patent refers to a local parcel data store 46 (Ex. 1001 at 7:37-42, 9:10-23, 10:8-11, Fig. 3) and refers to a network server 12, 22 (*e.g.*, Fig. 2). However, the '645 Patent never describes the local parcel store as a server or describes configuring the local parcel store as a server. For example, nothing in the specification of the '645 Patent describes local parcel store acting as either an HTTP server or a database server, nor is there any indication that the local parcel store on a client is used to serve data parcels to other client devices in a network. I have reviewed the specification of the '645 Patent and I have found no other teaching in the '645 Patent that a person of ordinary skill in the art would understand to support this claim language.

129. Nonetheless, in view of the specification, a person of ordinary skill in the art would understand the "configured ... as a server to provide access to" as referring to the local parcel store's ability to satisfy needs for certain image parcels, namely those that have already been downloaded and stored in the local parcel store. It would be obvious to a person of ordinary skill that a cache like the local parcel store described in the '645 Patent would be implemented as a software module with APIs for storing items to the cache and for retrieving items from the cache.

130. The cache of the Mozilla open source browser illustrates my point that a simple cache acts as a server. I use Mozilla as an example because it was a well-known open source web browser in late 1990s. While Mozilla's source code was and is still available for download from Mozilla, I examined a version of the source code packaged with a 1999 book titled "Netscape Mozilla Source Code Guide." (Ex. 1060.) This source code on the CD-ROM is packaged with the name "mozilla-source-m8" which corresponds to the M8 release on July 16, 1999.[8] The source code folders and files on the CD-ROM have modified dates that are from

[8] *See, e.g.*, https://www-archive.mozilla.org/releases/history.html (Mozilla release history).

July 15, 1999 or earlier:



131. The Mozilla cache included APIs to request resources from the cache. For example, the "GetObj" API allowed the caller to provide a URL to retrieve the associated content from the cache. Ex. 1061 (nsCacheManager.h) at line 79. The Mozilla cache similarly provided APIs ("AddObject") for adding an object to the cache. (Ex. 1062 (nsMemModule.h) at line 57; Ex. 1063 (nsDiskModule.h) at line 41. In my opinion, the Mozilla browser cache is conventional in these respects and

is consistent with how persons of ordinary skill in the art would have understood and expected caches to operate.

132. Because conventional client-side caches could provide locally stored content for a specified URL, a person of ordinary skill in the art would understand a conventional cache to be configured as a server in the context of the '645 Patent.

D. "Image Parcel" in Claims 1-4, 7-8, 12-16, 19-20, 25-28, 31-32, and 36

133. In my opinion, this term does not require construction, or alternatively should be given its plain and ordinary meaning, that is, "a parcel of image data." I understand that in IPR2015-01432, the IPR of the '794 Patent, the Board construed this term as "an element of an image array, with the image parcel being specified by the X and Y position in the image array coordinates and an image set resolution index." The Board cited language appearing at 6:22-26 of the '794 Patent (and at 6:44-48 of the '645 Patent) stating that "[t]he image parcels are *preferably* stored in a file of defined configuration such that any image parcel can be located by specification of a K_D, X, Y value representing the image set resolution index D and the corresponding image array coordinate" (emphasis added). In my opinion, this language in the specification describes a particular embodiment, rather than defining the term "image parcel." In fact, when Bradium wanted to claim that particular identification scheme, it did so explicitly in the claims. *See, e.g.*,

U.S. Patent No. 7,908,343 at claim 1 ("storing each data parcel on the remote computer in a file of defined configuration such that a data parcel can be located by specification of a K_D, X, Y value"); U.S. Patent No. 8,924,506 at claim 7. Therefore, in my opinion, it would be incorrect to read this more detailed claim language into the simple term "image parcel."

134. It is also my opinion that a person of ordinary skill in the art would not interpret the plain and ordinary meaning of the term "image parcel" to require a particular method of identifying such image parcels. There are a number of different identification schemes that could be used to uniquely identify a particular image parcel within an image pyramid. For example, U.S. Patent No. 4,972,319, issued in 1990, depicts and describes two different identification schemes for image parcels in the same kinds of image pyramids:

FIG.8

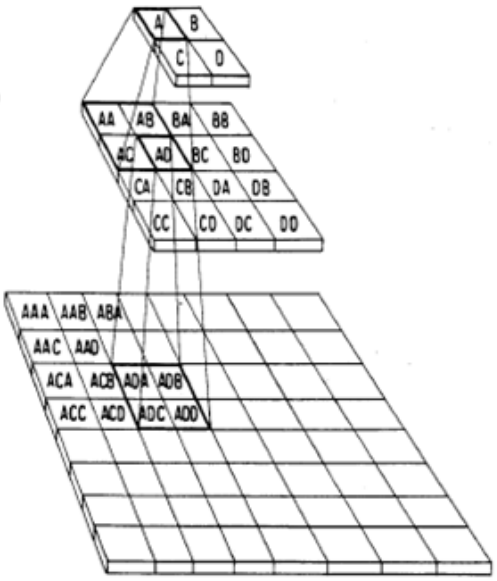
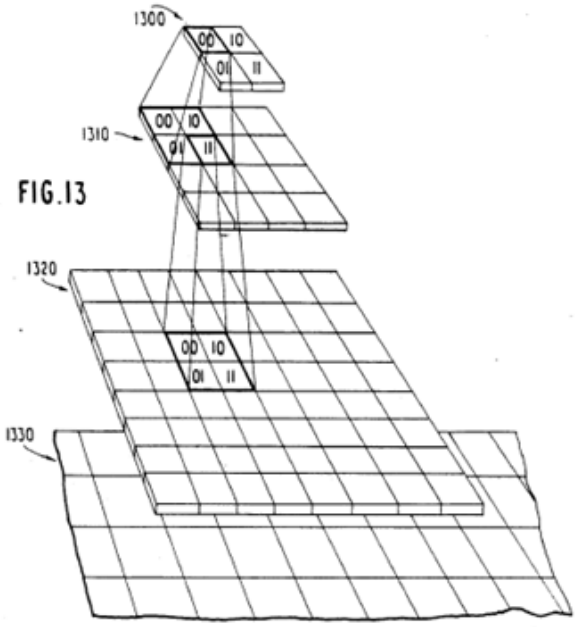


FIG.13



Ex. 1065 (Delorme '319 Patent) at 11:61-14:24 (describing scheme shown in Fig. 8) and 14:25-22:34 (describing scheme shown in Fig. 13). Therefore, in my opinion, a person of ordinary skill in the art would not understand the term "image parcel" to require any particular preferred identification scheme.

135. It is also my opinion that Bradium has taken positions in the related litigation which are inconsistent with a narrow construction of "image parcel." For example, Bradium's litigation infringement contentions with respect to the '343 Patent (Ex. 1072) appeared to suggest that the "tiles" in accused Microsoft products are the claimed "image parcels." Yet the Bing maps tile system uses a "quad key" system which interpolates X and Y values to create a single string identifying the tile, and does not include a separate variable for level of detail. Ex. 1073 at 4-6.

136. However, as I will discuss further in this declaration, it is my opinion that the challenged claims of the '645 Patent are obvious over the prior art references cited in this declaration even under the narrower construction of "image parcel" previously applied by the Board.

E. All Remaining Claim Terms

137. In my opinion, all other claim terms of the '645 Patent should be given their plain and ordinary meaning.

X. UNPATENTABILITY OF CLAIMS 1-36 OF THE '645 PATENT

A. GROUND 1: CLAIMS 1-7, 9-11, 13-19, 21-23, 25-31, AND 33-35 ARE UNPATENTABLE UNDER 35 U.S.C. § 103(a) AS BEING OBVIOUS OVER REDDY IN VIEW OF WOODS

138. In my opinion, each of Claims 1-7, 9-11, 13-19, 21-23, 25-31, and 33-35 are disclosed and rendered obvious by Reddy (Ex. 1004) in view of Woods (Ex. 1003). These references collectively teach all features of Claims 1-7, 9-11, 13-19, 21-23, 25-31, and 33-35, and a person of ordinary skill in the art would be motivated to combine the teachings of the references for the reasons discussed below.

139. Reddy and Woods provide related, contemporary teachings about the state of the arts in 2D and 3D visualization of large image data sets such as maps that give examples of reasons why the claims of the '645 Patent cover technology that was already well-known to those of skill in the art. Reddy describes in detail a versatile software system for retrieving geographic information over the Internet or WWW and displaying it to a user in a dynamic, three-dimensional manner using well-known level-of-detail management techniques to optimize the use of limited bandwidth to retrieve geographic data.

140. Reddy teaches a system for disseminating and viewing massive terrain databases and 3D maps over the Web using a VRML browser or a customized browser or a plug-in to a web browser such as Netscape Navigator or Internet

Explorer. The 3D map images are stored in a hierarchical data structure made up of a multiresolution hierarchy of images in which an original image and its downsampled versions at multiple resolutions are stored. Reddy teaches priority-related features such as a "coarse-to-fine" algorithm to enhance resolution and "pre-fetching" tiles based on an expected flightpath. However, Reddy does not explicitly use the word "priority." Woods is in a closely related field to Reddy because it also describes using VRML to access 3-D scenes over the Internet. Woods teaches how browsing in such scenes may be enhanced by prioritizing elements of the scene based on the proximity to the user viewpoint, as well as using a queue of browser fetch requests to retrieve elements of the scene. I will discuss each of these references in more detail below.

1. Overview of Asserted References

a. Reddy

141. Reddy is an IEEE publication that was published in the March/April 1999 issue of IEEE Computer Graphics and Applications journal. The authors worked at SRI International, where they developed a system called TerraVision II for visualization of massive terrain databases using the VRML language. Ex. 1004 at Title. TerraVision II was designed to allow users to access large terrain databases via a network connection such as the World Wide Web or the Internet. *Id*. at ¶ 1.

142. The TerraVision II system built on earlier work over several years by SRI on an earlier version of the TerraVision system, which was designed to operate in connection with a project called MAGIC ("Multidimensional Applications and Gigabit Internetwork Consortium") for the visualization of large amounts of three-dimensional data over a high-speed ATM network. I discussed a 1996 paper regarding earlier work on the MAGIC project, Fuller *et al.* (Ex. 1011) previously in this declaration. TerraVision, MAGIC, and TerraVision II were all funded partially by Defense Advanced Research Projects Agency (DARPA). *See* Ex. 1004, p. 37 (Acknowledgements).

143. As Reddy and his co-authors describe, TerraVision II extends the work performed in connection with TerraVision and MAGIC to a software program that could access data over a conventional WWW connection (not just a high-speed connection) and could operate on a variety of devices such as PCs and laptops, by working in connection with standard Internet browsers.

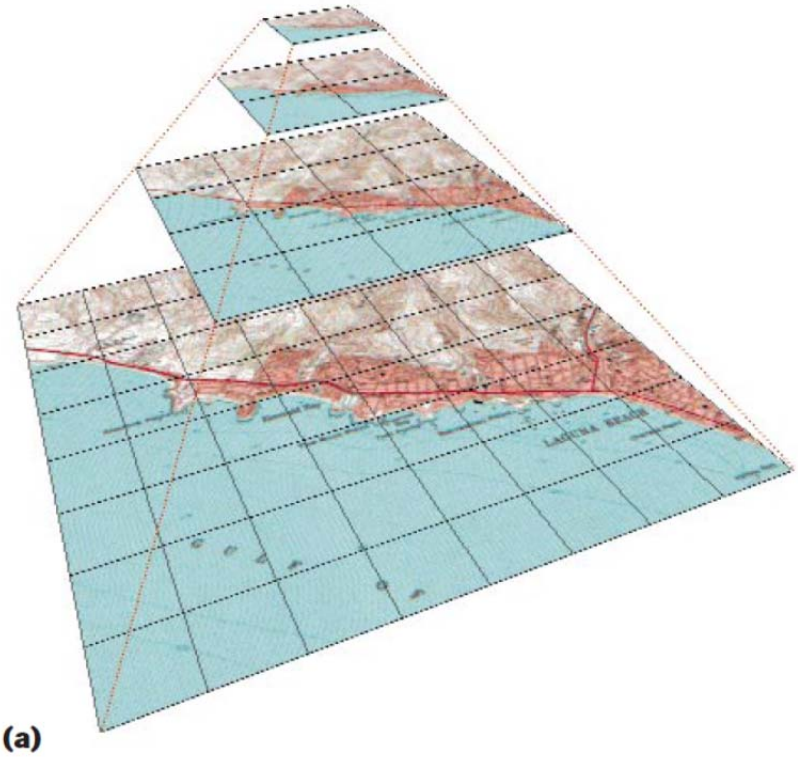
144. Reddy discloses that there was an increasing interest and need for researchers, including geographers, cartographers, geologists, and computer scientists, to access 3D maps and spatial data over the World Wide Web or even 28.8K modem connections. *Id*. at inset on p. 30. Reddy notes that, however, traditional single-resolution VRML images did not scale well for such use. *Id*. at ¶

1. For example, Reddy notes that some terrain models could run into hundreds of gigabytes in size. *Id*. at ¶ 2.

145. To allow user access to map details, *e.g.*, being able to view a particular building in a particular city starting from a satellite image of the earth Reddy discloses a technique in which data is progressively downloaded to the user as the user performs rotations and zooms. *Id*. at ¶ 3. In the TerraVision II system, the progressive downloading functionality was implemented on user devices in multiple ways including a VRML browser, a VRML plug-in to a web browser such as Netscape Communicator or Microsoft Internet Explorer or a custom TerraVision II browser. *Id*. at ¶ 31.

146. A key feature of Reddy involved generating a pyramid of multi-resolution images of an original image. *Id*. at ¶ 15. For example, if the original image is 1024 x 1024 pixels, then the pyramid might contain the original image along with down-sampled versions different levels of detail (LOD) at resolutions of 512 x 512 pixels, 256 x 256 pixels, 128 x 128 pixels, and so on. *Id*. at ¶ 15.

147. As Figure 1a shows, each pyramid image is then segmented into rectangular tiles, where all tiles have the same pixel dimensions.



148. Because each tile has the same pixel dimensions, the resolution and the number of tiles is reduced by a factor of two at each down-sampled level. The following chart summarizes the example disclosed by Reddy in Fig. 1 and the accompanying text at ¶¶15-17 (referring to the "source" layer as level 0):

Layer	**Total dimensions of layer (pixels)**	**Tiles in layer**	**Tile dimensions (pixels per tile)**	**Each pixel at this layer corresponds to ____ pixels in the source layer**
0 (source)	1024x1024	8x8	128x128	1
1	512x512	4x4	128x128	4 (2x2)
2	256x256	2x2	128x128	16 (4x4)
3	128x128	1	128x128	64 (8x8)

149. Using Figure 1b, Reddy further gives an example of how the multi-resolution pyramid can be used to download tiles at the appropriate resolutions to render a perspective view. *Id*. at ¶¶ 15-17. Reddy describes this as follows: "Figure 1b shows the lower-right corner in high resolution with the surrounding regions displayed in progressively lower resolution. Assuming a tile size of 128 x 128 pixels, this example requires downloading and rendering only 491 Kbytes (10 tiles) instead of the entire 3.1-Mbyte high-resolution image.[9] If the user's location

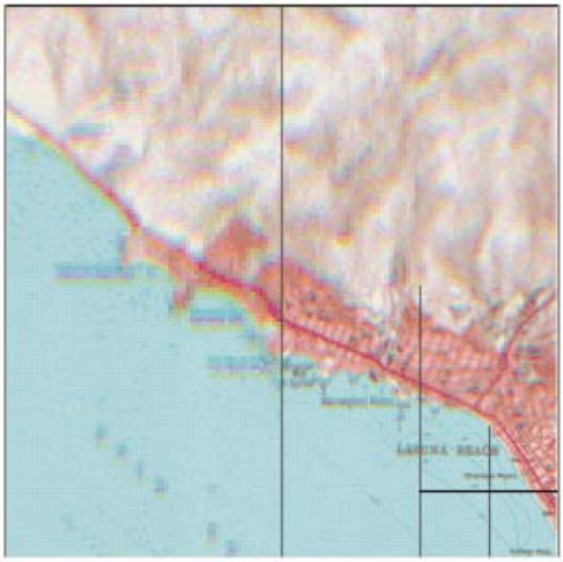
[9] The definitions of "kilobyte" and "megabyte" can be confusing because of an informal custom that has become prevalent in the computer industry. According to the standard definition, the terms "kilobyte" and "megabyte" refer to 1,000 and 1,000,000 bytes, respectively. However, because 2^{10} bytes is 1024 bytes, which is very nearly 1000, and 2^{20} bytes is 1,048,576 byes, which is very nearly 1,000,000, it has become commonplace in the computer industry to refer to 1024 bytes and 1,048,576 bytes as 1 kilobyte and 1 megabyte, respectively, and in fact this

is the bottom-right corner, then distant imagery is rendered at lower resolution than near imagery and we have achieved distance-based LOD." *Id*. at ¶ 16.[10]

---

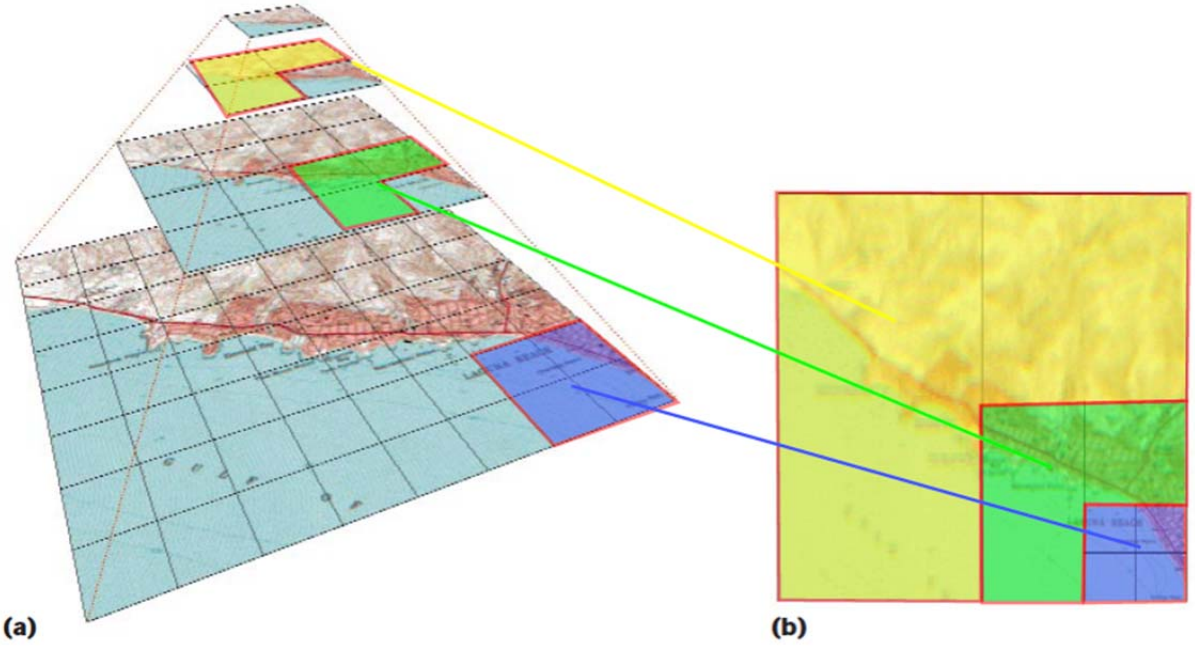
alternative interpretation has become incorporated in some standards such as the JEDEC family of memory standards. The reasons for the distinction are described in a National Institute of Standards and Technology webpage available at http://physics.nist.gov/cuu/Units/binary.html. I note the difference for purposes of this analysis because the figures given in Reddy for tile sizes suggest that Reddy uses the standard decimal (1000 bytes/kilobyte) notation while the '645 Patent use the binary (1024 bytes/kilobyte) notation.

[10] The numbers in this example work as follows. The original image has a size of 1,024 x 1,024 pixels, at 24 bits per pixel (*e.g.*, 8 bits for each color GB). This adds up to 1024 x 1024 pixels x 24 bits/pixel = 25165824 bits = 3145728 bytes = 3.145728 MB (assuming 1Mbyte = 1 million bytes). Similarly, 10 tiles at `18 x 128 resolution each add up to 128 x 128 x 24 bits/pixel x 10 = 3932160 bits = 491420 bytes = 491.42 KB (assuming 1 KB = 1,000 bytes).



(b)

150. The graphic below, based on Fig. 1 of Reddy, illustrates how ten of the tiles from the different levels in Fig. 1(a) correspond to Fig. 1(b), which shows the tiles that would be used to render a perspective view:



(a) (b)

151. Another important aspect of Reddy is that the image data is downloaded from the image database based on a user's viewpoint. *Id*. at ¶ 3. For example, if a user's location is the bottom-right corner, then distant imagery is rendered at lower resolution than near imagery and we have achieved distance-based LOD. *Id*. at ¶ 16. Thus, Reddy's technique only needs to fetch and display data for the region that the user is viewing and only at a sufficient resolution for the user's viewpoint. *Id*. at ¶ 17. As the user zooms into an image, the program downloads imagery at higher and higher resolutions. *Id*. at ¶ 3.

152. During prior proceedings involving the related '343 and '506 Patents, Patent Owner Bradium and its expert mischaracterized Reddy in numerous ways, which I will discuss briefly in this Declaration. However, I reserve the right to offer further rebuttal to any arguments made or arguments submitted by Bradium in this proceeding.

153. For example, Bradium previously characterized Reddy as "directed to a specialized client workstation image viewing software operating on conventional, fixed site computer system over a high bandwidth Internet connection." IPR2016-00448, Paper 20 at 19-20; Ex. 2003, ¶ 64. In my opinion, this characterization overlooks a substantial purpose of the teachings of the Reddy reference, which should be readily apparent to a person of ordinary skill in the art.

154. The work described on Reddy built on *previous* work funded by the DARPA Multidimensional Applications Gigabit Internet Consortium (MAGIC) project, which had previously resulted in the development of the original TerraVision system, which was partially described in 1996 in B. Fuller and I. Richer, *The MAGIC Project: From Vision to Reality*, IEEE Network May/June 1996 pp.15-25, Ex. 1011. The original TerraVision system was designed to operate over a high speed ATM network. However, the later work described in Reddy, including its description of the TerraVision II software and VRML browsing techniques, focused on extending the concepts developed in the earlier research performed by SRI to a wider variety of networks and devices, not just the high speed networks used in the original iteration of TerraVision.

155. To that end, Reddy states several times that a purpose of the teachings discussed therein is to develop a system that can be accessed over the Internet using a standard, generic browser on any device, not just a specialized system operating on a high and workstation on a specialized high speed network. For example, Reddy refers several times to the desire to disseminate maps and spatial data over the "World Wide Web" (Ex. 1004, ¶¶ 1, 9, 48), using standard browser software and java scripts (*Id.*, ¶¶ 3, 7, 9, 11, 31, 32, 39, 42, 47-49). In my opinion, it is readily apparent from reading Reddy as a whole that a significant purpose of Reddy's teaching is to enable a wide variety of users to access geographic data

over the Internet using standard browsing techniques and standard browser software that can be implemented on a wide variety of devices including small clients.

156. In my opinion, the characterization of Reddy previously provided by Bradium cherry-picks particular citations that relate to specific embodiments operating on a high-bandwidth network, and ignores both the additional teachings of Reddy that relate to applying the same principles either on more limited devices or more limited networks, as well as the broader teachings that a person of ordinary skill in the art would glean or infer from Reddy.

157. For example, Reddy teaches that its system can be implemented on a PC connected to the Internet and that a standard VRML browser can be used to browse the same data, which makes the system particularly well-suited to "military mission planning and battle damage assessments, emergency relief efforts, and other distributed time-critical environments." Ex. 1004, ¶ 48.

158. Bradium and Dr. Agouris both submitted arguments at some point during IPR2016-00448 which argued that the teachings in Reddy regarding military and emergency scenarios were intended for use with offline data saved on the laptop or with the basic functionality of VRML only (not TerraVision). The Board previously correctly rejected this argument as an improper reading of Reddy (IPR2016-00448, Paper 9 at 23-24). Likewise, Dr. Agouris admitted during her

deposition that Reddy taught that a laptop computer could be used for online browsing. Ex. 1018 at 157:4-158:12.

159. Bradium and its expert previously repeatedly attempted to distinguish between different embodiments taught in Reddy, such as the TerraVision II software and a standard VRML browser. I understand that a person of ordinary skill in the art would read Reddy as a whole for all that it teaches or suggests, and not simply for specific embodiments as Bradium reads it. Nevertheless, such distinctions are incorrect.

160. For example, Bradium previously cited Ex. 1014 (SRI Digital Earth Paper) which is a web posting by SRI at approximately the same time that Reddy was published describing the various facets of its terrain visualization system. While Bradium cited this exhibit in order to attempt to argue that the capability of a standard VRML browser would have been much more limited than TerraVision, this exhibit actually contains numerous teachings which showed that a person of ordinary skill in the art would have recognized the ability to apply the relevant teachings of Reddy to various devices. For example, page 1 of Ex. 1014 explains that SRI's digital earth proposal is to extend TerraVision functionality to "commercial, off-the-shelf" software (*id*. at 1), enable "open solutions" for a "wide cross-section of users" and integrate VRML support "directly with Internet browser software" (*id*. at 2-3). And while Bradium cited pp. 4-5 of Ex. 1014 as

"contrasting TerraVision running on fast graphics workstation with accessing the data only via a standard browser" (IPR2016-00448, Paper 20 at 22), the same section also clearly states that it is "feasible" that some of the features provided by TerraVision "could be implemented for a standard VRML browser through the use of various Java scripts embedded in the scene, or running externally to the browser." Ex. 1014 at 4. This evidence supports my opinion that a person of ordinary skill in the art would be motivated to consider all of the pertinent teachings of Reddy to be applicable to generic browsers that could execute on a standard computer, including one connected to a limited bandwidth communications channel, or a mobile device.

161. Bradium also previously confused the types of data that the embodiments described in Reddy can display. Reddy teaches a flexible system which allows browser software to locate and display several types of geographically linked data, including imagery such as satellite or aerial photography, as well as elevation data (based on digital elevation models such as the USGS digital elevation model), and features such as annotations or objects that exist on the terrain. Ex. 1004, ¶¶ 3, 12-18, 22-26. The imagery is divided into pyramids, just like the preferred embodiments described in the '645 Patent. *See, e.g.* Ex. 1004, Fig. 1; Ex. 1001, Fig. 2. It is my opinion that a person of ordinary skill in the art would recognize that satellite and aerial images are, by nature,

specific to particular coordinates since they depict a specific portion of the earth. While Reddy also teaches the ability to display digital elevation data as 3D polygons and drape imagery as textures over those 3D polygons, Reddy's teachings are not limited to such preferred embodiments, and a person of ordinary skill in the art would recognize that Reddy's teachings relating to displaying large sets of two-dimensional imagery utilizing a perspective viewpoint (see, *e.g.* Fig. 1) would apply whether or not elevation data was also used. This distinction is important because Bradium's previous arguments (*see, e.g*. IPR2016-00448, Paper 20 at 37-38) conflated the three-dimensional polygonal models used to model elevation in Reddy with the two-dimensional texture imagery. The '645 Patent itself does not describe any form of elevation modeling, but the claims do not exclude it either. It is my understanding that 3DVU developed an elevation modeling scheme several years after the applications to which the '645 Patent claims priority were filed. In effect, the '645 Patent effectively describes a later-developed, less sophisticated example of the teachings of Reddy, just without elevation.

162. I also have been informed by counsel that Bradium's counsel argued in IPR2016-00448 that the teaching of an "image pyramid" in Fig. 1 of Reddy should be disregarded because it is just a "concept," and further implied that the "image pyramid" was not actually used in implementation of Reddy. Appendix

NN at 46:22-48:5. This argument is wrong. Reddy teaches in numerous places that the systems described even in preferred embodiments include two-dimensional imagery, such as satellite and aerial photographs of particular geographic regions. *See, e.g.* Ex. 1004, ¶¶ 2-3, 6, 15-18, 23-24. Indeed, ¶ 23 specifically states that an "image pyramid" like those shown in Fig. 1 is incorporated into the model. Fig. 3 and the accompanying text at ¶¶ 19-21 describe how such image pyramids can be linked to other forms of data contained within the embodiments described in Reddy using a "geotile" structure. Therefore, even if the Board chooses to only read Reddy for the specific embodiments it describes and not for all that it teaches, the image pyramid of Fig. 1 of Reddy is very much a part of those embodiments. Moreover, it is my understanding that even if the image pyramid in Reddy were only taught as a "concept," and it is much more than that, such teachings would still be considered by a person of ordinary skill in the art to determine the obviousness of later patents claiming the same technology because a prior art reference must be considered for all that it teaches or suggests.

b. Woods

163. Woods teaches methods for "increasing the performance associated with creating simulated 3D worlds from a network" by using a priority scheme which sorts the priority of graphic objects (or "assets") in the 3D scene based on, *inter alia*, their proximity to a viewpoint. Ex. 1003 at Abstract. The priority

scheme is "used to determine the fetching, pre-fetching, and caching of URLs." *Id.* In a preferred embodiment, Woods teaches prioritization methods to retrieve graphic objects over the Internet in VRML. *Id.* at 4:62-5:4. Woods teaches that constraints in viewing 3D images online include "the bandwidth of the Internet connection and the limited resources and processing power of the local computer system. *Id.* at 2:61-34. For example, Woods teaches examples of Internet connections used to retrieve VRML data ranging from 28 KBps (kilobits per second) modems to 144 KBps ISDN lines, but that "bottleneck problems due to limited bandwidth" can arise even for faster connections. *Id*. at 3:17-27.

164. Woods proposes to address such problems and "increase[] performance by the efficient use of limited resources" by "fetching objects in order of their importance." *Id.* at 3:50-56. Accordingly, "a priority scheme is used to determine the fetching, pre-fetching, and caching of data assets." *Id.* at 3:56-58. Although the system taught by Woods is flexible and may incorporate multiple factors to determine priority, the proximity to the camera is typically a key factor in the priority determination. For example, as shown in Fig. 5A, regions are prioritized based on the camera position and direction, with the region directly in front of the camera receiving the highest priority:

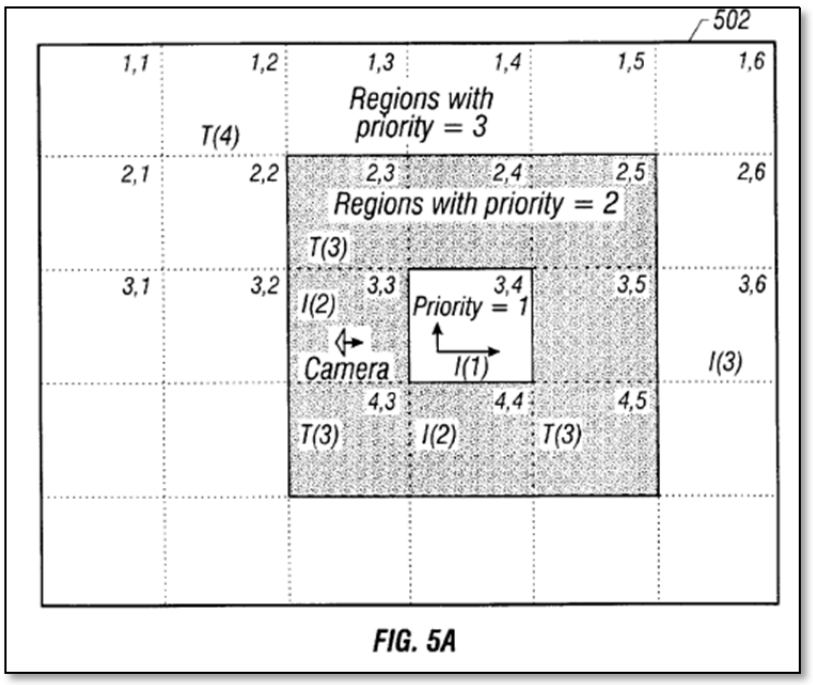


FIG. 5A

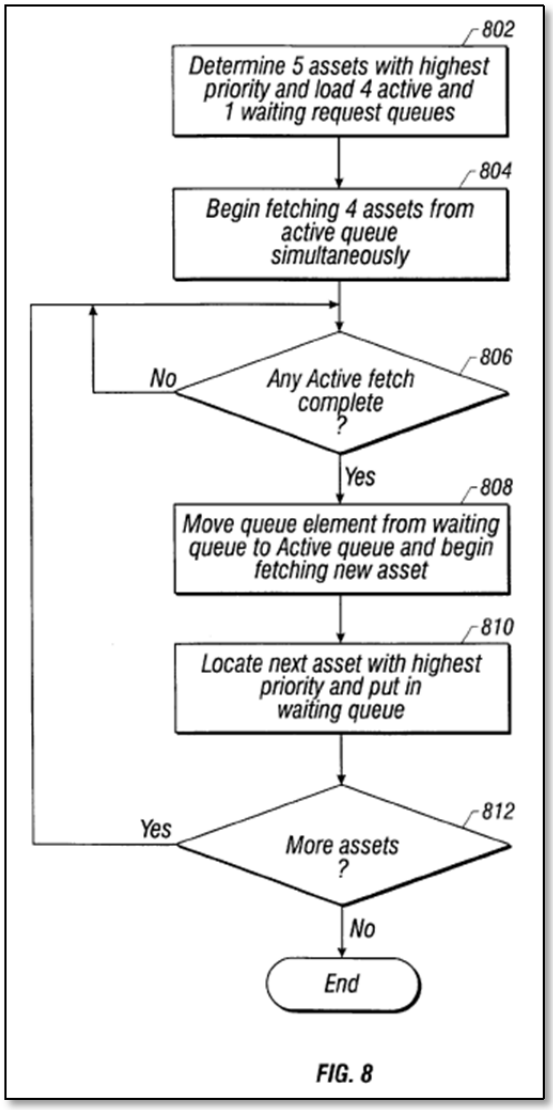
Id. at 9:19-10:64 (describing Fig. 5A).

165. Woods further teaches that the priority calculation may also take into account the movement of the camera; for example, more distant regions in the direction that the camera is moving may acquire more importance as the camera moves faster. *Id.* at 10:1-17.

166. Although Woods teaches that its prioritization scheme may be used to retrieve a variety of geographically linked "assets" including, for example, buildings, moving objects, and sounds, Woods specifically teaches that its prioritization scheme may be used to retrieve textures, which are used to "apply texture to geometric shapes after they are rendered." Such textures may be identified by "URLs from which textures can be obtained." *Id.* at 6:42-49. Woods

further teaches viewing VRML data at a variety of levels of detail ("LOD") using hierarchically-organized data. *Id.* at 6:24-35.

167. Woods prioritizes objects to be retrieved by first placing them in an "asset database table" which tracks (*inter alia*) the priorities of objects in the table. *Id.* at 11:66-12:5. Woods teaches that "[i]n a preferred embodiment, asset fetching is performed in an execution thread that is separate from the browser's runtime thread." *Id.* at 12:56-58. This fetching thread can support a "configurable queue of fetch elements comprising a number of 'active' fetches," so that "multiple assets can be fetched at the same time." *Id.* at 12:58-62. Although the number of simultaneous fetches may be configured "based on various hardware and software parameters" which are "apparent to those skilled in the relevant art(s)," Woods teaches a preferred embodiment featuring a fetch queue with four active fetch requests and one waiting fetch request, which becomes an active request when a previous active request is completed, as shown in Fig. 8:

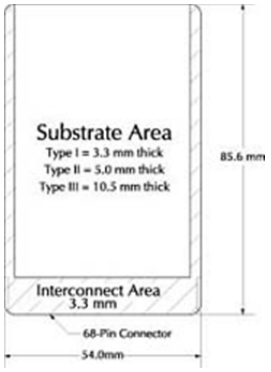


Id. at 12:56-13:47.

168. Woods teaches that a computer system suitable for performing the methods taught therein may include a communications interface 724, which may be a PCMCIA slot and which can carry signals over a cellular phone or RF link. *Id.* at 15:36-50. From my experience in the industry, I am aware that PCMCIA Personal Computer Memory Card International Association) cards were a

standardized format designed specifically to enable designers to adapt peripheral devices to laptop computers. The larger ISA expansion slots typically used in desktop computers were impractical for laptop computers because of their size. Additionally, for memory applications, traditional storage devices such as floppy disk drives and hard disks consumed too much power for laptop computers of the time, in addition to being too large. PCMCIA was designed to address these problems for laptops.

169. The PCMCIA organization was founded in 1989 and its membership throughout the 1990s included most major well known computer companies such as Fujitsu, Intel, Mitsubishi, IBM, Lotus, and Microsoft. This organization developed a series of standards for PCMCIA cards that could fit in laptops throughout the 1990s. These specifications defined the physical sizes and connection protocols for "PC card" peripheral devices that could fit into a laptop connection port. For example, a Type I PCMCIA card had a thickness of 3.3 mm, while a Type III card had a thickness of 10.5 mm. The figure below, from an archive capture of the PCMCIA organization website from the late 1990s, shows the dimensions of a typical PCMCIA card which could fit into a laptop:



170. According to the PCMCIA, "the power and versatility of PC cards quickly made them standard equipment in mobile computers." Such devices were used not only for memory, but for features such as wireless networks, modems, and other functions in notebook, laptop, palm-top, and other portable computers. The figure below, which is consistent with my recollection of the common appearance of PCMCIA devices in the 1990s from my own experience, shows a typical PCMCIA card device inserted into the slot on a laptop computer:

171. Because PCMCIA cards were specifically associated with an designed for laptop and other portable computers, it is my opinion that a person of ordinary skill in the art would therefore interpret the disclosure in Woods of PCMCIA cards to suggest use of a laptop or other portable computer.

2. Motivations to Combine Reddy and Woods

172. In my opinion, a person of ordinary skill in the art would combine Reddy and Woods for several reasons. First of all, both references are in the same field or closely analogous art. Reddy and Woods both relate to the retrieval of image data over the Internet in order to display an interactive, three-dimensional view of a world. Even more specifically, both references relates to browsing such data using VRML. In my opinion, a person of ordinary skill in the arts familiar with the teaching of one VRML reference such as Reddy to access the image data over the Internet would naturally look to the teachings of other VRML references such as Woods for potential improvements to such a system. Additionally, Reddy and Woods are both directed to similar problems because each reference relates to accessing portions of large image data sets over a network and retrieving the portions of the data set needed to display a particular view while optimizing the use of bandwidth. Such teachings would logically commend themselves to the attention of a person of ordinary skill in the art designing a system for routing geographic data over the Internet that taught in Reddy.

173. In particular, a person of ordinary skill in the art would recognize that one challenge faced by Reddy is optimizing the use of limited bandwidth and limited computing resources on a client device when accessing a large geographic database. For example, Reddy teaches in ¶48 that the software may be implemented on a PC or a laptop in a "distributed, time-critical" environment such as military mission planning, battle damage assessments, and emergency relief efforts. In my opinion, a person of ordinary skill in the art would recognize that mobility and portable access to a networks, like that provided by a wireless modem, would be extremely desirable in such circumstances. Additionally, Reddy specifically teaches that its system can be used on a PC (personal computer) or a laptop computer, which indicates that a primary purpose of the TerraVision II system is to expand on the previous TerraVision system by expanding it to be operable on a wide range of devices, including the type of devices suitable for mobile or field use in the scenarios described in Reddy. These teachings of Reddy would motivate a person of ordinary skill in the art to consider teachings relevant to how to (1) make the software operate effectively in on a portable device capable of operating in an environment such as emergency relief, and (2) access data remotely in a mobile context. In my opinion, based on my review of Reddy and Woods as well as my knowledge of the art of computer graphics and networking, it

is my opinion that a person of ordinary skill in the art would consider Woods relevant to both questions.

174. In particular, although Reddy teaches situations in which a wireless connection would be extremely desirable as discussed above, it does not explicitly teach a wireless connection. Woods, however, explicitly teaches methods of accessing data over the Internet using a wireless connection that are readily applicable to laptop devices like those taught by Reddy. Specifically, Woods teaches that the communications interface to access VRML data over the Internet may be a wireless link such as a cellular phone or RF link, implemented in a communications interface such as a PCMCIA card commonly used in laptop computers. Ex. 1003 at 15:36-50. As I discussed above, wireless connections using a PCMCIA card were extremely well known by the late 1990s and using such a connection to browse VRML data from a laptop as taught by Reddy would be a simple matter of using-the-shelf, already available technology. Accordingly, it is my opinion that a person of ordinary skill in the art would have virtually no difficulty making this modification and therefore would have a reasonable chance of success.

175. It is my understanding that prior art references should be considered for all that they teach and not merely for their preferred embodiments. In my opinion, a person of ordinary skill in the art would understand that Reddy teaches

broadly applicable methods of accessing and viewing geographic information over a network, not just the specific embodiments discussed as examples of these teachings. Even if the Board looks only at the separate TerraVision II and VRML browser embodiments, it would still be obvious to a person of ordinary skill in the art that both of these embodiments could be used on either a laptop computer or a tablet computer (PDA) and that the relevant features taught by Reddy could be used in connection with a VRML browser on either type of device. Reddy teaches that TerraVision II can be operated on a "PC connected to the Internet." In my opinion, a person of ordinary skill in the art would understand this teaching to mean that TerraVision II is software that can be operated on any appropriate common consumer computer hardware, whether that computer takes the form of a desktop computer, laptop computer, or tablet computer.

176. At the time of the alleged invention and the priority filing of the '645 Patent, processors for mobile devices (*e.g.* laptops) had been developed that offered similar processing power (*e.g.* clock speeds) to the processors used in most commonly known PCs. I reviewed Ex. 1031, which is a summary published by Intel of the key statistics of Intel processors over time, which confirms my

recollection that this is the case.[11] For example, as of October 1999, Intel had released a mobile processor with a 1GHz maximum clock speed. Ex. 1031 at 32-33 (showing 1 GHz and 1.13 GHz Pentium III Notebook Processors).

177. Additionally, at the time of the alleged invention, the operating systems that were commonly used on laptop computers (such as various versions of Windows, particularly Windows 95, Windows 98, Windows NT, and Windows 2000) were exactly the same as the most common operating systems used on desktop PCs. I personally installed Windows NT on laptop computers in the mid to late 1990s. Additionally, Microsoft offered a version of Windows (Windows CE) for even smaller portable devices which was based on Windows NT and could operate software based on similar coding languages, so that software written for Windows on a PC could be easily "ported" to a mobile computing device using techniques that would have been well-known to a person of ordinary skill in the art. For example, operating systems (including the various versions of Windows) would offer an Application Programmer Interface (API) to guide developers in

[11] In my opinion, a person of ordinary skill in the art would reasonably rely on such summaries published by Intel in order to evaluate the capabilities of processors at the time, and such summaries are generally reflective of the state of the art over time due to Intel's dominant market position.

writing software that can interact with the features of the operating system and its host computer through calls to those functions, so transposing software to a different operating system is generally a matter of updating the software calls to relate to the API for the new operating system. This was a very routine task in the software field. Indeed, Ex. 1014 itself notes specifically at p. 2 that "we have engineered TerraVision to be easily portable to other platforms and we have recently performed a port to Microsoft's Windows NT platform." Therefore, a person of ordinary skill in the art at the time (indeed, even an ordinary user who was not even a person of ordinary skill in the art) could have run and installed TerraVision II on a laptop computer with no modification. Therefore, a person of ordinary skill in the art would have every reason to expect that Reddy's teachings, including but not limited to TerraVision II itself, would operate on a suitable laptop or other mobile/portable computing device.

178. In my opinion, a person of ordinary skill in the art would further recognize that Reddy's use of commonly known Web browsers for accessing map information over the Internet would enable any device that is connected to the Internet, and which uses one of these known browsers, to access map data regardless of whether the connection is a wired connection or a mobile connection, and regardless of the speed at which that Internet connection operates. This is so because, as I discussed previously in section VI.A, the TCP/IP protocol provides a

layer of abstraction for the data being sent; in other words, as long as there is a means to transport TCP/IP packets, virtually any type of digital data can be sent using TCP/IP packets.

179. Therefore, applying the teachings of Reddy on a portable wireless device (*e.g.*, a laptop with a PCMCIA card connected to a wireless network, as taught by Woods) would require no more than the application of a known technique (a wireless network to access the Internet) with predictable results, and achieving the benefits of both references, because wireless connections allowed users to retrieve the same data from Internet as any other Internet connection. Additionally, there were at the time a finite number of known ways for a laptop computer to access the Internet (*e.g.* Ethernet plug, dial-up modem connection, internal or external modem), of which wireless PCMCIA cards were one very well-known method. Therefore, a person of ordinary skill in the art seeking to use a laptop to access geographic data over the internet would know that a wireless connection such as a wireless PCMCIA card was one of the known, finite ways to do that.

180. Additionally, Reddy teaches that tiles requested and downloaded from a server are prioritized using a "coarse-to-fine" algorithm to retrieve lower-resolution tiles first (Ex. 1004 at ¶¶ 21, 44) and by "prefetching" tiles along the user's predicted flight path (*id.* at ¶ 46). However, Reddy does not explain specific

technical details regarding how such tile requests are prioritized. Woods, however, explains in detail how VRML assets are prioritized for download to optimize use of bandwidth. Specifically, Woods teaches that the objects within a scene may be assigned priorities based on their distance from the viewpoint, as well as based on the motion of the viewpoint. Ex. 1003 at Abstract, 10:1-17. In my opinion, a person of ordinary skill in the art would recognize that the prioritization features of Woods would meet the goals taught by Reddy of retrieving needed data based on proximity to a viewpoint and projected flightpath, while improving the utilization of limited bandwidth to retrieve data, which is likewise a goal of Reddy. In other words, the priority weighing and priority queue of Woods would predictably improve fetching of imagery in Reddy (by using bandwidth to retrieve the data that contributes the most to a scene first) in the same manner as it improves fetching of imagery in Woods.

181. In my opinion, the teachings of the prioritization features of Woods are readily applicable to retrieving imagery tiles as taught by Reddy. For example, Woods teaches that the "assets" prioritized for retrieval using its priority scheme include inline nodes and textures. Ex. 1003 at 6:16-23, 6:42-49. For example, Fig. 5A of Woods shows an exemplary prioritization of inline and texture assets. Id. at 9:19-10:64, Fig. 5A. Reddy's hierarchical VRML data set relies on inlining and the terrain tiles include satellite or aerial imagery. Ex. 1004 at ¶¶ 3, 18, 19-21, 24.

A person of ordinary skill in the art would know that the satellite or aerial imagery tiles taught by Reddy are simply textures applied to the surface of the earth. Therefore, in my opinion such tiles could be retrieved just as readily using the priority fetching features of Woods as the building textures taught in the preferred embodiment of Woods.

3. Independent Claim 1 Is Obvious

182. Claim 1 is a method claim describing steps performed by one or more servers during interactions with a client. In my opinion, all limitations of Claim 1 are taught or suggested by Reddy in view of Woods.

Preamble: A method of communicating images for display, the method comprising steps of:

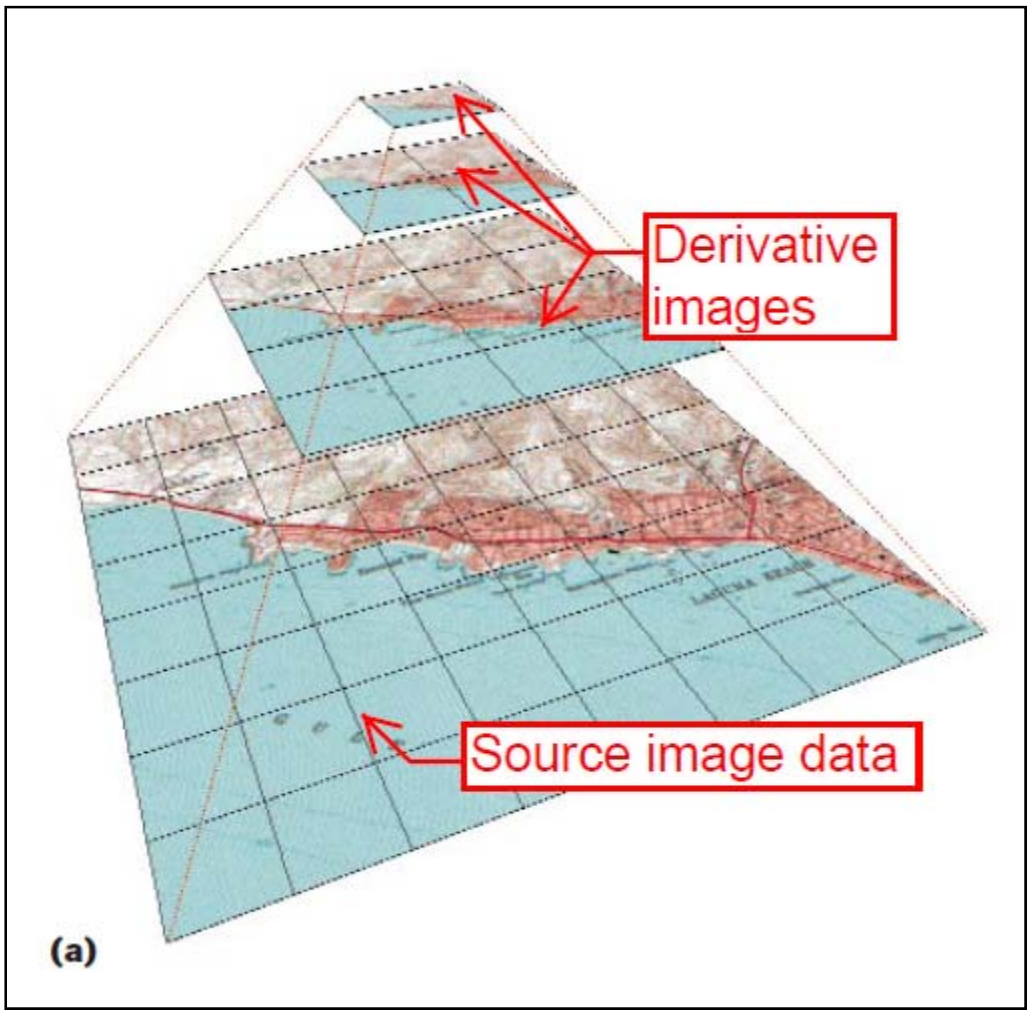
183. Both Reddy and Woods teach methods of communicating images for display. Reddy teaches communicating images (including map tiles) from a server to a client computer over a network such as the Internet. Ex.1004 at ¶¶ 3, 5, 15-17, 24, 44; Figs. 4-5; pg. 31 (sidebar). Woods similarly discloses communicating VRML data including 3D imagery from a server to a client over the Internet. Ex. 1003 at 2:34-43; 3:63-65; 7:37-47.

Element 1.A: processing data of a source image to obtain a series (K_0, K_{-1} . . . K_{1-N}) of related images of progressively lower image resolution,

184. In my opinion, Reddy teaches this element because it teaches that the server systems for providing imagery may process source image data into a grid of

tiles organized into a hierarchy of different resolution levels. Reddy discloses that the terrain data including the image pyramids may be pre-computed offline. Ex. 1004, ¶52. Reddy also discloses that terrain data may be generated "on the fly" by parsing the URL path name, using a script on the server, to generate the necessary VRML data. *Id.* In other words, in this second alternative, if processed data is not already available (which is a situation that might come up in a time-critical scenario, *e.g.* military or disaster response, where new imagery is being made available for a first time) the script can parse the URL request, decide what data needs to be made available, and then conduct the necessary processing in order to make that data available.

185. Reddy's Fig. 1 shows three derivative images created by down-sampling the source image data at the bottom of the pyramid:



186. While Woods does not describe processing the source image data, it does discuss levels of detail organized as a hierarchy such that an application can automatically select appropriate levels of detail for rendering based on the distance between the camera and the relevant area. Ex. 1003 at 6:23-35. In my opinion, a person of ordinary skill in the art would recognize that such level of detail hierarchies are created by processing a source image into a mip-map.

Element 1.B: wherein each related image of the series (K_0, K_{-1} . . . K_{1-N}) comprises image data and is subdivided into a regular array of image parcels,

187. Reddy explains that the image pyramid contains "down-sampled" images (i.e., image data). Ex. 1004 at ¶¶ 15-16. Reddy further teaches that each "pyramid image" (*i.e.*, each image in the series), including the highest resolution layer (K_0) is "segmented into" an array of tiles, and that each tile at a given level maps to four tiles at the next higher-resolution level. Ex. 1004, ¶¶12-16, Fig. 1. For example, in Fig. 1, the bottom layer is an 8x8 array of tiles. The rest of the "image pyramid" contains four different resolutions of the original image, each formed into an array ranging from 8x8 tiles at the highest resolution to a single tile at the lowest, with 4x4 and 2x2 arrays in between. This teaching is substantially identical to the '645 Patent's disclosure in Fig. 2 and at 6:23-38 of the division of source image data into derivative images of progressively lower image resolution, as shown by a comparison of Fig. 1 of Reddy and Fig. 2 of the '645 Patent:

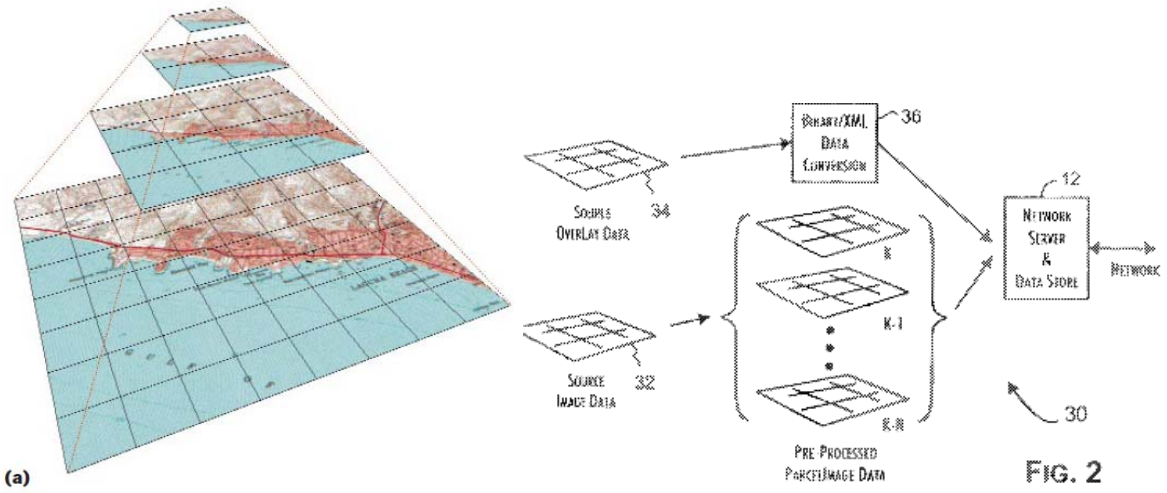


FIG. 2

188. As I discussed above in Section IX.D, I do not believe the term "image parcel" should be limited to the preferred identification scheme described in the '645 Patent which uses "the X and Y position in the image array coordinates and an image set resolution index." Nonetheless, this identification scheme would have been obvious even if it is required. A person of ordinary skill in the art would readily appreciate that the image parcels that make up any layer within Reddy's image pyramid form a two dimensional array. The standard way to identify an element of a two-dimensional array is to use an index for each dimension. An exemplary discussion of this is found in *Foundations of C++ and Object-Oriented Programming* by Namir C. Shammas (1998). Shammas explains, for example, that an element in a multidimensional array is accessed using the following syntax "arrayName[IndexOfDimension1][IndexOfDimension2]..." Ex. 1043 at 369. In my opinion, Shammas' discussion of multidimensional arrays is consistent with what a person of ordinary skill in the art would have learned in an introductory programming class by the late 1990s. I would also note that Woods uses this convention to identify particular grid regions which are similarly organized as an "array of grid elements" (Ex. 1003 at 8:46-52). The upper right corner grid region in Figs. 5A-5D is denoted "1, 6" because it is in the sixth position (X position) of the first row (Y position).

189. A person of ordinary skill in the art would further recognize that Reddy's image pyramid is a three-dimensional array of image parcels (or an array of two dimensional arrays) since there are several layers in the pyramid. The layers within the pyramid are a third dimension. And the standard way of accessing a three-dimensional array would be to use yet another index corresponding to the third dimension. *See, e.g.,* Ex. 1043 (Shammas) at 369; *id.* at 378 (example of three-dimensional array "fXMat" with three indices). The '645 Patent's preferred identification scheme which uses an X and Y position and an image set resolution index (which merely identifies the layer in the pyramid) is not inventive. It simply relies on an index for each dimension of a multi-dimensional array and is an obvious, perhaps the *most obvious*, way of identifying image parcels within Reddy's image pyramid.

Element 1.C: each image parcel of each regular array of the image parcels forming a discrete portion of the source image and having same predetermined pixel number and same predetermined color or bit per pixel depth,

190. In my opinion, Reddy teaches these limitations. First, Reddy's Fig. 1(a) shows that each image parcel of each derivative image corresponds to a discrete portion of the source image. There are not multiple tiles that show the same portion of the source image.

191. Reddy also teaches that within each pyramid image, "all tiles have the same pixel dimensions," and repeatedly discusses an example in which each tile is 128x128 pixels. Ex. 1004, ¶¶ 15-16, Fig. 1. This teaching is similar to the support for this limitation in the '645 Patent at 6:23-38, which says that "each resulting image parcel of the array has for example a 64 x 64 resolution."

192. In my opinion, there are at least two ways that a person of ordinary skill in the art reading Reddy in light of the existing knowledge in the art would recognize that Reddy teaches the use of image data having a predetermined color or bit per pixel depth. First, Reddy teaches the use of known imagery formats such as Portable Bitmap (PBM) and Land Analysis System (LAS) bitmaps. In my opinion, a person of ordinary skill in the art would be aware that both of these formats have a fixed bit per pixel depth. PBM (see http://oceancolor.gsfc.nasa.gov/staff/norman/seawifs_image_cookbook/faux_shuttle/pbm.html) is known in the art to be a monochrome format (1 bit/pixel), while LAS also uses bitmaps. In my opinion, a person of ordinary skill in the art would recognize both formats as having a fixed bit per pixel depth. Ex. 1004, p. 31 (sidebar).

193. Additionally, a person of ordinary skill in the art would be able to confirm that Reddy teaches tiles having a fixed bit per pixel depth based on the calculations given in ¶¶15-17. The size of the data representing an uncompressed

tile is simply the product of the bit depth (bits/pixel) multiplied by the pixel dimensions. For example, a 128x128 pixel tile (16,384 pixels) with 8-bit RGB color (*e.g.*, one byte for each of the three colors) would occupy approximately 49 Kbytes (1 byte is 8 bits) on disk.

194. Reddy discloses at ¶16 that the example in ¶15 and Fig. 1 takes up 491 Kbytes for 10 tiles and 3.1 Mbytes for the full high-resolution (1024x1024 pixel) image. A person of ordinary skill in the art would understand from this teaching that the data parcel size for each tile is the same because each tile has a bit depth of 24 bits per pixel, or 8-bit RGB (red, green, blue) color, yielding the total sizes in ¶16.

195. This teaching is similar to the support for this claim element in the '645 Patent at 6:28-34, which teaches that a 64x64 pixel parcel with a 16-bit color depth has a resulting data parcel size of approximately 8 Kbytes (using the 1024 bytes/KB convention).

Element 1.D: the step of processing being performed by one or more servers;

196. Element 1.D requires that the processing step recited in elements 1.A through 1.C are performed by one or more servers. Reddy discusses the "tsmAPI" library for generating the terrain datasets from "input formats such as raw imagery, Portable Bitmap (PBM) images, and Land Analysis System (LAS) bitmaps." Ex. 1004 at ¶¶ 11, pg. 31 (sidebar). Given Reddy's repeated discussions of "massive"

terrain datasets, it would be obvious that the image data would be processed using servers. Indeed, Reddy notes that "tsmApi distributions are available for Irix, Solaris, Linux, and other platforms." Ex. 1004 at pg. 31 (sidebar). A POSITA would have known that Irix and Solaris are operating systems Silicon Graphics ("SGI") and Sun's UNIX computers, respectively, and were commonly used on web servers. *Web Performance Tuning* (1998) by Patrick Killelea, is an O'Reilly reference book that provides an exemplary discussion. *See, e.g.,* Ex. 1044 (Killelea, 1998) at 184-187 (discussing UNIX operating systems for servers including Irix and Solaris), 206-208 (discussing advantages of UNIX-based servers).

197. Reddy also indicates that the full image pyramids are stored on the server(s) so that data of an appropriate resolution may be fetched by a client. Ex. 1004 at ¶ 17. It thus would have been obvious that the servers that store and serve the terrain data could be the same servers that process it. Indeed, Reddy discloses that terrain data could be generated on the fly via a Common Gateway Interface ("CGI") script. Ex. 1004 at ¶ 52. A POSITA would know that a CGI script would run on the server. *See, e.g.,* Ex. 1045 (MSFT Dictionary) at 80. Thus, Reddy's disclosure of dynamically generating the terrain data also suggests processing the image data on the same server(s) that handle that service the client requests.

Element 1.E: receiving a first request at the one or more servers from a wireless portable device over a network communication channel, the first request being for a first image parcel of the series, wherein the first image parcel is selected based on a first user-controlled image viewpoint on the wireless portable device relative to the source image;

198. Reddy and Woods disclose this element. Reddy teaches that clients fetch map tiles (*i.e.*, image parcels) based on a user-controlled image viewpoint. Reddy's Fig. 1(b) shows 10 tiles of varying resolutions that would be downloaded and rendered if the user's viewpoint is in the bottom-right corner. Ex. 1004 at ¶¶ 16-17.

199. Woods discloses that clients download VRML data files from a "web server" over a network. Ex. 1003 at 5:15-48, Fig. 1. Woods describes the use of URLs to request VRML objects including textures (*i.e.*, images). *Id.* at 3:27-47, 6:42-49, 13:20-30. These teachings are similar to the '645 Patent's disclosure of "HTML-based interactions with the server." Ex. 1001 at 7:31-35. Woods also discloses that the selection and prioritization of VRML resources for download is based on the camera which is a user-controlled viewpoint. Ex. 1003 at 7:66-8:8 (prioritized list of downloads is responsive to user movement of the camera).

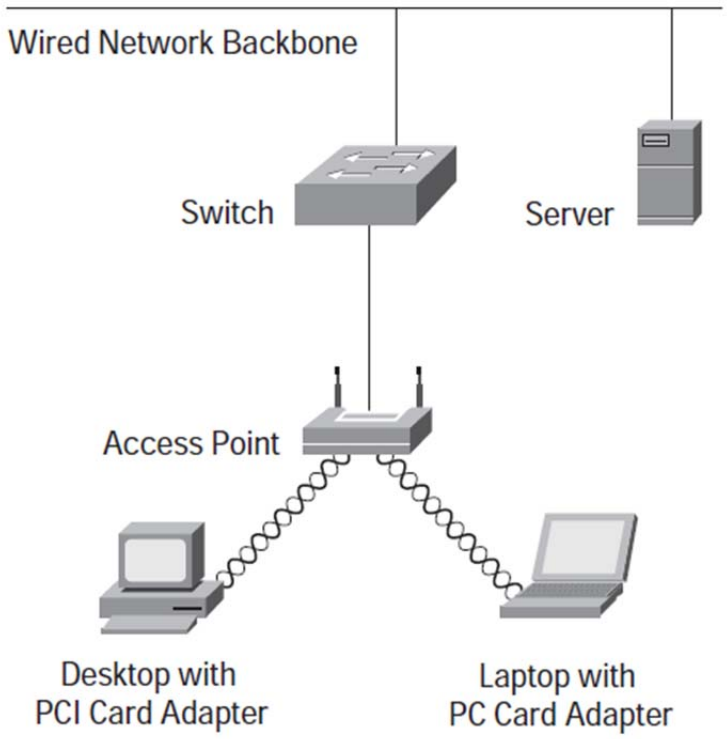
200. Regarding the "wireless portable device" requirement, Reddy discloses use of laptops (¶ 43) which were seen as portable. For example, the Microsoft Computer Dictionary (1999) includes a table of exemplary "portable computers" and the table includes laptops. *See, e.g.*, Ex. 1045 at 349-350. Woods

discloses that the client computers may communicate wirelessly with the servers. Ex. 1003 at 15:36-50 (channel 728 may be a "cellular phone link" or "RF link"). Woods also discloses that the communications interface of the client computing device may be a "PCMCIA slot and card." *Id.* at 15:39-42. PCMCIA (or PC Card) refers to a card-based interface for peripheral devices that was developed for laptops and other portable computing devices. *See* Ex. 1045 (MSFT Dictionary) at 336 (noting that PCMCIA standard was "primarily [for] laptop, palmtop, and other portable computers"). PCMCIA cards having wireless capabilities, as discussed in Woods, were widely available prior to the alleged invention and filing date of the '645 Patent.

201. For example, by 1999, Proxim, Inc. offered a RangeLAN2 PC Card designed to provide high speed wireless internet connections to "mobile users who require continuous LAN connectivity." Ex. 1051 at 1. This device, which was designed for "Windows CE Handheld PCs," offered a 1.6 Mbps data rate and is shown below:

Id.

202. As another example, by 2000, Cisco's Aironet 340 series of products included a PC Card wireless adapter for laptop computers, which supported data rates up to 11 Mbps. Ex. 1052. Such devices allowed a laptop computer to connect to a wireless router using a PC card in the laptop, as shown in the following diagram:



203. The Cisco Aironet products could use the IEEE 802.11 wireless internet or "wi-fi" standard, which is specifically mentioned in Ex. 1052.

204. More generally, it was well known by 2000 that portable laptop computers could connect wirelessly to networks and the Internet. By that time,

IBM was touting the wireless capabilities of its ThinkPad laptops. *See, e.g.*, Ex. 1050 (Thinkpad X press release). For example, IBM advertised that its Thinkpad X series Ultraportable computer offered "[f]or wireless LAN connectivity, an optional industry-standard 802.11b LAN PC Card" as well as Bluetooth wireless technology options, including a PC card.

205. Likewise, by 1999 Apple Computer, Inc. had also released its "AirPort" family of wireless connection products based on the IEEE 802.11 family of wireless connection standards. Ex. 1048, 1049. Among other products, the 11-Mbps AirPort card was available for Macintosh computers including the PowerBook, which was well known as an Apple marketing name for notebook computers:

AirPort Card
The 11-Mbps AirPort Card is available for all new Macintosh computers—iBook, iMac, PowerBook, and Power Mac G4. The credit-card-size card is user installable and comes with all the software needed to set up and operate an AirPort network. The same card is used in all AirPort-ready computers.

Ex. 1049 at 3.

206. I personally recall being aware as of 1999 and 2000 that similar wireless access devices on PC Cards were available, and the exhibits discussed above are consistent with my recollection.

207. Therefore, in my opinion a person of ordinary skill in the art would recognize that a "wireless portable device" is taught or suggested by the

combination of Reddy and Woods through Reddy's teaching of a laptop and Woods' teaching of wireless connections suitable for laptops. In my opinion, a person of ordinary skill in the art would also recognize that using such a wireless connection as taught by Woods on a laptop as taught by Reddy could be accomplished using methods well-known in the art with predictable results.

Element 1.F: sending the first image parcel from the one or more servers to the wireless portable device over the network communication channel, in response to the first request;

208. Reddy teaches a system for browsing geographic data over the Internet (network communications channel), in which a server sends map tiles (image parcels) in response to requests. A person of ordinary skill in the art would recognize that in the client-server interaction described in Reddy, the server provides the tiles after they are requested by the client user computing device. Specifically, persons of skill would have understood how web browsers and other client applications make HTTP requests for resources over the Internet and servers respond to those requests by providing the requested resource.

209. Similarly, Woods describes the fetching of VRML resources from a web server using URLs. Ex. 1003 at 2:31-37, 3:30-35, 5:14-25, 5:39-48. As I mentioned above, a person of ordinary skill would have been well aware of how clients use URLs to retrieve resources from servers on the Internet. *HTML & XHTML: The Definitive Guide* (Aug. 2000) by Chuck Musciano & Bill Kennedy,

an O'Reilly reference book contains an exemplary discussion of this subject matter. Ex. 1053 at 171-179.

Element 1.G: receiving a second request at the one or more servers from the wireless portable device over the network communication channel, the second request being for a second image parcel of the series, wherein the second image parcel is selected based on the first user-controlled image viewpoint or on a second user-controlled image viewpoint on the wireless portable device relative to the source image, the step of receiving the second request being performed after the step of receiving the first request;

210. This element is similar to element 1.E except that it recites a second request for a second image parcel. The second image parcel is selected either based on the first viewpoint (*i.e.*, the same viewpoint that led to the request for the first image parcel) or a second viewpoint (*i.e.*, a different viewpoint). This element also requires that the second request is received after the first request referenced in element 1.E.

211. As I discussed with respect to element 1.E, Reddy discloses that clients fetch image parcels based on a user-controlled image viewpoint. For example Fig. 1(b) shows 10 tiles of varying resolutions that would be downloaded and rendered if the user's viewpoint is in the bottom-right corner. Ex. 1004 at ¶ 16; *see also id.* at ¶ 17 ("[W]e need only fetch and display data for the region that the user is viewing, and only at a sufficient resolution for the user's viewpoint."). This also confirms that more than one is downloaded based on the same viewpoint (*e.g.,* a first user-controlled image viewpoint).

212. Woods discloses prioritized downloading of VRML resources including textures (*i.e.*, images) based on a user-controlled viewpoint. For example, Woods describes and shows via Fig. 5A an example wherein based on the camera position and direction, one grid region is designated priority 1, eight others are designated priority 2, and twelve others are designated priority 3. Ex. 1003 at 8:55-10:64. Thus, multiple resources (*e.g.*, images) are downloaded for the same camera position (*i.e.*, a first user-controlled image viewpoint). Additionally, because resources are downloaded in priority order (*id.* at 13:5-20), the requests for VRML assets associated with a priority 1 grid region would be made (and thus received by the server) before those associated with the priority 2 and 3 grid regions. Based on the combined teachings of Reddy and Woods, a person of skill in the art would understand that map tiles corresponding to the higher priority grid regions, *e.g.*, those in the field of view given the current viewpoint position and orientation, would be requested before those associated with lower priority grid regions, *e.g.*, those behind the camera or outside the field of view.

213. In the preceding paragraphs I focused on scenarios where both the first and second image parcels would be selected based on the same viewpoint (*i.e.*, a first user-controlled image viewpoint). But element 1.G also allows for the alternative that the second request is for an image parcel associated with "a second user-controlled image viewpoint." Because both Reddy and Woods teach that

images are downloaded based on the viewpoint position, a person of skill in the art would recognize that different imagery (*i.e.*, a second image parcel) would be fetched after a user moves the viewpoint to a second location. Woods further discloses that priority is reevaluated when the user moves the camera. Ex. 1003 at 7:66-8:8, 11:59-65. Thus, while the camera is at a first position, the client will download the high priority resources for that position. When the user moves the camera to a second position, the client will download the high priority resources for that position. Naturally, then, resources associated with a second viewpoint will be requested after the resources associated with a prior viewpoint and a server would receive those requests associated with a second viewpoint after the requests associated with a prior viewpoint.

Element 1.H: sending the second image parcel from the one or more servers to the wireless portable device over the network communication channel, in response to the second request.

214. Reddy and Woods disclose element 1.H. This claim element is nearly identical to claim element 1.F, except that it relates to the second image parcel rather than the first. It would be obvious to a POSITA in view of Reddy and Woods that subsequent tiles requested would be sent in the same manner as the "first" tile regardless of whether they are associated with a first viewpoint or a second viewpoint. Therefore, element 1.H is disclosed for the same reasons I discussed above for element 1.F.

4. Independent Claim 13 Is Obvious

215. Claim 13 is styled as a system claim in which the servers are configured to perform certain actions. Nonetheless, the substantive limitations of Claim 13 are virtually identical to those of Claim 1 discussed above. Accordingly, I will discuss the unique limitations of Claim 13 but refer to my discussion above for those limitations that are substantially identical to limitations of Claim 1.

Preamble: A computing system comprising one or more servers, wherein the one or more servers are coupled to a wireless portable device by a network communication channel, the one or more servers being configured to:

216. A person of ordinary skill in the art would understand Reddy to disclose a computing system in which clients connect to one or more servers over a network such as the Internet (a network communication channel). Ex. 1004, ¶48 (client can be a "PC connected to the Internet"). Woods also describes servers (including "Internet web server[s]") that communicate with clients over a network such as the Internet. Ex. 1003 at 5:15-38. Reddy and Woods also disclose a "wireless portable device" client as discussed above in addressing Element 1.E.

Element 13.A: process data of a source image to obtain a series (K_0, K_1 . . . K_{1-N}) of related images of progressively lower image resolution,

217. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.A.

Element 13.B: wherein each related image of the series (K0, K1 . . . K1-N) comprises image data and is subdivided into a regular array of image parcels,

218. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.B.

Element 13.C: each image parcel of each regular array of the image parcels forming a discrete portion of the source image and having same predetermined pixel number and same predetermined color or bit per pixel depth,

219. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.C.

Element 13.D: resolution of each related image of the series except initial of the related images in the series being related to resolution of the immediately preceding related image in the series by a factor of four, number of image parcels into which each related image of the series except the initial of the related images is subdivided being related by a factor of four to number of image parcels into which the immediately preceding related image in the series is subdivided;

220. Reddy discloses Element 13.D. Reddy discloses processing imagery into a pyramid where the next higher resolution image in the series has four times the resolution. For example, a 1024x1024 original image (1,048,576 pixels) gets down-sampled to 512x512 pixels (262,144 pixels, *i.e.*, 1,048,576 ÷ 4), then 256x256 pixels (65,536 pixels, *i.e.*, 262,144 ÷ 4), and so on. Ex.1004, ¶¶14-15, Fig.1. Thus the resolutions between consecutive images in the series are related by a factor of four.

221. Additionally, each level in Reddy's pyramid includes four times the number of tiles at the preceding level. For example, the second image from the top

includes 4 tiles (4 x 1), the next includes 16 (4 x 4) and the next includes 64 (16 x 4). This is the same "factor of four" relationship between images in the series that is described for the preferred embodiment of the '645 Patent. Ex.1001, 6:34-38.

Element 13.E: receive a first request from the wireless portable device over the network communication channel, the first request being for a first image parcel of the series, wherein the first image parcel is selected based on a first user-controlled image viewpoint on the wireless portable device relative to the source image;

222. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.E.

Element 13.F: send the first image parcel from the one or more servers to the wireless portable device over the network communication channel, in response to the first request;

223. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.F.

Element 13.G: receive a second request at the one or more servers from the wireless portable device over the network communication channel, the second request being for a second image parcel of the series, wherein the second image parcel is selected based on the first user-controlled image viewpoint or on a second user-controlled image viewpoint on the wireless portable device relative to the source image, wherein the second request is received after the first request; and

224. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.G.

Element 13.H: send the second image parcel to the wireless portable device over the network communication channel, in response to the second request.

225. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.H.

5. Independent Claim 25 Is Obvious

226. Claim 25 is a method claim which describes *from the client-side* substantially the same steps recited in Claim 1 discussed above. The substantive limitations of Claim 25 are virtually identical to those of Claim 1 discussed above. Accordingly, I will discuss the unique limitations of Claim 25 but refer to my discussion above for those limitations that are substantially identical to limitations of Claim 1.

Preamble: A method of communicating images for display, the method comprising steps of:

227. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Preamble.

Element 25.A: sending a first request from a wireless portable device to one or more servers over a network communication channel, the first request being for a first image parcel, the first image parcel being selected based on a first user-controlled image viewpoint on the wireless portable device relative to a source image;

228. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.E.

Element 25.B: receiving the first image parcel by the wireless portable device from the one or more servers over the network communication channel, in response to the first request;

229. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.F.

Element 25.C: sending a second request from the wireless portable device to the one or more servers over the network communication channel, the second request being for a second image parcel, the second image parcel being selected based on the first user-controlled image viewpoint or on a second user-controlled image viewpoint on the wireless portable device relative to the source image, the step of sending the second request being performed after the step of sending the first request; and

230. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.G.

Element 25.D: receiving the second image parcel by the wireless portable device from the one or more servers over the network communication channel, in response to the second request;

231. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.H.

Element 25.E: wherein the source image is processed by one or more servers

232. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.D.

Element 25.F: to obtain a series (K0, K1 . . . K1-N) of related images of progressively lower image resolution,

233. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.A.

Element 25.G: wherein each related image of the series (K0, K1 . . . K1-N) comprises image data and is subdivided into a regular array of image parcels,

234. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.B.

Element 25.H: each image parcel of each regular array of the image parcels forming a discrete portion of the source image and having same predetermined pixel number and same predetermined color or bit per pixel depth,

235. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 1, Element 1.C.

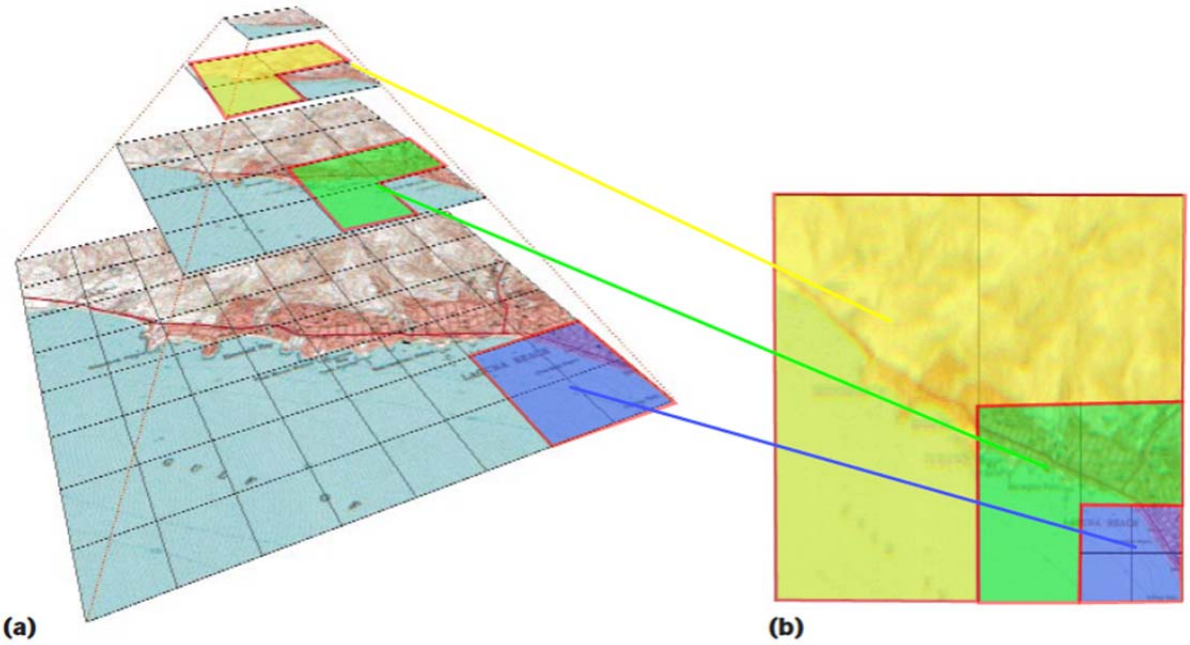
Element 25.I: resolution of each related image of the series except initial of the related images in the series being related to resolution of the immediately preceding related image in the series by a first predetermined factor, number of image parcels into which each related image of the series except the initial of the related images is subdivided being related by a second predetermined factor to number of image parcels into which the immediately preceding related image in the series is subdivided, and

236. In my opinion, this claim element is obvious in view of the teachings that I previously discussed in regard to Claim 13, Element 13.D.

Element 25.J: wherein the series comprises the first image parcel and the second image parcel.

237. Element 25.J further specifies that the first and second image parcels requested by the client and received from the server are among the series of images generated by processing the source image. Reddy discloses that the imagery tiles downloaded by the client are part of an image pyramid generated by processing the source imagery. *See* Ex. 1004, ¶¶ 15-17. For example, each of the 10 tiles in Fig

1(b) is one of the tiles from the pyramid in Fig. 1(a), as shown in the annotated figures below.



6. Dependent Claims 2, 14, and 26 Are Obvious

Claim 2: The method of claim 1, further comprising: [a] providing client software to the wireless portable device; [b] wherein the wireless portable device renders at least a portion of the first image parcel before finishing receiving the second image parcel.

Claim 14: The computing system of claim 13, [a] wherein the one or more servers are further configured to provide client software to the wireless portable device; and [b] wherein the wireless portable device renders at least a portion of the first image parcel before finishing receiving the second image parcel.

Claim 26: The method of claim 25, further comprising rendering by the wireless portable device at least a portion of the first image parcel before finishing receiving the second image parcel.

238. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims and Claim 26 omitting the first element (sub-element [a]) of Claims 2 and 14. Therefore, I will discuss these claims together.

239. In my opinion, sub-element [a] of Claims 2 and 14 is taught by and obvious in view of Reddy and Woods. As an initial matter, the '645 Patent does not clearly state or provide any detail as to how client is software is "provided" to a client device as claimed. The closest disclosure in the '645 Patent is its teaching that the client software "is preferably implemented by software plug-in or application executed by the client system . . . that utilizes basic software and hardware services provided by the client system." Ex. 1001 at 7:28-31. Nothing in this teaching specifies whether software is "provided" to a client device prior to runtime, at runtime, or how it is loaded onto the client device, such as whether the server sends the software to the client or whether the software is provided by some other means. Nor does the claim language specify *what* client software is provided to the wireless portable device, because none of the other claim elements or dependent claims refer back to the client software. However, it is my opinion that Reddy and Woods disclose providing client software to a wireless portable device to at least the same extent, if not more, that the term is supported by the specification of the '645 Patent.

240. For example, Reddy teaches using Java scripts and applets in connection with a browser plug-in to enable users to view data. Ex. 1004 at ¶¶ 10, 26, 32-33. Browser plug-ins were typically software designed to extend the functionality of an existing browser, and so would be installed in addition to the browser that was already installed. A person of ordinary skill in the art as of 1999 or 2000 would also be familiar with both Java scripts and applets. Java scripts and applets were typically small applications that would be downloaded from a server to a client when the client device viewed a web page. Moreover, it was standard practice by the late 1990s for web sites that used plug-ins to provide the plug-ins for download or at least to provide links to where the plug-ins could be downloaded.

241. Woods further teaches that computer programs enabling the computer system "to perform the features of the present invention" may be received via the communications interface. Ex. 1003 at 15:58-16:8. Therefore, in my opinion, a person of ordinary skill in the art would understand that both Reddy and Woods teach or suggest providing client software to the user device, which may be a wireless portable device, from a server.

242. The limitation in sub-element [b] of Claims 2 and 14 (and the only limitation in Claim 26) is also obvious. For example, in addition to its disclosure that imagery is downloaded from "coarse to fine," Reddy notes that "[m]ost

VRML browsers perform nonblocking network reads so that the user can still interact with the scene while higher resolution imagery and elevation loads." Ex. 1004, ¶¶ 21, 44. A person of ordinary skill in the art would understand that "non-blocking reads" means that the network reads (that is, the requests for data over the network) do not occupy (that is, "block") the processor the entire time that they are being executed. This allows other functions, such as rendering the scene with the data that is received, to be performed while data is still being downloaded from the network. In my opinion, a person of ordinary skill in the art would understand this disclosure to confirm that imagery is rendered while additional imagery is received. This is necessarily the case because unless some imagery was rendered while other imagery was being downloaded for a particular scene, there would be nothing for a user to "interact" with.

243. Reddy discloses other scenarios which make it clear that some imagery is rendered while other imagery is still being downloaded. For example, Reddy discloses "walk" and "fly" navigation modes. These modes would have to allow for tiles to be displayed while other tiles are still being requested and received, because the needed tiles would be identified while the viewpoint is moving as the viewpoint is controlled by the user. Reddy also discloses pre-fetching of the tiles identified by predicting the user's path of movement. Ex. 1004, ¶ 46. The pre-fetched tiles "are immediately available for rendering." *Id*.

244. In my opinion, Woods also teaches or suggests this element because Woods repeatedly refers to which assets are needed first for display. Indeed, retrieving assets first needed for display so that those assets can be displayed while lower-priority objects are being retrieved is a central purpose of Woods. *See*, *e.g.*, Ex. 1003, 4:56-59 (invention prioritizes "that data which is most likely perceived"); 7:36-55 (describing prioritization of objects that "user will be able to interact with" to avoid spending time "fetching images which are not immediately rendered because they are not visible to the user"). A person of ordinary skill in the art would understand from these teachings that earlier retrieved, higher priority objects are rendered while later, lower priority objects are still be requested. Woods also teaches that "asset fetching is performed in an execution thread that is separate from the browser's runtime thread." Ex. 1003, 12:56-6. A person of ordinary skill in the art would understand from this teaching that rendering of imagery retrieved by the browser would be performed by the browser's runtime thread, since asset fetching is the only function separated into a distinct thread. The purpose of such multi-threading is to enable different functions to be performed simultaneously (or very nearly simultaneously, to an extent imperceptible to a human user) without the processor needing to switch between different processes. In my opinion, a person of ordinary skill in the art would recognize that this feature would enable a user to interact with earlier downloaded

objects which are rendered by the browser runtime thread while other objects are being fetched by the asset fetching execution thread. And similar to Reddy, Woods describes cameras which move at different speeds which a person of ordinary skill in the art would understand would necessitate iterative rendering of VRML resources downloaded based on the movement of the camera just as in Reddy's "walk" and "fly" navigation modes.

245. In my opinion, this element is also obvious because it merely recites a conventional and well-known feature of Web browsers at the time of the alleged invention and of the earliest asserted priority date. By 1999 and 2000, many common web browsers, such as Netscape, downloaded images and rendered them one at a time as they were downloaded, without waiting for the remaining images to be downloaded before rendering. *See, e.g.,*. Ex. 1054 (Brown, Using Netscape 2) at 95, 443 (showing "Display Images: While Loading" setting). Woods notes that VRML browsers operate similarly to web browsers. Ex. 1003, 2:34-47. A person of ordinary skill in the art would read this teaching of Woods in view of the common knowledge in the art that web browsers typically rendered and retrieved data simultaneously.

7. Dependent Claims 3, 15, and 27 Are Obvious

Claim 3: The method of claim 2, wherein: [a] the wireless portable device issues the first request and the second request according to a priority order; [b] priority of the second request in the priority order is not higher than

priority of the first request in the priority order; and [c] wherein the wireless portable device stores the first image parcel and the second image parcel in a local parcel storage.

Claim 15: The computing system of claim 14, wherein: [a] the wireless portable device issues the first request and the second request according to a priority order; [b] priority of the second request in the priority order is not higher than priority of the first request in the priority order; and [c] wherein the wireless portable device stores the first image parcel and the second image parcel in a local parcel storage.

Claim 27: The method of claim 26, wherein: [a] the steps of sending the first request and sending the second request are performed according to a priority order; and [b] priority of the second request in the priority order is not higher than priority of the first request in the priority order; [c] the method further comprising storing the first image parcel and the second image parcel in a local parcel storage of the wireless portable device.

246. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims. Therefore, I will discuss these claims together.

247. Reddy and Woods disclose sub-elements [a] and [b]. Woods discloses *prioritized* fetching of VRML resources. Ex. 1003 at 3:50-58, 4:56-59. At any given time, the highest priority resources are downloaded by the client. *Id.* at 13:5-20. Due to Woods' priority scheme a later-issued (second) request would obviously not have a higher priority than a prior-issued (first) request. Based on the combined teachings of Reddy and Woods, a person of ordinary skill in the art would understand that requests for map tiles could be prioritized based on the position and direction of the camera (viewpoint).

248. Both Reddy and Woods disclose sub-element [c] because they disclose client-side caches which are local parcel storage that would be used to cache downloaded resources such as map tiles. Ex. 1004 at ¶¶ 21, 45; Ex. 1003 at 3:56-60, 14:18-27.

8. Dependent Claims 4, 16, and 28 Are Obvious

Claim 4: The method of claim 2, wherein the wireless portable device stores the first image parcel and the second image parcel received by the wireless portable device in a local store of the wireless portable device.

Claim 16: The computing system of claim 14, wherein the wireless portable device stores at least some image parcels received by the wireless portable device in a local store of the wireless portable device.

Claim 28: The method of claim 26, further comprising storing the first image parcel and the second image parcel received by the wireless portable device in a local store of the wireless portable device.

249. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims. Therefore, I will discuss these claims together.

250. As just discussed for Claims 3, 15, and 27, both Reddy and Woods disclose client-side caches which are local stores of the client devices that are used to store downloaded resources including map tiles. Ex. 1004 at ¶¶21, 45; Ex. 1003 at 3:56-60, 14:18-27.

9. Dependent Claims 5, 17, and 29 Are Obvious

Claim 5: The method of claim 4, wherein the first user-controlled image viewpoint is determined based on navigational input of the wireless portable device.

Claim 17: The computing system of claim 16, wherein the first user-controlled image viewpoint is determined based on navigational input of the wireless portable device.

Claim 29: The method of claim 28, further comprising determining the first user-controlled image viewpoint based on navigational input of the wireless portable device.

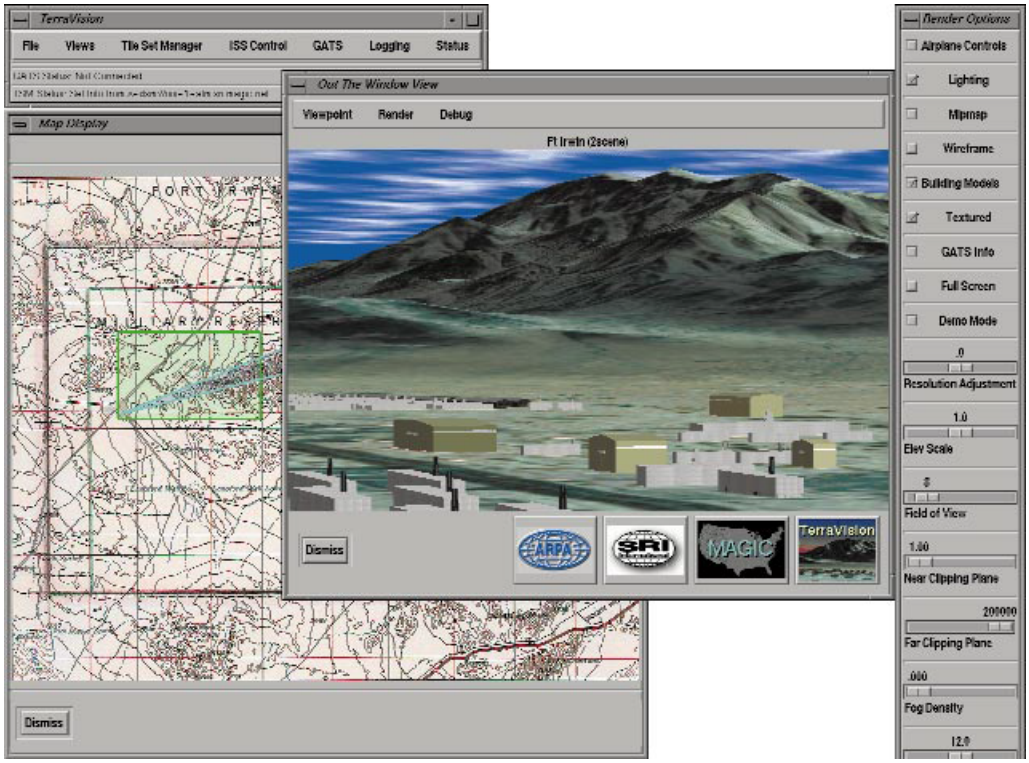
251. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims. Therefore, I will discuss these claims together.

252. As I previously discussed in regard to claim element 1.E, Reddy teaches that a user may select an image viewpoint, for example, by "flying" and zooming into an area of interest. For example, in ¶ 3, Reddy describes the scenario as follows:

> The following scenario indicates the capabilities required. Say a user wants to find a particular building in a particular city. Her journey begins with a 3D model of the earth viewed from space. This model is texture mapped with satellite imagery of 100 kilometers resolution— that is, each pixel in the texture map represents a region on the planet's surface covering 100 km^2. To find the city, **the user first rotates the earth** to

> view the target region in more detail. **As she zooms into the region**, higher resolution data, such as elevation and imagery, are progressively downloaded and displayed until she is "flying" over mountains with imagery down to one-meter resolution. Over certain parts of the terrain, alternative imageries are available, such as aerial photographs; the user can select any image to view on top of the terrain geometry. **As she approaches a built up area**, 3D models of buildings come into view. When the user clicks on a building, information about it is displayed in a separate frame on the browser. Using this method, the user locates the target building. Throughout the navigation, the user's location is displayed via an active map interface that provides a context for the landscape being viewed.

253. In my opinion, it would be clear to a person of ordinary skill in the art in view of this teaching that the navigation system taught by Reddy would require navigational inputs on the user computing device to determine a viewpoint. Reddy teaches an alternative form of navigational input for determining a viewpoint in Fig. 5 and the accompanying text at paragraph 37:



Active maps. When flying over terrain, it's often difficult for users to maintain a global context for their position. We thus employ a map display, managed by a Java applet. Through the EAI, we can obtain the user location in the geographic environment. We might do this, for example, using the position_changed eventOut of a ProximitySensor placed around the entire scene. We can then project this 3D geocentric coordinate onto the map display so users can easily ascertain their location in the world. **Users can also click over the map and then move the viewpoint directly to that location**. We do this by updating and binding a Viewpoint node in the VRML scene graph.

254. In my opinion, this teaching of Reddy also satisfies this claim element.

255. Woods similarly discloses that VRML allows users to navigate spaces (*e.g.*, "the streets of a city") by controlling the viewpoint of a virtual camera using an input device. Ex. 1003, 2:39-43 ("3D rendering is performed from the viewpoint of a virtual camera that has the ability to move and tilt in any direction ***in response to user input, via a mouse, keyboard or other input device***."); 2:48-56 (describing navigation through "three dimensional worlds"). Woods further discloses prioritizing the download of VRML resources based on the user-controlled camera position / viewpoint. Ex. 1003, 7:66-8:8, 8:55-59, 11:62-65. In my opinion, these teachings of Woods also disclose that a viewpoint is determined in response to a navigational input (*e.g.* via the input devices described).

10. Dependent Claims 6, 18, and 30 Are Obvious

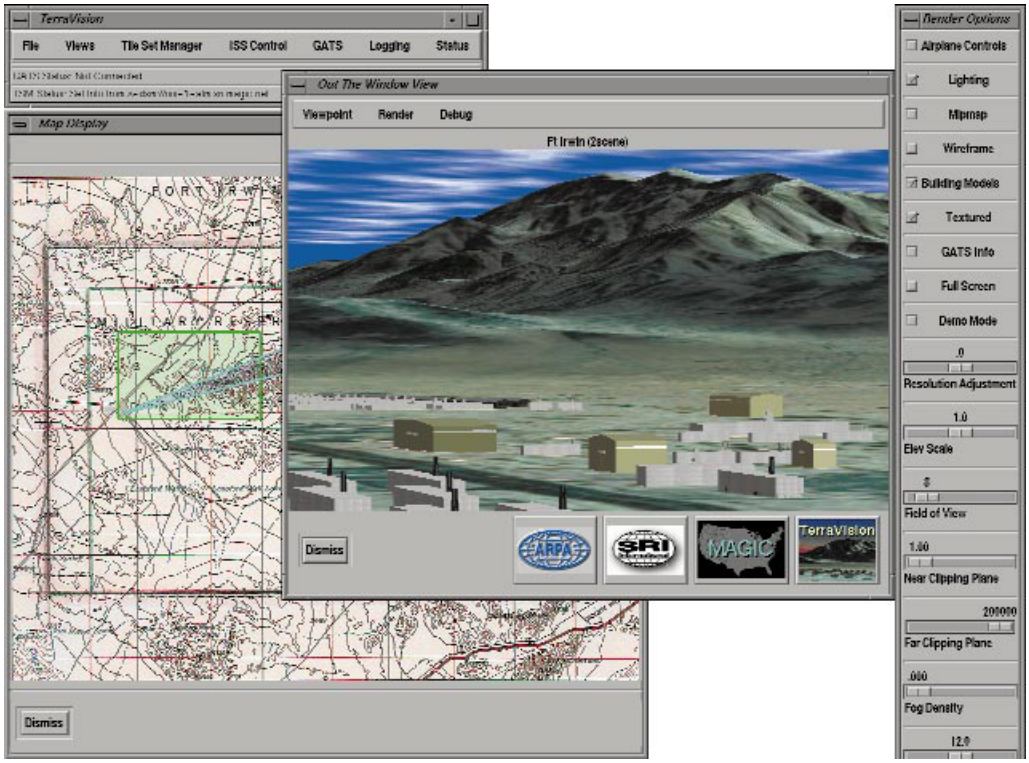
Claim 6: The method of claim 5, wherein the navigational input comprises three-dimensional positional coordinate data and rotational positional data.

Claim 18: The computing system of claim 17, wherein the navigational input comprises three-dimensional positional coordinate data and rotational positional data.

Claim 30: The method of claim 29, wherein the navigational input comprises three-dimensional positional coordinate data and rotational positional data.

256. In my opinion, these claim limitations are substantially similar to each other, other than depending from different independent claims. Therefore, I will discuss these claims together.

257. In my opinion, it would be obvious to a person of ordinary skill in the art that displaying a perspective view from a viewpoint would require at least x, y, and z (altitude or height) coordinates, as well as the direction of view (rotational position data, although the specification of the '645 Patent never uses this term). For example, a person of ordinary skill in the art would readily recognize that Figures 3, 4, and 5 of Reddy all depict perspective views of a scene from a defined viewpoint (with x, y, and z coordinates) in a particular direction. Paragraph 37 of Reddy further explains how a user can use a map display (shown in a separate window from the perspective view, to move directly to a particular location). For example, in Fig. 5 of Reddy, the perspective view in the center of the image corresponds to the map shown on the left. The green square in the map shows the area of interest, and the blue wedge shows the view direction (east-northeast in the example, which is rotational position data) from the viewpoint:



258. Additionally, the scene shown in Fig. 5 of Reddy also requires an altitude (the "z" coordinate under the most commonly used convention, although the naming of the axes is arbitrary) in order to create a three-dimensional perspective. Reddy also teaches that TerraVision "supports 6-degrees-of-freedom input devices," which a person of ordinary skill would understand to control pitch, roll, and yaw (*i.e.*, rotational positional data) in addition to translation in the three standard directions (x, y, and z). Ex. 1004 at ¶ 38;

259. Woods also teaches a three-dimensional viewpoint including three-dimensional coordinates and rotation data. Woods discloses that users can "move and tilt" the camera / viewpoint "in any direction." Ex. 1003, 2:39-43; *see also id.*,

3:41-42 ("Users are free to move in any direction, change viewing angles, and movement speeds."). "Viewing angles" (*i.e.* the direction that the camera is looking) are another way of describing the rotational data for a camera. Woods also discloses that VRML uses a three-dimensional coordinate system and that the default camera position "is located at (0,0,10) and looks along the negative z-axis." *Id.*, 6:13-15. In my opinion, a person of ordinary skill in the art would understand "(0,0,10)" to be three-dimensional positional coordinate data. Using the conventional (x,y,z) coordinate notation, the coordinates (0,0,10) would denote a position at the origin of the x and y axes (*i.e.* the X and Y axes are 0) and 10 units along the Z axis, while "look[ing] along the negative z-axis" means that the camera is looking from this position toward the origin (0,0,0) of the coordinate system. In layman's terms, this describes a camera above the x, y plane looking down. The view of the camera (looking down) is rotational positional data. Woods also discloses that prioritization may take into account "camera position, orientation, speed and direction." *Id.*, 9:61-63, Abstract (referencing "position, orientation, and velocity of the camera"). In my opinion, these disclosures are analogous to the '645 specification's high level description of navigation controls. Ex. 1001, 5:65-6:4, 7:67-8:4.

11. Dependent Claims 7, 19, and 31 Are Obvious

Claim 7: The method of claim 5, wherein [a] the client software configures the local store as a server to provide access to at least some image parcels received by the wireless portable device, [b] the at least some image parcels comprising the first image parcel and the second image parcel.

Claim 19: The computing system of claim 17, wherein [a] the client software configures the local store as a server to provide access to the at least some image parcels received by the wireless portable device, [b] the at least some image parcels comprising the first image parcel and the second image parcel received by the wireless portable device.

Claim 31: The method of claim 28, further comprising [c] receiving client software by the wireless portable device from the one or more servers, wherein [a] the step of storing comprises storing at least some image parcels received by the wireless portable device in the local store configured by the client software as a server to provide access to the at least some image parcels, [b] the at least some image parcels comprising the first image parcel and the second image parcel received by the wireless portable device.

260. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims and claim 31 including an additional limitation (sub-element [c]). Therefore, I will discuss these claims together.

261. I discussed above in Section IX.C the proper construction of the "configures ... as a server to provide access to at least some image parcels received by the wireless portable device" language in sub-element [a]. Reddy and Woods disclose this limitation by disclosing client-side caches that would store downloaded resources such as map tiles. Ex. 1004 at ¶¶ 21, 45; Ex. 1003 at 3:56-60, 14:18-27. The client-side caches disclosed in Reddy and Woods are just like

local parcel data store 46 in the '645 Patent such that a POSITA would understand them to be "configure[d] . . . as a server to provide access to at least some image parcels received by the wireless portable device" because they can satisfy the application's needs for certain image parcels. As I noted in Section IX.C, there is no discussion in the '645 Patent of the local parcel store being configured as a server in any other sense.

262. Regarding sub-element [b], a POSITA would understand that once received, the first and second image parcels would be cached. For example, Reddy discloses that the cache includes data "for regions that the user has already browsed. Ex. 1004, ¶ 45. Woods discloses that the same priority determinations used for downloading can be used to determine whether and where the asset is cached. Ex. 1003 at 14:18-61. Woods thus indicates that resources with sufficient priority to be downloaded (*e.g.*, the claimed first and second image parcels) would also be cached.

263. Sub-element [c] of Claim 31 is substantially similar to sub-element [a] of Claim 2 and is obvious for the same reasons discussed above for that sub-element.

12. Dependent Claims 9, 21, and 33 Are Obvious

Claim 9: The method of claim 1, further comprising sending overlay data by the one or more servers to the wireless portable device over the network communication channel.

Claim 21: The computing system of claim 13, wherein the one or more servers are further configured to send overlay data to the wireless portable device over the network communication channel.

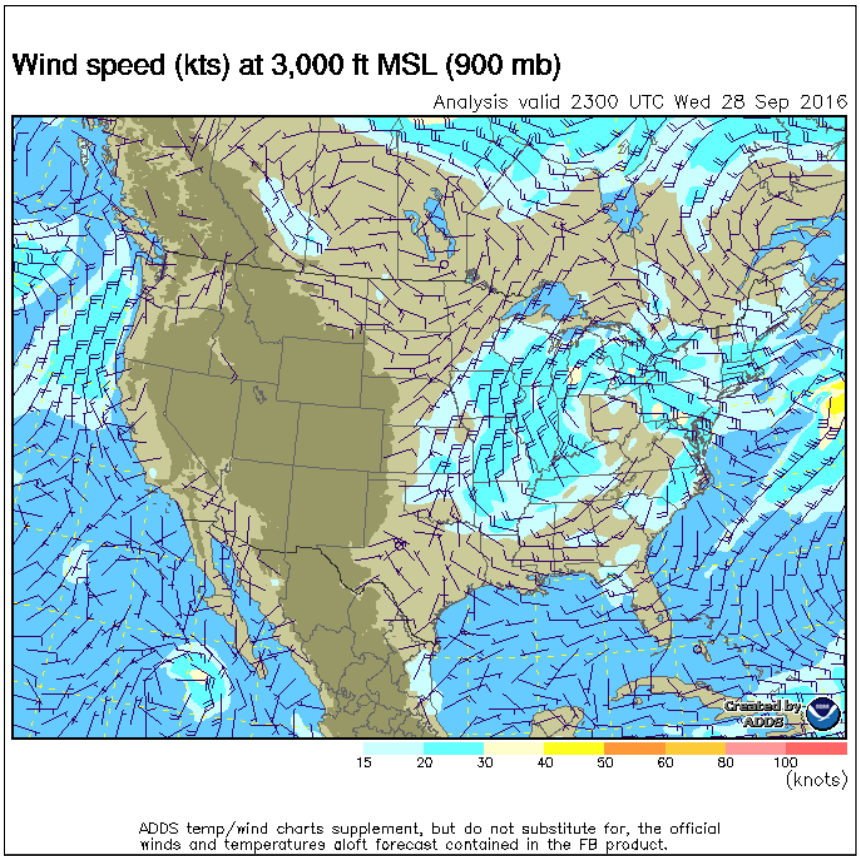
Claim 33: The method of claim 25, further comprising receiving by the wireless portable overlay data sent by the one or more servers to the wireless portable device over the network communication channel.

264. In my opinion, these claims are similar to each other, other than depending from different independent claims. Therefore, I will discuss these claims together.

265. In my opinion, Reddy teaches this claim. Reddy teaches that terrain tile files are linked to "feature files," which may contain information such as cultural features, roads, and terrain or other annotations (Ex. 1004, ¶¶ 22-26), while example in the introduction describes a user viewing 3D buildings and information about the buildings (*id.*, ¶ 3). The feature files are retrieved by clients just like the other VRML data. Ex. 1004, ¶¶ 3, 9. The information in the feature files may be overlaid on the image in order to show the information in its correct position on the map. For example, Fig. 4 shows roads and buildings in Ft. Benning overlaid on the aerial view:

266. In my opinion, all of these features in the "feature files" of Reddy satisfy the claimed "overlay data" in this claim. It is also my opinion that a person of ordinary skill in the art would recognize that many of the types of data discussed in Reddy, particularly in ¶ 25, would typically and preferably be displayed as overlay data. For example, the data contained in feature files may include features such as weather data, *e.g.* wind vectors, and the system may be used in military mission planning (*id.*, ¶ 48). A "wind vector" is a graphical icon that shows the direction and speed of the wind in a particular direction. For example, the figure below from an FAA aviation weather report shows the wind vector graphical icons:[12]

[12] Although this illustrative example is recent, a person of ordinary skill in the art would recognize that similar figures have been used in official weather reports for decades.



267. In my opinion, it would be obvious to a person of ordinary skill in the art that the most likely use of such information is to overlay it on a map in order to provide the most relevant information at a user, as shown in the example above.

268. In my opinion, Reddy's teaching of military mission planning uses would also lead a person of ordinary skill in the art to incorporate military operational graphics, which are graphical icons widely used in military planning and operations to depict military units in a concise, standardized manner. For

example, the symbol below depicts an armored cavalry or reconnaissance battalion:[13]

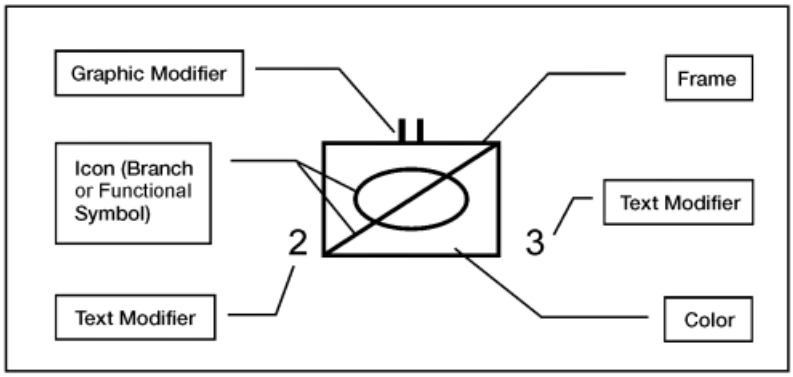
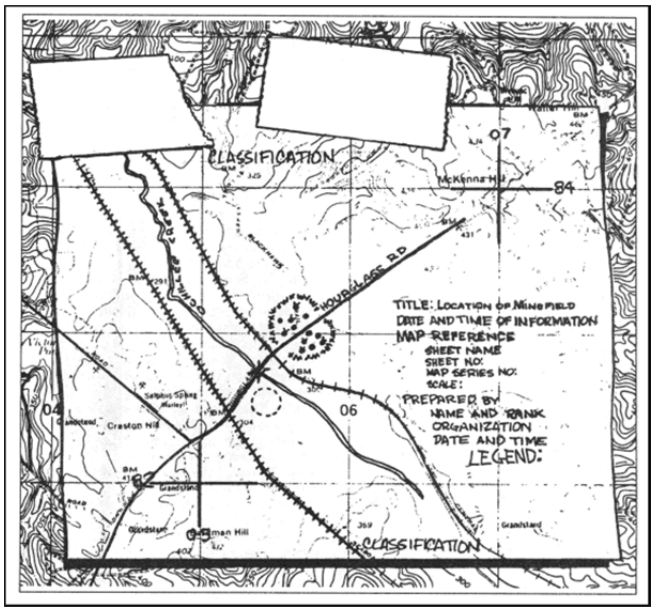


Figure 5-1. Unit Symbol Components

269. In my opinion, and based on my experience working on military and defense-related projects, the ubiquitous use of such operational symbols in military maps would provide a strong motivation for a person of ordinary skill in the art to utilize such graphical icons in the system of Reddy, based on Reddy's suggestion

[13] *See* Army Field Manual (FM) 1-02/Marine Corps Reference Publication (MCRP) 5-12A, available online at http://www1.udel.edu/armyrotc/current_cadets/cadet_resources/manuals_regulations_files/FM%201-02%20-%20Operational%20Terms%20&%20Graphics.pdf (accessed September 28, 2016). Although this is the current version of the manual, in my experience working on defense contracts the concept of using operation icons to depict military units would have been well-known by 1999 or 2000.

of using the system for military mission planning. Further, one of the most common uses of such icons is to display them on a map as an overlay in order to conveniently depict their locations. For example, even before the modern computer era it was common military practice to display such information on literal "overlays," which were sheets of transparent material (such as acetate) marked with reference lines in order to enable the overlay to be lined up with an underlying map so that geographically referenced information can be shown on its correct place without marking the underlying map:[14]



270. In the modern computer era, including prior to the effective filing date of the '645 Patent, the same function may be accomplished electronically by providing annotations such as geographically-referenced operational symbols as an

[14] *See, e.g.* Army Field Manual (FM) 3-25.26, Chapter 7 ("Overlays").

electronic "overlay" so that the data can be displayed on top of a base map. This is precisely what Reddy describes in regard to "feature files." Therefore, in my opinion, it would be obvious to a person of ordinary skill in the art that Reddy teaches overlay data.

13. Dependent Claims 10, 22, and 34 Are Obvious

Claim 10: The method according to claim 9, wherein the overlay data comprises text annotations relating to at least one item selected from the group consisting of: one or more street names, one or more building names, and one or more landmarks.

Claim 22: The computing system according to claim 21, wherein the overlay data comprises text annotations relating to at least one item selected from the group consisting of: one or more street names, one or more building names, and one or more landmarks.

Claim 34: The method according to claim 33, wherein the overlay data comprises text annotations relating to at least one item selected from the group consisting of: one or more street names, one or more building names, and one or more landmarks.

271. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims. Therefore, I will discuss these claims together.

272. Reddy teaches that the feature files include information such as annotations, Ex. 1004, ¶¶ 6, 22, 25-26, and that the user in the example case in the introduction can access annotations about a target building, *id.*, ¶ 3. In my opinion, it would be obvious to a person of ordinary skill in the art that since (1) a purpose of Reddy is to visualize and understand geographic information and (2) the system

supports annotations, particularly through its use of "feature files," text annotations such as street or building names and landmarks would be a likely use for the system in order to provide usable information to a user in addition to visualizing the terrain. For example, a user who wanted to see roads displayed on a map (*See, e.g.* Fig. 4 of Reddy) would naturally want to see the names of roads, and it would be reasonable to expect that the information about the target building discussed in the scenario in ¶ 3 would include the name of the building, which satisfies the claim element. Woods expressly describes navigating among the streets in a city, and it similarly would have been obvious to display street name annotations to help orient the user.

14. Dependent Claims 11, 23, and 35 Are Obvious

Claim 11: The method of claim 1, wherein the wireless portable device issues the first request and the second request according to a priority order based at least in part on viewable areas corresponding to the first user-controlled image viewpoint.

Claim 23: The computing system of claim 13, wherein the wireless portable device issues the first request and the second request according to a priority order based at least in part on viewable areas corresponding to the first user-controlled image viewpoint.

Claim 35: The method of claim 25, wherein the steps of sending the first request and sending the second request are performed according to a priority order based at least in part on viewable areas corresponding to the first user-controlled image viewpoint.

273. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims. Therefore, I will discuss these claims together.

274. In my opinion, Reddy and Woods teach this element. The term "viewable areas" is not used in the specification of the '645 Patent or discussed in the file history, and in my opinion this term does not require construction. I do note that the term "viewable" is used several times in the specification to simply refer to tiles that are viewable. *See, e.g.* Ex. 1001, Fig. 5 item 82 ("determine viewable parcels for update"), 9:11-13 (" in order to make optimal use of the available memory, only currently *viewable* image parcels are subject to download"); 10:52-56 (" the argument P vertices sent to S represent the position of the vertices composing each of the polygons, after being clipping [*sic*] to the viewing frustum, *viewable* within the display space having the fixed resolution [xRes, yRes]"); 11:2-6 ("thus, the accumulated priority for any image parcel pending download is the sum of the values returned by the function S for each of the viewable polygons that require some part of the image parcel as the source data for texture map rendering of the polygon"). Therefore, in my opinion, the term "viewable areas" includes at least prioritizing areas that are viewable over areas that are not.

275. Woods describes a priority scheme to ensure "the most important data is fetched first (*i.e.*, that data which is most likely perceived)." Ex. 1003 at 4:56-61. Woods suggests it is wasteful and inefficient to prioritize the downloading of VRML resources that would not be visible to the user. *Id.* at 7:37-47. Woods further describes prioritizing VRML assets based on the user-controlled camera position and direction. In Fig. 5A, for example, the grid region in front of the camera (region 3,4) is designated priority 1, surrounding grid regions are designated priority 2, and still further grid regions are designated priority 3. *Id.* at 9:40-10:64. Woods describes a similar scheme that prioritizes behavior assets that are within the field of view. *Id.* at 11:18-36; Fig. 5C.

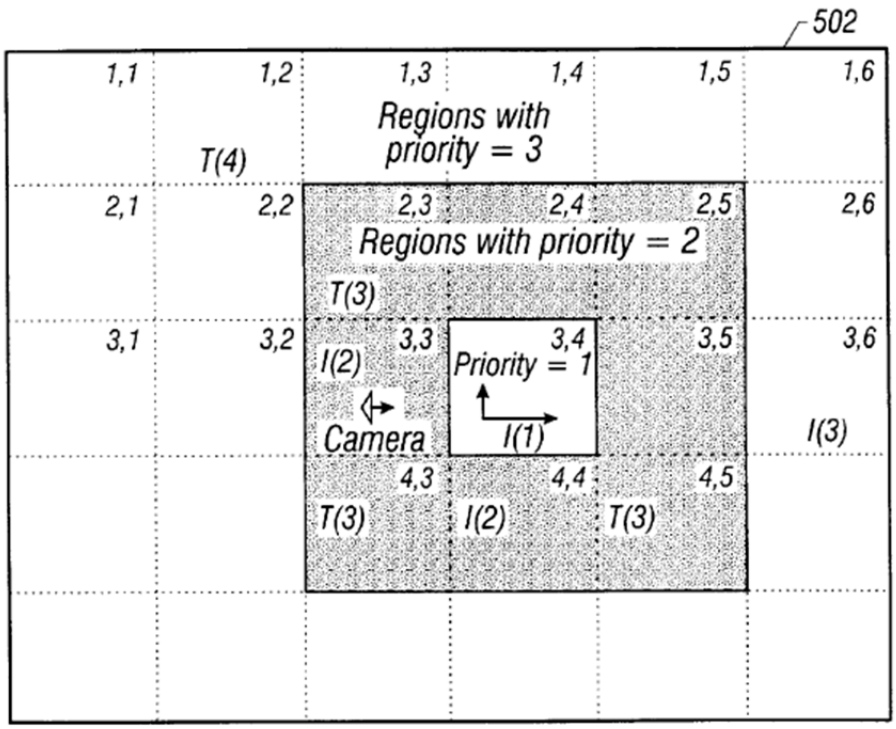


FIG. 5A

276. Thus, in the Reddy–Woods combination, the tiles in front of the camera close to the gaze or focus point (that is, those in the area that is viewable from the camera) be prioritized over those that are further away (*e.g.*, grid region 3,5 in Fig. 5A) or off-screen entirely (*e.g.*, grid region 3,2 in Fig. 5A, which is behind the camera).

277. A person of ordinary skill in the art would also understand that VRML assets closer to the camera would generally take up more screen space than assets further away. Indeed, this is why lower-resolution imagery is adequate for distant areas but not for areas near the gaze point. *See* Ex. 1004, ¶¶ 16-17. Because of the correlation between distance and screen size, prioritization by distance as disclosed by Woods also prioritizes assets "at least in part" based on viewable areas.

B. GROUND 2: CLAIMS 8, 20, AND 32 ARE UNPATENTABLE UNDER 35 U.S.C. § 103(a) AS BEING OBVIOUS OVER REDDY IN VIEW OF WOODS AND CHIARABINI

1. The Reddy-Woods-Chiarabini Combination

278. In my opinion, Claims 8, 20, and 32 are unpatentable over Reddy in view of Woods and further in view of Chiarabini. Chiarabini is a Hewlett-Packard patent that discloses "methods and systems that enable faster and more reliable downloading of data received from an external content source." Ex. 1006 at 1:19-23. Chiarabini describes a system in which a client downloads large images from a server over the Internet by downloading segments of the image in parallel. *Id.* at

2:43-51, 3:27-43. Chiarabini further teaches that the number of parallel download threads can be adjusted in order to download the image data more efficiently. *Id.* at 3:27-43, 9:46-10:15.

2. Motivations to Combine

279. A person of ordinary skill in the art would be motivated to combine the teachings of Chiarabini with those of Reddy and Woods. Similar to Reddy and Woods, Chiarabini describes downloading segments of image data corresponding to "a big image file." Ex. 1006 at 8:63-67. Like Reddy and Woods, Chiarabini describes client-side browsers that download image data from servers over the Internet. *Id.* at 5:59-66. Similar to Woods, Chiarabini describes downloading multiple image segments in parallel for improved downloading efficiency. *Id.* at 3:27-43. Additionally, Reddy teaches the use of multi-threaded browsing. Ex. 1004 at ¶ 41. A person of ordinary skill would have easily recognized that Chiarabini's improved downloading technique was applicable to the type of image downloading described in Reddy and Woods. Indeed, Reddy's image tiles are segments of an original image just like the image segments described in Chiarabini.

280. Woods provides express motivation to combine in noting that the number of active and waiting fetches in a given implementation depends on various parameters that will be apparent to persons of ordinary skill in the art. Ex.

1003 at 12:67-13:4. Chiarabini provides specific and complementary teachings regarding how the number of parallel download requests can be optimized to more efficiently use available network bandwidth.

281. While Chiarabini discusses using its parallel downloading scheme in a system that downloads images for printing, it discloses that, alternatively, the downloaded image data could be displayed or simply stored for later use. Ex. 1006 at 6:21-26. A person of ordinary skill in the art would appreciate that Chiarabini's teachings are not limited to printing-related systems but are also applicable to image display applications like those described in Reddy and Woods. Indeed, Chiarabini discloses on HTTP GET commands. *Id.* at 9:20-26. A person of ordinary skill in the art would understand that VRML applications like most Internet-based applications use HTTP commands to download resources. *See* Ex. 1003 at 2:34-47 (VRML browsers operate like web browsers).

3. Dependent Claims 8, 20, and 32 are Obvious

Claim 8: The method of claim 1, wherein [a] the second image parcel is selected based on the first user-controlled image viewpoint, and [b] number of parallel requests by the wireless portable device for image parcels of the series is determined based at least in part on network response latency and available system resources, thereby enabling efficient use of network bandwidth in conditions of network latency.

Claim 20: The computing system of claim 13, wherein number of parallel requests by the wireless portable device for image parcels of the series is determined based at least in part on network response latency and available

system resources, thereby enabling efficient use of network bandwidth in conditions of network latency.

Claim 32: The method of claim 25, further comprising determining number of parallel requests by the wireless portable device for image parcels of the series based at least in part on network response latency and available system resources, to enable efficient use of network bandwidth in conditions of network latency.

282. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims and Claim 8 including an additional limitation (sub-element [a]). Therefore, I will discuss these claims together.

283. Sub-element [a] of Claim 8 specifies that the second image parcel is selected based on the *first* user-controlled image viewpoint (rather than a second user-controlled image viewpoint). As I discussed above in addressing the independent claims, Reddy discloses that clients request multiple parcels for a given viewpoint such that there would be a second image parcel selected based on a first viewpoint. Reddy's Fig. 1(b) shows that a client may request higher resolution tiles for areas close to the user's location and lower resolution parcels for more distant imagery when the viewpoint is in the lower-right corner. *See also* Ex. 1004 at ¶ 17. Similarly, Woods discloses that clients may download multiple VRML resources that are selected and prioritized based on a user-controlled viewpoint. Ex. 1003 at 7:66-8:8.

284. Sub-element [b] of Claim 8 and the only element of Claims 20 and 32 state that the number of parallel requests for image parcels is determined based at least in part on network response latency and available system resources such that network bandwidth is used efficiently when there is latency.

285. Chiarabini discloses determining the number of parallel image requests based at least in part on network latency. Chiarabini describes adjusting the number of parallel requests for imagery data based on network performance. Ex. 1006 at 3:33-43. Chiarabini notes that this approach "optimi[zes] the capacity of the line" (*id.*), which a person of ordinary skill in the art would understand as meaning it makes efficient use of available bandwidth to more quickly download the image data. Chiarabini discloses checking the download speed of the executing download threads. *Id.* at 9:9:46-51. If any of the active download threads is achieving a transfer speed at or above a predetermined threshold (*e.g.*, 20 Kb/s), this indicates that the connection is not yet "saturated," and an additional download thread is spawned. *Id.* at 9:51-53.

286. However, Chiarabini discloses that if none of the executing download threads is achieving good speed, no additional download thread will be spawned until one of the executing threads achieves good speed or no other download thread is executing. *Id.* at 9:46-51. This would result in a decrease of the number of parallel downloads if a network began to exhibit increased latency. To illustrate,

imagine five threads are downloading but none at a rate greater than 10 Kb/s. When one of the five download threads finishes its download, the number of concurrent threads is reduced to four. A (new) fifth thread will only be spawned if the speed of one of the four executing threads reaches the threshold (*e.g.*, 20 Kb/s).[15] The same would occur when the next thread completes leaving only three executing threads and so on. The only time a new thread would be spawned notwithstanding poor network performance (or high latency) would be when there are no threads running. *Id.* at 9:46-51. Therefore, Chiarabini's approach adjusts the number of threads up or down based on network performance including in response to conditions of network latency.

287. A person of ordinary skill would have understood the relationship between latency and download speed. For example, Killelea describes "latency" as "the time between making a request and beginning to see a result." Ex. 1044 at 43. To determine the number of parallel download requests based on download speed as disclosed in Chiarabini is to do so "based at least in part on network response latency" as claimed. Chiarabini also discloses that a client can determine transfer speed by pinging the server. Ex. 1003 at 8:16-19, Fig. 4 (steps 240, 242). A person of ordinary skill would have been familiar with using the ping utility to test

[15] Chiarabini discloses using on-demand threads as opposed to a thread pool.

network latency. Killelea discusses using ping to measure the latency between a first computer and a remote computer over a network. Ex. 1044 at 45-46.

288. Chiarabini also discloses capping the number of threads that can be spawned "so that the line is not overloaded." Ex. 1003 at 9:54-60. Chiarabini suggests a maximum of eight for a 128 Kb/s connection and a maximum of four for slower connections. *Id.* at 9:53-60. This limit also promotes efficient use of network bandwidth by preventing additional threads from simply stealing bandwidth from the already executing threads without improving overall download speed.

289. It also would have been obvious to a POSITA that the number of parallel image parcel requests be determined based at least in part on available system resources. Woods discloses a multi-threaded implementation with a "fetching process thread" that is separate from the browser's main "runtime thread." Ex. 1003 at 12:56-59. Woods describes an example that allows for four parallel, "active fetches." Ex. 1003 at 12:56-67. (This is similar to the '645 Patent's suggestion that four download threads are used in the preferred embodiments. *See* Ex. 1001 at 8:62-64.)

290. Woods also points out that "the number of active fetches and waiting fetches used in a specific implementation of the present invention *depends on various hardware and software parameters*, and will be apparent to those skilled in

the relevant art(s)." Ex. 1003 at 12:67-13:4 (emphasis added).[16] A person of ordinary skill would have understood Woods to be alluding to the obvious relationship between multithreading and system resources (e.g., operating system support, number and speed of the processors, memory). It was well understood that more powerful computers could support a larger number of threads. For example, *Programming Microsoft Visual Basic 6.0* by Francesco Balena (1999) notes that "you can increase the size of the thread pool when you deploy your application on a more powerful system." Ex. 1064 at 877.

291. While many operating systems did support multithreading by the late 1990s, not all operating systems did. For example, *Operating Systems Concepts* by Abraham Silberschatz and Peter Baeer Galvin (1998) noted that "new operating systems" were providing thread support. Ex. 1055 at 192. (In my opinion, Silberschatz is representative of textbooks undergraduate students would have used

[16] While Bradium may argue that this disclosure is vague, I would note that the '645 Patent itself says nothing about which "available system resources" should impact the number of parallel requests or how or about how the "network response latency" should impact the number of parallel requests. Thus, the '645 Patent suggests that persons of skill in the art would have understood how to determine the appropriate number of parallel requests.

in operating systems courses in the late 1990s.) The Palm operating system, for example, did not support multithreaded applications as discussed in *Palm Database Programming* by Eric Giguere (1999). Ex. 1059 at 15-16. As a result, a person of ordinary skill would have known it would be more challenging to try to implement parallel download requests in an application for a Palm device. *See, e.g., id.* at 15-16, 102-105 (describing techniques for breaking up long operations into small chunks "to avoid the appearance of hanging the device"). By comparison, it would be easier to implement parallel download requests in a Windows CE application because Windows CE supported multithreaded applications as discussed for example in *Programming Microsoft Windows CE* by Douglas Boling (1998). Ex. 1058 at 493-507 (comparing multithreading in Windows CE with Windows NT and Windows 98).

292. It was also well known that computers with multiple processors could support a greater number of threads than single processor devices. *Programming with Threads* by Steve Kleiman et al. (1996), provides exemplary guidance on this point:

> You may wish to experiment with different strategies for choosing the right number of threads; your experiments ***need to consider*** both ***the number of processors available*** and the amount of work required for reach thread (sometimes called the grain size). ***Allocating a***

few more threads than the number of processors is a good starting point.

Ex. 1057 at 285 (emphasis added). Kleiman's book was known in the art as a practical guide to multithreaded programming. A person of ordinary skill in the art would have known that single processor computers must perform context switches to execute instructions associated with different threads. *See, e.g.,* Ex. 1058 (Boling) at 493. Furthermore, there is overhead associated with the context switches that must occur for a processor to switch between threads. *Win32 Multithreaded Programming* by Aaron Cohen and Mike Woodring (1998) is an O'Reilly reference book discusses this in its introductory chapter. Ex. 1056 at 5-6. Because CPU cycles are lost during context switches, a faster processor would better be able to handle the increased number of context switches for a larger number of threads. *See, e.g., id.* at 335-336 (stressing importance of capping the number of threads to avoid overwhelming the system). Finally, because threads also require memory (*e.g.*, a stack and program counter) it was also well understood that the number of threads is constrained by available memory. *See, e.g.,* Ex. 1055 (Silberschatz) at 103; Ex. 1058 (Boling) at 499.

293. Sub-element [b] is also inherently disclosed because both Woods and Chiarabini disclose using four parallel download requests which is the same number used in the preferred (and only) embodiments described in the '645 Patent.

Ex. 1003 at 12:65-67; Ex. 1006 at 9:53-60. The '645 Patent suggests that four parallel requests strikes the right "balance between the available system resources and the network response latency, given the available bandwidth of the network connection." Ex. 1001 at 8:62-67.

294. Finally, Woods also discloses that a particular system resource—cache space—can impact the number of parallel requests. A simple example illustrates this. Assume that the next four VRML resources to be downloaded, call them resources A-D, have the following priorities:

Resource	Priority
A	5
B	6
C	7
D	8

Woods describes an exemplary client that supports four simultaneous fetches. Ex. 1003 at 12:65-67. Woods also discloses that it may be desirable to avoid downloading resources with priority 7 or lower if the cache is already full. *Id.* at 14:45-47. In the above example, therefore, if the cache is full, only two assets, A and B, would be downloaded, and they would be downloaded in parallel. However, if the cache is not full, all four assets would be downloaded in parallel.

C. GROUND 3: CLAIMS 12, 24, AND 36 ARE UNPATENTABLE UNDER 35 U.S.C. § 103(a) AS BEING OBVIOUS OVER REDDY IN VIEW OF WOODS AND FULLER

1. The Reddy-Woods-Fuller Combination

295. In my opinion, Claims 12, 24, and 36 are unpatentable over Reddy in view of Woods and further in view of Fuller. Fuller, which was published approximately three years before Reddy, describes an earlier version of the TerraVision application used with the MAGIC ("Multidimensional Applications and Gigabit Internetwork Consortium") project. Ex. 1011 at 15. Like Reddy, Fuller teaches how a user may "view and navigate through (*i.e.* 'fly over') a representation of a landscape created from aerial or satellite imagery." *Id.* at 17. Such imagery is processed into a series of "tiles" at resolutions varying by factors of two at each level, so that "[l]ow-resolution tiles are required for terrain that is distant from the viewpoint, whereas high-resolution tiles are required for close-in terrain." *Id.* at 17, Fig. 3:

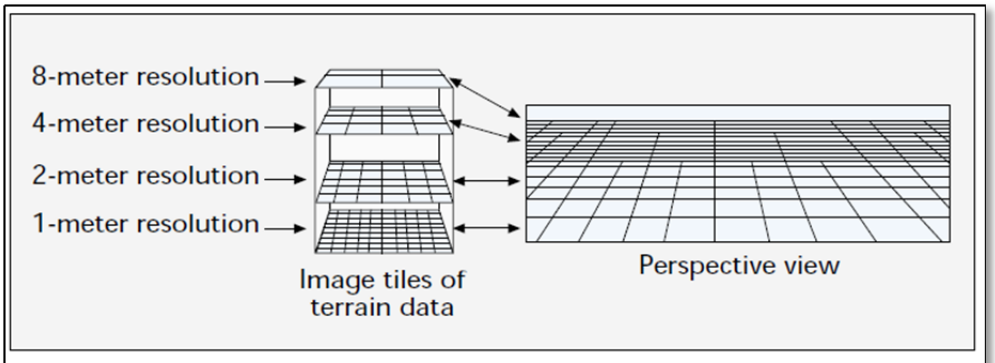


Figure 3. *Relationship between tile resolutions and perspective view. (Source: SRI International)*

Cf. Ex. 1004, Fig. 1.

296. Fuller further teaches prioritizing tile retrieval in order to address problems caused by network congestion resulting in late delivery of the tiles. Requests to the server for tiles "assign[] one of three levels of priority to each tile requested," so that coarser tiles are likely to be retrieved first. Ex. 1011 at 19. Among the reasons for which coarser tiles are downloaded first are that "the rendering algorithm needs the coarse tiles before it needs the next-higher-resolution tiles" and that "there are fewer tiles at the coarser resolutions, so it is less likely that they will be delayed." *Id.* Therefore, the priority order of first and second tiles (*e.g.*, a higher-resolution tile and a lower-resolution tile) is issued based on the resolution of the two tiles as claimed.

297. Fuller further teaches that planned future work included delivering data to "end users with a range of communications speeds, link qualities, computational powers, and display capabilities" as well as providing data to mobile users. *Id.* at 19.

2. Motivations to Combine Reddy, Woods, and Fuller

298. In my opinion, Fuller is analogous art to Reddy and Woods such that a person of ordinary skill in the art addressing issues relating to retrieving data to display imagery in three dimensions would naturally look to Fuller. Like Reddy and Woods, Fuller refers to retrieving data over a network to display in three

dimensions. In fact, Fuller describes an earlier embodiment of the TerraVision projects described in Reddy, although in my opinion a person of ordinary skill in the art would recognize that the teachings of Fuller are not limited to these specific preferred embodiments described in Fuller.

299. In my opinion, Reddy and Woods teach a common goal of optimizing bandwidth usage by prioritizing retrieval of the most needed elements of a three-dimensional scene, including geographic imagery. For example, Reddy teaches that it is desirable to load tiles in a "coarse-to-fine" manner (Ex. 1004, ¶¶ 21, 44). Fuller teaches that assigning higher priority to lower resolution tiles is one means of accomplishing this goal. Ex. 1011 at 19. In my opinion, a person of ordinary skill in the art would recognize that assigning higher priorities to lower-and resolution tiles is compatible with the prioritized fetching scheme of Woods. Indeed, because Woods teaches that a variety of factors they be incorporated into a priority calculation, prioritizing lower resolution tiles as taught by Fuller is simply adding another variable into the existing algorithm. Incorporating priority based on resolution as taught by Fuller into the efficient data request of Woods would achieve the goal (prioritizing lower resolution tiles) taught by Reddy and Fuller with predictable results.

3. Dependent Claims 12, 24, and 36 Are Obvious

Claim 12: The method of claim 1, wherein the wireless portable device issues the first request and the second request according to a priority order based at least in part on resolutions of the first image parcel and the second image parcel.

Claim 24: The computing system of claim 13, wherein the wireless portable device issues the first request and the second request according to a priority order based at least in part on resolutions of the first image parcel and the second image parcel.

Claim 36: The method of claim 25, wherein the steps of sending the first request and sending the second request are performed according to a priority order based at least in part on resolutions of the first image parcel and the second image parcel.

300. In my opinion, these claims are substantially similar to each other, other than depending from different independent claims. Therefore, I will discuss these claims together.

301. Reddy, Woods, and Fuller render these limitations obvious. As discussed, Fuller describes prioritizing the download of map tiles based at least in part on resolution differences. Fuller teaches that requests to the server for tiles "assign[] one of three levels of priority to each tile requested ... with the coarsest assigned the highest priority within the set" Ex. 1011 at 19. Coarser tiles are prioritized because (1) they are needed for rendering before the higher resolution tiles and (2) there are fewer of them that correspond to the display area. *Id.* Accordingly, Fuller teaches that the priority order of retrieval for tiles

corresponding to a particular viewpoint is based at least in part on their respective resolutions.

XI. CONCLUSION

302. It is my opinion that the Challenged Claims of the ’645 Patent are invalid as obvious.

303. This is the END of my declaration.

Curriculum Vitae for
William R. Michalson

Research Associates, LLC
26 West Main Street, STE 2
Dudley, MA 01571
Email: wrm@wmichalson.com
Tel: (508) 461-6242
Cell: (508) 331-4134

1. Personal:

1.1 Education

Ph.D. in Electrical Engineering, 1989, Worcester Polytechnic Institute, Worcester, Massachusetts.

Dissertation: *A Parallel Computer Architecture for Real-Time Decision Making*. The dissertation develops a hierarchical, multiple processor, computer architecture for executing artificial intelligence programs in real-time. Dissertation Directors: Dr. Peter E. Green and Dr. R. James Duckworth.

Minor Areas: Minor sequences completed in Mathematics and Physics.

Specialties: Area examinations passed in the fields of Computer Architecture, Probabilistic Systems Analysis, and State Space Analysis.

M.S. in Electrical Engineering, 1985, Worcester Polytechnic Institute, Worcester, Massachusetts.

Specialties: The courses taken stressed Computer Architecture, Communications Systems, and Solid-State Physics.

B.S. in Electrical Engineering, 1981, Syracuse University, Syracuse, New York.

1.2 Work experiences - Academic.

1991-Present **Worcester Polytechnic Institute**
Professor of Electrical and Computer Engineering; Professor of Computer Science.

Effective July 1, 2005	Promoted to the rank of Full Professor (Professor of Electrical and Professor of Computer Science)
November 17, 2004	Appointed dual professorship, adding the title of Associate Professor of Computer Science.
July 1, 1998	Granted tenure and promoted to the rank of Associate Professor.
August 1, 1992	Assistant Professor of Electrical Engineering (tenure-track).
August 1, 1991	Visiting Assistant Professor of Electrical Engineering.
January 1, 1990	Adjunct Assistant Professor of Electrical Engineering.

1.3 Work experiences other than teaching (chronological).

2012-2014 **Grid Roots, LLC**

Grid Roots, LLC is a company which was formed in 2012 for the purpose of commercializing a navigation and tracking device for use by children and the elderly to allow caregivers to non-intrusively monitor their activities. The system under development integrates GPS, inertial and beacon-based navigation technologies to develop a system for users to track deployed devices. My responsibilities within Grid Roots, LLC relate to hardware and software engineering, as well as the development of IP related to tracking individuals.

1995-Present **Research Associates, LLC**

Research Associates, LLC is a company I formed in which I perform engineering and consulting in the areas of computer systems, communications and navigation. All of my litigation-related and other consulting activities are performed through Research Associates, LLC.

1988-1991 **Raytheon Company**

Subsequent to receiving my Ph. D., I returned to the Equipment Division of the Raytheon Company. Shortly after I returned, I was promoted to a title of Engineer, Design and Development which was the highest title I could hold based on my level of education and years of experience. Within a year, I was selected to sit on the engineering staff of a newly formed System Engineering Department of the Division's Computer and Displays Laboratory. In this department I acted primarily as a consultant to other departments within the laboratory. My responsibilities ranged from leading the hardware/software development of supercomputer-class computer systems to performing applied research into the exploitation of new technology. My role was similar to that of a Principle Investigator in an academic setting as I was responsible for securing funding and personnel, leading research efforts, interacting with the research sponsor, and reporting results. At the time of my departure I was involved with the following projects:

Fault-Tolerant Multiprocessor

The development of a highly fault tolerant, highly reliable, real-time computer system intended for long-duration spaceborne applications. This system is designed to produce in excess of one gigaoperation per second of raw processing power.

Optimal Task Allocation

A program of applied research into the use of Genetic Algorithms for deriving optimal mappings of software tasks to the hardware processing elements in distributed systems.

Performance Modeling and Scaling

This project focused on the development of simulation models for characterizing the performance of a large scale multiple processor system. These models formed a basis for predicting system performance for several different hardware configurations to ensure compliance with system specifications.

High Clutter Signal Detection

A program of applied research into the use of Neural Networks to detect the presence of targets in extremely high clutter environments.

Power Efficient Computing

A program of applied research into an Integrated Optical computer structure that is designed to maximize the number of computations that can be performed per unit of power.

1985-1988 Raytheon Company (Leave of Absence)

In 1985 I became one of two people in the Equipment Division to receive Aldo Miccioli Fellowships. This Fellowship was awarded to allow me to pursue full-time study towards the Ph.D. degree. I returned to Raytheon during the summer of 1986, but otherwise remained on leave of absence to dedicate my time to my studies.

1982-1985 Raytheon Company

Engineer in the VLSI Design Department of the Computer and Displays Laboratory within Raytheon's Equipment Division. I was lead engineer for the design of several semi-custom VLSI circuits for both signal and data processing applications.

1981-1982 Raytheon Company

Engineer in the Cursive Displays Department of the Computer and Displays Laboratory. I designed and debugged circuit assemblies which were used in vector displays for air traffic control applications.

1.4 Consulting experiences.

1.4.1 Law-Related

Locata LBS LLC v. YellowPages.com LLC,
Retained by Baker Botts on behalf of defendant YellowPages.com. Case before the Central District of California (2:13-cv-07664). See also IPR2015-00151. Retained 9/14 to present.

M/A-COM Technology Solutions Holdings, Inc. v. Laird Technologies, Inc.
Retained by Erise IP on behalf of Laird Technologies, Inc., for invalidity consulting regarding U.S. Patent No. 6,272,349. Retained 6/14 to present.

Certusview Technologies, LLC v. S&N Locating Services, LLC and S&N Communications, Inc.,
Retained by Baker &McKenzie on behalf of defendant S&N. Patents-in-suit are U.S. Patents 8,265,344, 8,290,204, 8,340,359, 8,407,001, and 8,532,341. Case before the Eastern District of Virginia, (2:13-cv-346). Deposed 11/8/14; Retained 6/14 to present.

adidas AG and adidas America, Inc. v. Under Armour, Inc. and MapMyFitness, Inc.
Retained by Kilpatrick Townsend on behalf of plaintiff adidas. Case before the District of Delaware, (1:14-cv-00130). Retained 5/14 to present.

GeoTag, Inc., v. AT&T Mobility LLC and AT&T Services, Inc.,
Retained by Baker Botts as an expert on behalf of defendant AT&T. Patent-in-suit is U.S. Patent 5,930,474. Case before the Northern District of Texas, Dallas Division, (3:13-cv-00169). Deposed 5/29/14; Retained 1/14 to 9/14. Matter settled.

Nokia Corp v. HTC Corp.
Retained by Quinn Emanuel as an expert on behalf of defendant HTC. Case being litigated in Germany. Patent number EP0766811B1. Retained 12/13 to 2/14. Matter settled.

Porto Technology, Co., Ltd. et al. v. Cellco Partnership d/b/a Verizon Wireless
Retained by Wiley-Rein as an expert on behalf of defendant Verizon. Case before the United States District Court for the Eastern District of Virginia (Case No. 3:13-cv-00265). Retained 10/13 to 2/14. Matter dismissed.

Nokia Corp v. HTC Corp.
Retained by McDermott Will and Emery and White & Case as an expert on behalf of defendant HTC. Case before the United States District Court for the District of Delaware (Case No. Case No. 1:12-cv-00550-UNA and Case No. 1:12-cv-551-UNA. Retained 6/13 to 2/14. Matter settled.

NXP B.V. v. Research In Motion, Ltd., et al.
Retained by Fish and Richardson as an expert on behalf of defendant Research In Motion. Patent-in-suit is U.S. Patent 6,501,420. Case before the United States District Court for the

Middle District of Florida (Case No. 6:12-cv-00498). Deposed 9/18/13; Testified in Court: 4/1/14 and 4/2/14; Retained 5/13 to 4/14.

Vehicle IP LLC v. Wal-Mart Stores Inc., et al.
Retained by Polsinelli-Shugart, as an expert on behalf of defendant Werner Enterprises. Patent-in-suit is U.S. Patent 5,694,322. Case before the United States District Court for the District of Delaware (Case No. 1:10-cv-00503). Deposed 7/15/13 and 9/20/13; Testified in Court: 9/27/13 and 9/30/13; Retained 4/13 to 9/13.

TracBeam, LLC v. Google Inc.
Retained by Quinn-Emanuel as an expert on behalf of defendant Google. Patents-in-suit are U.S. Patents 7,764,231 and 7,525,484. Case before the United States District Court for the Eastern District of Texas (6:11-cv-00093). Deposed 2/5/14; Retained 3/13 to 6/14.

Microsoft Corporation and Google Inc., v. GeoTag, Inc.
Retained by Perkins-Coie as an expert on behalf of plaintiff Microsoft Corporation. Patent-in-suit is U.S. Patents 5,930,474. Case before the United States District Court for the District of Delaware (1:11-cv-00175). Retained 1/13 to present.

GeoTag, Inc., v. Frontier Communications Corp., et al.,
Retained by multiple firms as an expert on behalf of defendants. Patent-in-suit is U.S. Patents 5,930,474. Case before the Eastern District of Texas, Marshall Division, (2:10-cv-00265; other defendants are listed in case numbers 2:10-cv-00265, 2:10-cv-00272, 2:10-cv-00437, 2:10-cv-00569, 2:10-cv-00570, 2:10-cv-00571, 2:10-cv-00572, 2:10-cv-00573, 2:10-cv-00574, 2:10-cv-00575, 2:10-cv-00587, 2:11-cv-00175, 2:11-cv-00404, 2:11-cv-00421, 2:11-cv-00424, 2:11-cv-00425, 2:11-cv-00570, 2:12-cv-00043, 2:12-cv-00051, 2:12-cv-00436, 2:12-cv-00438, 2:12-cv-00439, 2:12-cv-00441, 2:12-cv-00442, 2:12-cv-00444, 2:12-cv-00445, 2:12-cv-00446, 2:12-cv-00447, 2:12-cv-00448, 2:12-cv-00449, 2:12-cv-00450, 2:12-cv-00452, 2:12-cv-00454, 2:12-cv-00456, 2:12-cv-00459, 2:12-cv-00460, 2:12-cv-00462, 2:12-cv-00464, 2:12-cv-00466, 2:12-cv-00468, 2:12-cv-00469, 2:12-cv-00470, 2:12-cv-00471, 2:12-cv-00473, 2:12-cv-00474, 2:12-cv-00475, 2:12-cv-00476, 2:12-cv-00476, 2:12-cv-00477, 2:12-cv-00480, 2:12-cv-00481, 2:12-cv-00482, 2:12-cv-00482, 2:12-cv-00483, 2:12-cv-00486, 2:12-cv-00487, 2:12-cv-00520, 2:12-cv-00521, 2:12-cv-00523, 2:12-cv-00524, 2:12-cv-00525, 2:12-cv-00527, 2:12-cv-00528, 2:12-cv-00530, 2:12-cv-00532, 2:12-cv-00534, 2:12-cv-00535, 2:12-cv-00536, 2:12-cv-00537, 2:12-cv-00542, 2:12-cv-00543, 2:12-cv-00545, 2:12-cv-00547, 2:12-cv-00548, 2:12-cv-00549, 2:12-cv-00550, 2:12-cv-00551, 2:12-cv-00552, 2:12-cv-00555, 2:12-cv-00556, 2:12-cv-00570, 2:12-cv-00572, 2:12-cv-00573, 2:12-cv-00575, 2:12-cv-00587, 3:13-cv-00217). Deposed 5/29/14; Retained 1/13 to 9/14.

MOSAID Technologies Inc., v. Realtek Semiconductor Corporation
Retained by Sidley Austin, LLP as an expert on behalf of defendant Realtek Semiconductor Corporation. Patents-in-suit are U.S. Patents 5,131,006; 5,151,920; 5,422,887; 5,706,428; 6,563,786; and 6,992,972. Case before the United States District Court for the Eastern District of Texas (Tyler Division) (Case No. 2:11-cv-00179). Retained 12/12 to 12/12. Matter settled.

Hoyt A. Flemming v. Cobra Electronics Corporation

Retained by Sidley Austin, LLP as an expert on behalf of defendant Cobra Electronics Corporation. Patents-in-suit are U.S. Patents RE39038, RE40653 and RE41905. Case before the United States District Court for the District of Idaho (Case No. 1:12-cv-00392). Retained 11/12 to 06/13. Matter settled.

LBS Innovations LLC v. Aaron Bros., Inc., et al.
Retained as an expert on behalf of defendants Whole Foods Marketplace, Comerica, Hotels.com, Academy, Ltd., and Homestyle Dining. Patent-in-suit is U.S. Patent 6,091,956. Case before the Eastern District of Texas, Marshall Division, (Case No. 2:11-cv-00142-MHS-CMC. Deposed 10/5/12; Retained 7/12 to 12/12. Plaintiff moved to dismiss.

Advanced Media Networks, L.L.C. v. Gogo LLC et al.
Retained by Sidley Austin, LLP as an expert on behalf of defendant Gogo. Patent-in-suit is U.S. Patent 5,960,074. Case before the United States District Court for the Central District of California (Case No. 11-cv-10474). Deposed 2/6/13. Retained 7/12 to 8/13.

Walker Digital, LLC v. Google Inc.
Retained by O'Melveny & Meyers, LLP as an expert on behalf of defendant Google Inc. Patents-in-suit are U.S. Patents 6,199,014. Case before the United States District Court for the District of Delaware (Case No. 1:11-cv-00309-SLR). Deposed 2/27/13 - 2/28/13. Retained 6/12 to present (case stayed as of 8/13).

Silver State Intellectual Technologies, Inc. v. Garmin International, Inc., et al.
Retained by Erise IP, P.A. as an expert on behalf of defendants Garmin International, Inc. and Garmin USA, Inc. Patents-in-suit are U.S. Patents 6,525,768; 6,529,824; 6,542,812; 7,343,165; 7,522,992; 7,593,812; 7,650,234; 7,702,455 and 7,739,039. Case before the United States District Court for the District of Nevada (Case No. 2:11-cv-1578). Deposed 2/19/14; Retained 4/12 to present.

Beacon Navigation GmbH v. Toyota Motor Corporation, et al.
Retained by Kirkland & Ellis, LLP on behalf of defendants Toyota Motor Corporation; Toyota Motor North America, Inc.; Toyota Motor Sales, U.S.A. Inc.; Toyota Motor Engineering & Manufacturing North America, Inc.; Toyota Motor Manufacturing, Indiana, Inc.; Toyota Motor Manufacturing, Kentucky, Inc.; Toyota Motor Manufacturing Mississippi, Inc.; Mazda Motor Corporation; Mazda Motor of America, Inc.; Fuji Heavy Industries, Ltd.; Fuji Heavy Industries U.S.A. Inc.; Subaru of America, Inc.; Jaguar Land Rover North America, LLC; Jaguar Cars Limited; Land Rover; Volvo Car Corporation; and Volvo Cars of North America, LLC; Adduci Mastriani & Schaumberg, LLP on behalf of defendants Suzuki and Garmin; Crowell-Moring on behalf of General Motors; Dickstein Shapiro on behalf og Chrysler Group, LLC; Finnegan, Henderson, Farabow, Garrett & Dunner on behalf of Bayerische Motoren Werke AG, BMW of North America, LLC, and BMW Manufacturing Co. LLC; Fish & Richardson on behalf of Honda Motor Co., Ltd., Honda North America, Inc., American Honda Motor Co., Inc., Honda Manufacturing of Alabama, LLC, Honda Manufacturing of Indiana, LLC, and Honda of America, Mfg., Inc.; Frommer Lawrence and Haug, LLP on behalf of Dr. Ing. h.c.F. Porsche AG and Porsche Cars North America, Inc.; Hogan Lovells on behalf of Daimler AG, Mercedes-Benz USA, LLC, or Mercedes-Benz U.S. International, Inc.; Quinn-Emanuel on behalf of Nissan and

Ford. Case before the US International Trade Commission, Washington D.C., in the matter of: "Certain Automotive Navigation Systems, Components Thereof, and Products Containing Same, Inv. No. 337-TA-814. Case withdrawn by Plaintiff. Retained 1/12 – 4/12.

Beacon Navigation GmbH v. Toyota Motor Corporation, et al.
Retained by Kirkland & Ellis, LLP on behalf of defendants Toyota Motor Corporation; Toyota Motor North America, Inc.; Toyota Motor Sales, U.S.A. Inc.; Toyota Motor Engineering & Manufacturing North America, Inc.; Toyota Motor Manufacturing, Indiana, Inc.; Toyota Motor Manufacturing, Kentucky, Inc.; Toyota Motor Manufacturing Mississippi, Inc.; Mazda Motor Corporation; Mazda Motor of America, Inc.; Fuji Heavy Industries, Ltd.; Fuji Heavy Industries U.S.A. Inc.; Subaru of America, Inc.; Jaguar Land Rover North America, LLC; Jaguar Cars Limited; Land Rover; Volvo Car Corporation; and Volvo Cars of North America, LLC. Multiple cases before the United States District Court for the District of Delaware. Case numbers 1:11-cv-00942-UNA, 1:11-cv-00941-UNA, 1:11-cv-00951-UNA, 1:11-cv-00952-UNA, 1:11-cv-00936-UNA, 1:11-cv-00937-UNA, 1:11-cv-00955-UNA, 1:11-cv-00959-UNA, and 1:11-cv-00960-UNA. Currently stayed. Retained 1/12 to Present.

Beacon Wireless Solutions, Inc., et al., v. Garmin International, Inc., et al.
Retained by Shook, Hardy and Bacon, LLP as an expert on behalf of defendant Garmin. Matter involves alleged trade secret misappropriation. Case before the United States District Court for the Western District of Virginia, Harrisonburg Division (Case No. 5:11-cv-00025). Testified in Court: 5/25/12. Retained 12/11 to 5/25/12.

Tramontane IP, LLC v. Garmin Int'l, Inc., et al.
Retained by Shook, Hardy and Bacon, LLP as an expert on behalf of defendant Garmin. Patents-in-suit are U.S. Patents 6,526,268 and 7,133,775. Case before the United States District Court for the Eastern District of Virginia (Case No. 1:2011-cv-00918). Case Settled. Retained 11/11 to 12/11.

Sourceprose, Inc. v. AT&T, Inc., MetroPCS Communications, Inc., *et al.*
Retained by Kilpatrick Townsend as an expert on behalf of defendant AT&T. Patents-in-suit are US Patent Nos. 7,142,217 and 7,161,604. Case before the United States District Court for the Western District of Texas, Austin Division. Case number 1:11-cv-00117. Retained 11/11 to present.

Furuno Electric Co., Ltd. and Furuno U.S.A., Inc. v. Honeywell International, Inc.
Retained by Quinn-Emanuel as an expert on behalf of complainant Furuno. Case before the US International Trade Commission, Washington D.C., in the matter of: "Certain GPS Navigation Products, Components Thereof, and Related Software," Investigation number 337-TA-810. Patents-in-suit are U.S. Patent Nos. 6,084,565; 7,095,367; 7,089,094; and 7,161,561. Case settled. Retained 8/11 – 12/11.

Honeywell International, Inc. v. Furuno Electric Co., Ltd. and Furuno U.S.A., Inc.
Retained by Quinn-Emanuel as an expert on behalf of respondent Furuno. Case before the US International Trade Commission, Washington D.C., in the matter of: "Certain GPS Navigation

Products, Components Thereof, and Related Software," Investigation number 337-TA-783. Patents-in-suit are U.S. Patent Nos. 7,209,070; 6,865,452; 5,461,388; and 6,088,653. Case Settled. Retained 8/11 – 12/11.

Triangle Software, Inc. v. Garmin International, Inc.
Retained by Weil, Gotshal & Manges, LLP., as an expert on behalf of defendant Garmin. Patents-in-suit are US Patents 7,557,730, 7,221,287, 7,375,649, 7,508,321 and 7,702,452. Case before the United States District Court, Eastern District of Virginia, Case No. 1:10-cv-1457 CMH/TCB. Deposed 7/28/11; Testified in Court: 11/3/11 (jury trial). Retained 4/11 to 11/11.

Garmin International, Inc. v. Pioneer Corporation and Pioneer Electronics (USA), Inc.
Retained by Shook, Hardy and Bacon, LLP as an expert on behalf of plaintiff Garmin. Patents-in-suit are U.S. Patents 5,365,448; 5,424,951; and 6,122,592. Case before the United States District Court for the District of Kansas. Case No. 10-CV-2080 JWL/GLR. Declarative Judgment action stayed. Retained 3/11 to 11/11.

Visteon Global Technologies, Inc. And Visteon Technologies, LLC v. Garmin International, Inc.
Retained by Shook, Hardy and Bacon as an expert on behalf of defendant Garmin. Patents-in-suit are US Patents 5,544,060, 5,654,892, 5,832,408, 5,987,375 and 6,097,316. Case before the United States District Court, Eastern District of Michigan, Case No. 2:10-cv-10578--PDB-MAR. Deposed 10/9/12; Retained 12/10 to present.

Thomson Licensing SAS and Thomson Licensing, LLC. v. Realtek Semiconductor Corporation
Retained by Sidley-Austin as an expert on behalf of respondent Realtek Semiconductor. Case before the US International Trade Commission, Washington D.C., in the matter of: "Certain Liquid Crystal Display Devices, Including Monitors, Televisions, Modules, And Components Thereof," Investigation number 337-TA-741. Patent-in-suit is US Patent 6,121,941. Deposed 6/29/11; Testified in Court: 9/15/11 and 9/16/11. Retained 11/10 – 9/11.

Ambato Media, LLC. v. Clarion Co., LTD., *et al.*
Retained by Traurig-Greenberg as an expert on behalf of defendant Garmin. Patent-in-suit is US Patent 5,432,542. Case before the United States District Court for the Eastern District Of Texas, Marshall Division. Case number 2:09-CV-242. Deposed 4/26/12 and 5/10/12; Testified in Court: 7/11/12. Retained 10/10 to present.

Gabriel Technologies Corporation and Trace Technologies, LLC, v. Qualcomm Incorporated, Snaptrack, Inc. and Norman Krasner
Retained by Cooley-Godward as an expert on behalf of defendants Qualcomm, Snaptrack, and Krasner. Trade secret misappropriation case related to US Patents 6,377,209, 6,583,757, 6,661,372, 6,799,050, 6,861,980, 6,895,249, 7,254,402, 7,289,786, 7,319,876, 7,421,277, 7,446,655, 7,570,958, 7,574,195, and 7,660,588. Case before the Southern District of California San Diego Division, Case No. 08-cv-1992 MMA POR. Retained 6/10 to 7/12.

SiRF/CSR v. Global Locate/Broadcom Corporation

Retained by Wilmer-Hale as an expert on behalf of defendant Global Locate / Broadcom. Patents-in-suit are US Patents 5,663,735, 6,480,150, 6,519,466, 6,650,879, 6,882,827, 6,934,322, 7,412,157, 7,236,883, and 7,573,422. Case before the Central District of California, Case No. 8:06-cv-01216 and Case No. 8:10-cv-01281. Retained 9/10 to 1/11.

Pioneer Electronics v. Garmin Corporation
Retained by Shook, Hardy and Bacon, LLP as an expert on behalf of respondent Garmin In the matter of Certain Multimedia Display and Navigation Devices and Systems, Components Thereof, and Products Containing Same; Inv. No. 337-TA-694. Patents-in-suit are U.S. Patents 5,365,448; 5,424,951; and 6,122,592. Case before the U.S. International Trade Commission. Deposed on 7/29/10; Testified at technology tutorial (8/27/20) and in the evidentiary hearing (9/20/10). Retained 4/10 to 9/10.

EMSAT Advanced Geo-Location Technology, LLC and Location Based Services LLC, v. AT&T Mobility, LLC
Retained by Baker-Botts as an expert on behalf of defendant AT&T Mobility, LLC. Patents-in-suit are U.S. Patents 7,289,763; 5,946,611; 6,324,404; and 6,847,822. Case before the U.S. District Court for the Northern District of Ohio, Eastern Division, Civil Action No. 4:08 CV 822. Deposed 5/4/10; Testified in Court: 5/10/10 (Markman hearing). Retained 12/09 to 3/11.

Tendler Cellular of Texas, LLC v. AT&T Mobility, LLC, et al.
Retained by Baker-Botts as an expert on behalf of defendants AT&T Mobility, LLC, et al. Patents-in-suit are U.S. Patents 7,447,508; 7,305,243; 7,050,818; and 6,519,463. Case before the U.S. District Court for the Eastern District of Texas (Tyler), Civil Action No. 6:09-CV-00115. Retained 8/09 to 7/10.

Ambit Corporation v. Delta Air Lines, Inc., and Aircell LLC.
Retained by Sidley Austin, LLP., as an expert on behalf of defendants Delta Airlines, Inc., and Aircell, LLC. Patent-in-suit is US patent 7,400,858. Case before the US District Court, District of Massachusetts, Boston, Civil Action No. 1:09-CV-10217-WGY. Deposed 12/4/09; Testified in Court: 12/7/09 (evidentiary hearing), 7/10 (jury trial). Retained 8/09 to 7/10.

GPS Industries, Inc. and Optimal I.P. Holdings, L.P. v. Altex Corporation, et. al.
Retained by Hitchcock-Evert as an expert on behalf of defendants Altex Corporation, Deca International Corporation, Golflogix, Inc. and L1 Technologies. Patent-in-suit is US patent 5,364,093. Case before the US District Court, Northern District of Texas, Dallas Division, Civil Action No. 3-07-CV0831-K. Deposed 6/30/09. Retained 5/08 through 7/09.

Satellite Tracking of People, LLC v. Omnilink Systems, Inc.
Retained by DLA Piper as an expert on behalf of defendant Omnilink Systems, Inc.. Patent-in-suit is US patent RE39,909. Case before the US District Court, Eastern District of Texas, Marshall Division, Civil Action No. 2-08CV-116. 12/08 – 1/11.

SiRF Technology, Inc. v. Global Locate, Inc.
Retained by DLA Piper/WilmerHale as an expert on behalf of Global Locate. Patents-in-suit include US patents 6,304,216; 6,417,801; 6,606,346; 6,651,000; 6,704,651; 6,937,187;

7,043,363; 7,091,904; 7,132,980 and 7,158,080. Case before the US International Trade Commission, Washington D.C., in the matters of: "Certain GPS Devices and Products Containing Same," Investigation number 337-TA-602 (Global Locate, plaintiff) and "Certain GPS Chips, Associated Software and Systems, and Products Containing Same," Investigation number 337-TA-596 (SiRF Technologies, plaintiff). My work focused on the 7,043,363 and 7,091,904 patents in defense of Global Locate/Broadcom from June 2007 through March 2008. Deposed 1/18-1/19/08; testified at trial 3/18-3/19/08.

Intellectual Science and Technology
Retained Dykema Gossett, PLCC as a technical expert on patent infringement issues related to "suspend-to-RAM" technologies in personal computers. Pre-litigation work.

Intellectual Science and Technology, Inc., v. Sony, JVC and Panasonic
Retained Dykema Gossett, PLCC as a technical expert on patent infringement issues related to US Patent 5,748,575, US Patent 6,222,799, US Patent 6,785,198, US Patent 6,662,239 and US Patent 6,717,890. Sony Electronics Inc., case number 2:06-CV-10406, JVC Americas Corp., case number 2:06-CV-10409 and Panasonic Corporation of North America case number 2:06-CV-10412. Cases heard in United States District Court, Eastern District of Michigan, Southern Division. Expert for Intellectual Science and Technology, Inc. Dec 2006 – 2008..

Kirsch Technologies v. Xerox, Canon
Retained Dykema Gossett, PLCC as an Expert Witness on patent infringement issues related to US Patent 4,816,911, Canon case number CA 00-72775, Xerox case number CA 00-72778, cases heard in United States District Court, Eastern District of Michigan, Southern Division. Expert for Kirsch Technologies. Nov 2006 – 2008..

American Video Graphics v. ATI Technologies
Retained by Sidley, Austin, Brown & Wood, Dallas, TX as a technical expert on patent infringement issues related to US Patents 5,132,670, 5,109,520, 5,084,830, 4,761,642, 4,742,474, 4,734,690, 4,730,185 and 4,694,286. Hewlett-Packard Co., et al., defendants, case number CA 6:04-CV-379-LED and Sony Corporation of America et al., defendants, case number CA 6:04-CV-399-LED. Cases heard in United States District Court, Eastern District of Texas, Tyler Division. Expert for ATI Technologies, intervener. Jan 2005 – Sep 2005.

Microsoft v. EMC
Dewey Ballantine, LLP, Washington, D.C., as a technical expert on patent infringement issues related to US Patents 5,588,147; 5,689,700; 6,393,466; 6,424,151; 6,490,594; and 6,632,248. Wrote a declaration on behalf of Microsoft. Oct 2004 – Jan 2005.

Optimum Return v. Meier Brothers
Retained by Sidley, Austin, Brown & Wood, Dallas, TX as a technical expert on Copyright infringement allegations related to software owned by Optimum Return, LLC. Cyberkatz Consulting, Inc., Handsquare, LLC, Meier Brothers, et al., defendants, case number CA 3-03CV1064-D. Case heard in United States District Court, Northern District of Texas, Dallas Division. Expert for Meier Brothers. July 2004.

Parental Guide of Texas, Inc. v. Funai Corp., et. al.
Technical expert for defendants JVC and Panasonic in their dispute over non-infringement of US Patent number 4,605,964.

Elonex I.P. Holdings, LTD. and Elonex PLC, Phase II
Expert witness for defendants Chuntex, Acer, Tatung, Lite-On, Daewoo and Envision in their dispute with Elonex non-infringement and validity for US Patent numbers 5,389,952; 5,648,799; and 5,880,719.

Storage Computer Corporation vs. Veritas Software
Technical expert for the plaintiff in matters involving Patents US 5,257,367; US 5,893,919; and US 6,098,128.

Storage Computer Corporation vs. Seagate Technology LLC
Technical expert for the defendant in matters involving US Patent RE 34,100.

Elonex I.P. Holdings, LTD. and Elonex PLC, vs. Packard Bell et. al., CA 98-689-GMS
Expert witness for defendants ViewSonic Corp., Dell Computer Corporation, MAG Technology USA, Princeton Graphic Systems, Inc., Micron Electronics, Sony Electronics and Capetronic Computer USA in their dispute with Elonex non-infringement and validity for US Patent numbers 5,389,952; 5,648,799; and 5,880,719.

1.4.2 Engineering Consulting

Offspring Media Inc.
Technical consultant for the development of real-time auralization algorithms for integration into a consumer electronics product. Sep 2004.

Raytheon Company, Sudbury, MA
Development of techniques and requirements for implementing a fault tolerant computer system using software implemented fault tolerance (SIFT) techniques on commercial off-the-shelf processing hardware. The resultant system is to be used for highly reliable radar data and signal processing.

TVM Techno Venture Management
Provided consulting services to assist in assessing the technical claims of a company pursuing venture capital investment for a hardware implemented RAID 5 system .

Keyhold Engineering Inc., Northboro, MA.
Development of a prototype system for automatically calibrating multiple channel audio systems.

American Navigation Systems Inc., Milbury, MA.
Consulting on the development of the hand-held personal navigation and mapping system.

Lincoln Laboratory, Bedford, MA.

Simulated, tested and evaluated a GPS integrity monitoring algorithm developed at Lincoln Laboratories.

1.5 Licenses and Certifications

1.5.1 Commercial

General Radiotelephone Operator License
Ship RADAR endorsement

1.5.2 Amateur

Amateur Extra class radio operator license.

2. Courses Taught at WPI

2.1 Course Descriptions

Short descriptions of the courses taught are as follows:

EE572N Advanced System Architecture

This course focuses on the architectural techniques used to achieve high-performance in SISD and SIMD computer systems. In this course the interaction between the software application and hardware architecture and the effect of this interaction on achievable performance is stressed. The course begins by covering the basic architectural tricks used to enhance system performance and ends with a series of case studies that analyze specific architectures such as the CRAY and CDC vector supercomputers, the MasPar, the Connection Machine, the ICL DAP, and others.

ECE505 Computer Architecture

This course is an introductory graduate course in computer architecture. Most aspects of CPU architecture are covered using a combined hardware/software approach. Specific topics include datapath design, memory systems, microprogramming, memory management, operating systems, and instruction set design.

ECE579M Real-Time System Design

This course focuses on the design of computer systems for which the timeliness of producing results is a critical factor for establishing the correctness of the system design. Topics covered include hardware specification, real-time operating systems and programming, scheduling, communications, and validation/verification. Issues and choices arising for single processor and distributed systems are also covered. Both hard and soft real-time system issues and the interactions between real applications and real systems is stressed.

ECE2010 Introduction to Electrical and Computer Engineering.

The objective of this course is to introduce students to the broad field of electrical and computer engineering within the context of real world applications. This course is designed for first-year students who are considering ECE as a possible major or for non-ECE students fulfilling an out-of-major degree requirement. The course will introduce basic electrical circuit theory as well as analog and digital signal processing methods currently used to solve a variety of engineering design problems in areas such as entertainment and networking media, robotics, renewable energy and biomedical applications. Laboratory experiments based on these applications are used to reinforce basic concepts and develop laboratory skills, as well as to provide system-level understanding. Circuit and system simulation analysis tools are also introduced and emphasized.

ECE2022 Introduction to Digital Circuits and Computer Engineering.

The objective of this course is to expose students (including first year students) to basic electrical and mathematical concepts that underlie computer engineering while continuing an introduction to basic concepts of circuits and systems in a hands-on environment. Experiments representing practical devices introduce basic electrical engineering concepts and skills which typify the study and practice of electrical and computer engineering. In the laboratory, the students construct, troubleshoot, and test analog and digital circuits that they have designed. They will also be introduced to the nature of the interface between hardware and software in a typical microprocessor based computer.

ECE2801 Foundations of Embedded Systems

This course teaches the principles of programming microprocessors and microcontrollers for real-time applications. Students are introduced to software engineering principles and are taught how to translate product specifications into engineering solutions.

ECE2799 ECE Design

This is a new course added to the curriculum that teaches sophomore Electrical Engineering students the basic principles of design. Topics are covered which range from project planning and management through manufacturing and implementation. Students are exposed to external factors influencing design such as safety, liability, cost, and other constraints.

ECE3801 Logic Circuits

This is an introductory course in logic circuit design. Topics covered include Boolean Logic, Algebraic minimization of logic equations, Karnaugh Maps, sequential machine design and timing analysis.

ECE3803 Introduction to Microprocessor Systems

This is an undergraduate-level first-course in microprocessor design. Topics covered include timing analysis, address decoding, memory system design, assembly language programming, programmed I/O, and digital/analog interfacing. Experiments are run using ISA-bus interfaces to standard PCs.

ECE3810 Advanced Digital System Design

This course addresses the design of advanced digital logic systems using VHDL to design, synthesize and model digital circuits, and to implement these circuits using Xilinx FPGA devices. The course emphasizes understanding functional design, designing for speed and power objectives, and testing. The course ends with students designing moderately complicated "system on a chip" (SOC) based systems

ECE4815 Computer Architecture (crosslisted as CS4515)

A first course in computer architecture. Essential aspects of CPU architecture are covered using a combined hardware/software approach. Students learn how a CPU interprets and processes instructions. Issues associated with interfacing hardware with software are covered in detail as are the hardware/software tradeoffs associated with performance optimization.

ECE4801 Microprocessor System Design

Microprocessor System Design is the second course in the microprocessor sequence. In this course, students learn the advanced concepts used in modern microprocessor systems. Topics such as system organization, dynamic and cache memory systems, communications, mixed language programming, and device driver design are covered.

ECE430X Fundamentals of Navigation Systems

This course introduces students to the fundamentals of navigation using electronic systems. The course covers types of navigation systems, how to interpret sensor data and sources of navigation system error. Topics include: types of navigation systems (dead reckoning, inertial, radio based systems), sensors and error sources, coordinate frames and transformations, system dynamics and measurement processing. Case studies explore the use of accelerometers, gyroscopes, GPS (including, differential and assisted GPS) as well as other types of navigation systems.

RBE 3001 Unified Robotics III

Third of a four-course sequence introducing foundational theory and practice of robotics engineering from the fields of computer science, electrical engineering and mechanical engineering. The focus of this course is actuator design, embedded computing and complex response processes. Concepts of dynamic response as relates to vibration and motion planning will be presented. The principles of operation and interface methods various actuators will be discussed, including pneumatic, magnetic, piezoelectric, linear, stepper, etc. Complex feedback

mechanisms will be implemented using software executing in an embedded system. The necessary concepts for real-time processor programming, re-entrant code and interrupt signaling will be introduced. Laboratory sessions will culminate in the construction of a multi-module robotic system that exemplifies methods introduced during this course.

RBE 3002 Unified Robotics IV

Fourth of a four-course sequence introducing foundational theory and practice of robotics engineering from the fields of computer science, electrical engineering and mechanical engineering. The focus of this course is navigation, position estimation and communications. Concepts of dead reckoning, landmark updates, inertial sensors, vision and radio location will be explored. Control systems as applied to navigation will be presented. Communication, remote control and remote sensing for mobile robots and tele-robotic systems will be introduced. Wireless communications including wireless networks and typical local and wide area networking protocols will be discussed. Considerations will be discussed regarding operation in difficult environments such as underwater, aerospace, hazardous, etc. Laboratory sessions will be directed towards the solution of an open-ended problem over the course of the entire term.

RBE 400x Robot System Engineering and Design

The designers of robotic systems start with a system requirement to define the mechanical, electrical and software systems which must work together to achieve the system goals. Typically, parallel teams of engineers will work concurrently to create the requirements document as well as model various aspects of the system to verify operational capabilities and the ability to meet time and budget constraints. For complex systems, the development of such teams can itself be a complex problem since the project has to be organized in such a way that parallel teams can work independently, yet have excellent communication channels and information passing to insure project success.

This course explores the tools and techniques used to develop complex systems. The topics covered include: requirements development; system architecture and partitioning; requirements flowdown; functional and interface specifications; trade studies; system modeling and simulation; system integration; as well as design verification and validation.

RBE500 Foundations of Robotics

Foundations and principles of processing sensor information in robotic systems. Topics include an introduction to probabilistic concepts related to sensors, sensor signal processing, multi-sensor control systems and optimal estimation. The material presented will focus on the types of control problems encountered when a robot must operate in an environment where sensor noise and/or tracking errors are significant. Techniques for assessing the stability, controllability and expected accuracy of multi-sensor control and tracking systems will be presented. Lab projects will involve processing live and synthetic data, robot simulation and projects involving the control of robot platforms.

3. List of Publications:

3.1 Journal Papers

[1] T. Padir, W.R. Michalson, et. al., "Implementation of an Undergraduate Robotics Engineering Curriculum," Computers in Education Journal, vol. I, no. 3, pp. 92-101, 2010.

[2] W. R. Michalson, A. Navalekar and H. Parikh, "Error mechanisms in indoor positioning systems without support from GNSS," The Journal of Navigation, Cambridge University Press, vol. 62, no. 2, pp. 239-49, 2009.

[3] H.K. Parikh and W. R. Michalson, "Impulse Radio - UWB or Multicarrier UWB for Non-GPS based Indoor Precise Positioning Systems," *NAVIGATION J. Inst. Nav*., Vol. 55, no. 1, 2008.

[4] I. F. Progri, W. R. Michalson, J. Wang and M.C. Bromberg, "Indoor Geolocation Using FCDMA Pseudolites: Signal Structure and Performance Analysis", *NAVIGATION J. Inst. Nav*., Vol. 54, No. 3, Fall 2007.

[5] I. F. Progri, M. C. Bromberg and W. R. Michalson, "Maximum-Likelihood GPS Parameter Estimation", *NAVIGATION J. Inst. Nav.*, vol. 52, no. 4, pp. 229-238, Winter, 2005-2006.

[6] I.F. Progri, W.R. Michalson, and D. Cyganski, "An OFDM/FDMA Indoor Geolocation System," *NAVIGATION J. Inst. Nav.*, vol. 51, no. 2, pp. 133-142, Summer, 2004.

[7] I.F. Progri, G. Bogdanov, V.C. Ramanna, and W.R. Michalson, "An Investigation Of A GPS Adaptive Temporal Selective Attenuator," *NAVIGATION J. Inst. Nav.*, Vol. 49 No. 3, pp. 137-147, 2002.

[8] G. Bogdanov, W. R. Michalson, and R. Ludwig, "A new apparatus for non-destructive evaluation of green-state powder metal compacts using the electrical resistivity method," *Measurement Science and Technology*, IOP Publishing, vol. 11, pp. 157-166, January 2000.

[9] B. Findlen, E. Reuter, R. Campbell, and W. R. Michalson, "Effects of time domain response on the sonic characteristics of microphones," *Journal of the Acoustical Society of America*, vol. 104, no. 3, pt. 2 pp. 1764, September 1998.

[10] J. Sedgwick, W. R. Michalson, and R. Ludwig, "Design of a Digital Gauss Meter for Precision Magnetic Field Measurements," IEEE Transactions on Instrumentation and Measurement, vol. 47, no. 4, pp. 972-977, August 1998.

[11] J. Stander, J Plunkett, W. Michalson, J. McNeill, and R. Ludwig, "A Novel Multi-Probe Resistivity Approach to Inspect Green-State Powdered Metallurgy Compacts," Journal of Non-Destructive Evaluation, vol. 16, no. 4, pp. 205-214, 1997.

[12] Woltereck, R. Ludwig, and W. Michalson, "A Quantitative Analysis of the Separation of Aluminum Cans Out of a Waste Stream Based on Eddy Current Induced Levitation," IEEE Tr*ansactions on Magnetics,* vol. 33, no. 1, pp. 772-781, January 1997.

[13] W. R. Michalson, "Auralization on a Laptop PC," abstract appears in *The Journal of the Acoustical Society of America* , vol. 100 No. 4, Pt 2, pp. 2608, October 1996.

[14] R. H. Campbell, S. K. Martin, I. Schneider, and W. R. Michalson, "Analysis of Mosquito Wingbeat Sound," *The Journal of the Acoustical Society of America* vol. 100 No. 4, Pt 2, pp. 2710, October 1996.

[15] W. R. Michalson, "Ensuring GPS Navigation Integrity using Receiver Autonomous Integrity Monitoring," *IEEE Aerospace and Electronic Systems Magazine*. vol. 10, no. 10, pp. 31-34, October 1995.

[16] S. Clayton, R. J. Duckworth, W. Michalson, A. Wilson, "Determining Update Latency Bounds in Galactica Net," *Concurrency: Practice, and Experience*, vol. 7, no. 7, pp. 595-611, October 1994.

3.2 Conference Papers

[1] K.C. Seals, W.R. Michalson, R.J. Hartnett and P.F. Swaszek, "Using Both GPS L1 C/A and L1C: Strategies to Improve Acquisition Sensitivity," Proc. 26th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2013), Sep. 16-20, 2013.

[2] K.C. Seals, W.R. Michalson, R.J. Hartnett and P.F. Swaszek, "Semi-Coherent and Differentially Coherent Integration for L1C Acquisition," Proc. 2013 International Technical Meeting of the Institute of Navigation (ION ITM 2013), Honolulu, HI, Apr. 22-25, 2013.

[3] J.M. Barrett, M.G. Gennert, W. R. Michalson, et. al., "Development of a Low-Cost, Self-Contained, Combined Vision and Inertial Navigation System," 2013 International Conference on Technologies for Practical Robotic Applications, Boston, MA Apr. 22-23, 2012.

[4] K.C. Seals, W.R. Michalson, R.J. Hartnett and P.F. Swaszek, "Analysis of L1C Acquisition by Combining Pilot and Data Components Over Multiple Code Periods," Proc. 2013 International Technical Meeting of the Institute of Navigation (ION ITM 2013), San Diego, CA, Jan. 28-30, 2013.

[5] K.C. Seals, W.R. Michalson, R.J. Hartnett and P.F. Swaszek, "Analysis of Coherent Combining for L1C Acquisition," Proc. 25th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2012), pp.384-393, Nashville, TN, Sep. 17-21, 2012.

[6] J.M. Barrett, M.G. Gennert, W. R. Michalson and J.L. Center, "Analyzing and Modeling an IMU for Use in a Low-Cost Combined Vision and Inertial Navigation System," 2012 International Conference on Technologies for Practical Robotic Applications, Boston, MA Apr. 23-24, 2012.

[7] G. Fischer, W. R. Michalson, T. Padir and G. Pollice,"Development of a Laboratory Kit for Robotics Engineering Education" AAAI 2010 Spring Symposium on Educational Robotics and Beyond: Design and Evaluation, Mar. 22-24, Palo Alto, CA, 2010.

[8] W.R. Michalson, and F. J. Looft, "Designing Robotic Systems: Preparation for an Interdisciplinary Capstone Experience," American Society of Engineering Educators 2010 Annual Conference, Louisville, KY, Jun 20-23, 2010.

[9] W.R. Michalson, S.J. Bitar and R.C.Labonte, "The Technical, Process, and Business Considerations For Engineering Design – A 10 Year Retrospective," American Society of Engineering Educators 2010 Annual Conference, Louisville, KY, Jun 20-23, 2010.

[10] M. Gennert, M. Demietrio and W. R. Michalson, "A Robotics Engineering M.S. Degree," American Society of Engineering Educators 2010 Annual Conference, Louisville, KY, Jun 20-23, 2010.

[11] R. Beach, W. R. Michalson, et. al., "Robotics Innovations Competition and Conference (RICC): Building Community Between Academia and Industry Through a University-Level Student Competition," American Society of Engineering Educators 2010 Annual Conference, Louisville, KY, Jun 20-23, 2010.

[12] G. Tryggvason, W.R. Michalson, et. al., Teaching Multidisciplinary Design to Engineering Students: Robotics Capstone," American Society of Engineering Educators 2010 Annual Conference, Louisville, KY, Jun 20-23, 2010.

[13] A.C.Navalekar, W.R. Michalson, "Asymmetric throughput problem due to Push-to-talk (PTT) delays in CSMA/CA based heterogeneous Land Mobile Radio (LMR) networks", accepted for publication, Milcom 2009.

[14] A. Navalekar and W.R. Michalson, "Effects of Unintentional Denial of Service (DOS) due to PTT delays on performance of CSMA/CA based Adhoc Land Mobile Radio (LMR) networks", accepted for publication, ICST Adhocnet 2009.

[15] W. R. Michalson, et. al., "Unified Robotics: Balancing Breadth and Depth in Engineering Education", American Society of Engineering Educators 2009 Annual Conference, AC 2009-1681, Austin, TX, Jun 14-17, 2009.

[16] G. Tryggvason, W.R. Michalson, et. al., "Robotics Engineering: A New Discipline for a New Century", American Society of Engineering Educators 2009 Annual Conference, AC 2009-997, Austin, TX, Jun 14-17, 2009.

[17] M. DiBlasi, W. R. Michalson, et. al., "Social Networking in the FIRST Robotics Competition Community," ASEE Northeast Section Conference, University of Bridgeport, Apr 3-4, Bridgeport, CT, 2009.

[18] A. Navalekar, H.K Parikh and William R. Michalson,"Error Mechanisms in Indoor Positioning Systems without Support from GNSS", RIN NAV 08.

[19] A. Navelekar, W.R. Michalson, et. al., "Effects of Push-To-Talk (PTT) delays on Throughput Performance of CSMA/CA based Distributed Digital Radios (DDR) for Land Mobile Radio (LMR) Networks," 37th International Conference on Parallel Processing (ICPP-08), Portland, OR, Sep. 8-12, 2008.

[20] M. Ciraldi, W. Michalson, et. al., "The New Robotics Engineering BS Program at WPI," American Society of Engineering Educators 2008 Annual Conference, AC 2008-1048, Pittsburgh, PA, Jun 22-25, 2008.

[21] A.C.Navalekar, W.R. Michalson, "A New Approach to Improve BER Performance of a High Peak-to-Average Ratio (PAR) OFDM signal over FM based Land Mobile Radios (LMR)", IEEE WTS 08, Pomona, CA.

[22] A. Navelekar and W.R. Michalson, "Effects of Push-To-Talk (PTT) delays on CSMA based Capacity Limited Land Mobile Radio (LMR) Networks," *Proc. IEEE Intl. Symp. Wireless Pervasive Computing 2008 (ISWPC08)*, Santorini, Greece, May 7-9, 2008.

[23] H.K. Parikh and W.R. Michalson, "Error Mechanisms In An Rf-Based Indoor Positioning System," *Proc. 2008 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '08)*, Las Vegas, NV Mar 30 – Apr 4, 2008.

[24] B. Woodacre, W. Michalson, et. al., "WPI Precision Personnel Locator System - Automatic Antenna Geometry Estimation," to appear, *Proc. ION-NTM 2008*, San Diego, CA, Jan 27-29, 2008.

[25] H.K. Parikh, A. Navelekar and W.R. Michalson, "Issues in Achieving Precise Positioning Indoors Without Support from GNSS," *Proc. ION-NTM 2008*, San Diego, CA, Jan 27-29, 2008.

[26] D. Cyganski, W. Michalson, et. al., "WPI Precision Personnel Locator System - Evaluation by First Responders," *Proc. Institute of Navigation GNSS 2007*, Fort Worth, TX, September 25-28, 2007.

[27] I.F. Progri, W. R. Michalson, et. al., "Maximum Likelihood OFDMA Parameter Estimation," *Proc. Institute of Navigation GNSS 2007*, Fort Worth, TX, September 25-28, 2007.

[28] C.W. Kelley, W. R. Michalson, et. al., "Discrete vs. Continuous Carrier Tracking Loop Theory, Implementation, and Testing with Large BnT," *Proc. Institute of Navigation GNSS 2007*, Fort Worth, TX, September 25-28, 2007.

[29] W. R. Michalson and J. W. Matthews, "Distributed Digital Radios and WLAN Interoperability," 2007 IEEE Conference on Technologies for Homeland Security, May 16-17, Woburn, MA, 2007.

[30] I.F. Progri, W.R. Michalson, J. Wang, M.C. Bromberg, and R.J. Duckworth, "Requirements of a C-CDMA Pseudolite Indoor Geolocation System," *Proc. ION-AM 2007*, Cambridge, MA, pp. 654-658, Apr. 2007.

[31] D. Cyganski, R. J. Duckworth, S. Makarov, W. R. Michalson, et.al., "WPI Precision Personnel Locator System," *Proc. Institute of Navigation National Technical Meeting (NTM 2007),* Catamaran Resort Hotel, San Diego, CA, January 22-24, 2007.

[32] I.F. Progri, M.C. Bromberg, W.R. Michalson, and J. Wang, "A Theoretical Survey of the New GPS Signals (L1C, L2C, and L5)," *Proc. Institute of Navigation National Technical Meeting (NTM 2007),* Catamaran Resort Hotel, San Diego, CA, January 22-24, 2007.

[33] I.F. Progri, M.C. Bromberg, W.R. Michalson, and J. Wang, "Field Measurement Data on Support of a Unified Indoor Geolocation Channel Model," *Proc. Institute of Navigation National Technical Meeting (NTM 2007),* Catamaran Resort Hotel, San Diego, CA, January 22-24, 2007.

[34] I. F. Progri, J. Maynard, W.R. Michalson, and J. Wang, "The Performance and Simulation of a C-CDMA Pseudolite Indoor Geolocation System," Proc. Institute of Navigation GNSS 2006, Fort Worth, TX, September 26-29, 2006.

[35] I. F. Progri, W. Ortiz, W.R. Michalson, and J. Wang, "The Performance and Simulation of an OFDMA Pseudolite Indoor Geolocation System," Proc. Institute of Navigation GNSS 2006, Fort Worth, TX, September 26-29, 2006.

[36] J.W. Coyne, R. J. Duckworth, W. R. Michalson and H.K. Parikh, "2-D Radio Navigation System Using MC-UWB," *Proc. NAV 2005 – Pushing the boundaries, Royal Institute of Navigation*, London, Nov 1-3, 2005.

[37] H.K. Parikh, W.R. Michalson and R.J. Duckworth,, "MC-UWB Precise Positioning System –Field Tests, Results and Effect of Multipath," *Proc. Institute of Navigation GNSS 2005,* Long Beach Convention Center, Long Beach, CA, September, 2005.

[38] R.J. Duckworth, H.K. Parikh and W.R. Michalson, "Radio Design and Performance Analysis of a Multi-carrier Ultrawideband (MC-UWB) Positioning System," *Proc. Institute of Navigation National Technical Meeting (NTM 2005),* Catamaran Resort Hotel, San Diego, CA, January 24-26, 2005.

[39] H.K. Parikh and W.R. Michalson, "Performance Evaluation of the Receiver RF Front-End of a Precision Positioning System," *Proc. Institute of Navigation GNSS 2004,* Long Beach Convention Center, Long Beach, CA, September 21-24, 2004.

[40] W.R. Michalson and H. Ahlehagh, "A 3D Location Discovery Algorithm for Ad Hoc Networks," *ICWN-04, 2004 International Conference on Wireless Networks*, Las Vegas, pp. 567-572, Jun 21-24, 2004.

[41] I.F. Progri, W.R. Michalson and M.C. Bromberg, "Accurate Synchronization Using a Full Duplex DSSS Channel," *Proc. IEEE PLANS 2004*, Monterrey, CA, pp. 220-226, Apr 26-29, 2004.

[42] I.F. Progri and W.R. Michalson, "An Investigation of a DSSS-OFDM-CDMA-FDMA Indoor Geolocation SYSTEM," *Proc. IEEE PLANS 2004*, 2004, Monterrey, CA, pp. 662-670, Apr 26-29, 2004.

[43] D. Cyganski, J.A. Orr, W.R. Michalson, "Performance of a Precision Indoor Positioning System Using a Multi-Carrier Approach," *Proc. Institute of Navigation 2004 National Technical Meeting*, San Diego, CA, pp. 175-180, Jan 26-28, 2004.

[44] I.F. Progri, W.R. Michalson and M.C. Bromberg, "An Enhanced Acquisition Process of a Maximum Likelihood GPS Receiver," *Proc. Institute of Navigation 2004 National Technical Meeting*, San Diego, CA, pp. 390-398, Jan 26-28, 2004.

[45] D. Cyganski, J. Orr, W. Michalson, "A Multi-Carrier Technique for Precision Geolocation for Indoor/Multipath Environments", *Proc. Institute of Navigation GPS 2003,* Portland, OR, pp. 1069-1073, Sep 9-12, 2003.

[46] I.F. Progri, M.C. Bromberg and W.R. Michalson, "The Acquisition Process of a Maximum Likelihood GPS Receiver," *Proc. Institute of Navigation GPS 2003,* Portland, OR, pp. 2533-2542, Sep 9-12, 2003.

[47] W.R. Michalson, H. Ahlehagh and I.F. Progri, "Dynamic Node Location in an Ad Hoc Indoor Communications and Positioning Network," *Proc. Institute of Navigation GPS 2003,* Portland, OR, pp. 1185-1191, Sep 9-12, 2003.

[48] J. DeChiaro, C. Strus and W.R. Michalson, "A Personal Navigation Test Platform Based on Low-Cost Inertial Components," *Proc. Institute of Navigation GPS 2003,* Portland, OR, pp. 2869-2877, Sep 9-12, 2003.

[49] H. AhleHagh, W. R. Michalson and D. Finkel, "Statistical Characteristics of Wireless Network Traffic and its Impact on Ad Hoc Network Performance," Proc. Applied Telecommunication Symposium2003 Advancec Simulation Technologies Conference, Orlando, FL, pp. 66-71, Mar 30 – Apr 3, 2003.

[50] I.F. Progri, W.R. Michalson and D. Cyganski, "An OFDM/FDMA Indoor Geolocation System," *Proc. ION National Technical Meeting*, Anaheim, CA, pp. 272-281, Jan 22-24, 2003.

[51] I.F. Progri and W.R. Michalson, "Synchronization of Measurements of Power System Harmonics By Means of the Global Positioning System," *Proc. ION Annual Meeting*, Albuquerque, NM, pp. 372-382, Jun 24-26, 2002.

[52] I.F. Progri, W.R. Michalson, and M.C. Bromberg, "A recursive solution to the generalized eigenvalue problem," *Proc. ION Annual Meeting*, Albuquerque, NM, pp. 154-162, Jun 24-26, 2002.

[53] I.F. Progri and W.R. Michalson, "A Combined GPS Satellite/Pseudolite System for Category III Precision Landing," *Proc. IEEE PLANS*, pp. 212-218, Apr 15-17, 2002.

[54] I.F. Progri, W.R. Michalson and M. Bromberg, "A Comparison Between the Recursive Cholesky and MGSO Algorithms," *Proc. Institute of Navigation 2002 National Technical Meeting*, San Diego, CA, pp. 655-665, Jan 28-30, 2002.

[55] I.F. Progri, W.R. Michalson and M. Bromberg, "A Study of a Blind Adaptive Algorithm in the Time and Frequency Domain," *Proc. Institute of Navigation 2002 National Technical Meeting*, San Diego, CA, pp. 439-447, Jan 28-30, 2002.

[56] L. Polizzotto and W. R. Michalson, "The Technical, Process, and Business Considerations for Engineering Design," *Frontiers in Education 2001,* Reno, NV, pp. F1G-19-F1G-24, Oct 10-13, 2001.

[57] I.F. Progri and W.R. Michalson, "An Improved Adaptive Spatial Temporal Selective Attenuator," *Proc. Institute of Navigation GPS 2001,* Salt Lake City, UT, pp. 932-938, Sep 11-14, 2001.

[58] I.F. Progri and W.R. Michalson, "An Alternative Approach to Multipath and the Near-Far Problem for Indoor Geolocation Systems," *Proc. Institute of Navigation GPS 2001,* Salt Lake City, UT, pp. 1434-143, Sep 11-14, 2001.

[59] W.R. Michalson and I.F. Progri, "An Investigation of the Adaptive Spatial Temporal Selective Attenuator," *Institute of Navigation GPS 2001,* Salt Lake City, UT, pp 1985-1996, Sep 11-14, 2001.

[60] I.F. Progri and W.R. Michalson, "The Impact of Proposed Pseudolite's Signal Structure on the Receiver's Phase and Code Error," *Proc. ION Annual Meeting*, Albuquerque, NM, pp. 414-422, Jun 11-13, 2001.

[61] I.F. Progri, J.M. Hill and W.R. Michalson, "An Investigation of the Pseudolite's Signal Structure for Indoor Positioning," *Proc. ION Annual Meeting*, Albuquerque, NM, pp. 453-462, Jun 11-13, 2001.

[62] J. M. Hill and W. R. Michalson, "Real Time Verification of Bit-Cell Alignment for C/A Code Only Receivers," *Proc. ION Annual Meeting,* pp. 463-468, Jun 11-13, 2001.

[63] I. Progri , J. M. Hill and W. R. Michalson, "The Impact of the Proposed Pseudolite's Signal Structure on the Receiver's Phase and Code Error," *Proc. ION Annual Meeting,* pp. 414-422, Jun 11-13, 2001.

[64] I.F. Progri, W.R. Michalson and R. Chassaing "Fast and efficient filter design and implementation on the TMS320C6711 digital signal processor," *Proc. Student Forum ICASSP*, Salt Lake City UT, May 2001.

[65] I. Progri and W. R. Michalson, "An Innovative Navigation Algorithm Using a System of Fixed Pseudolites," *Institute of Navigation National Technical Meeting,* pp. 619-627, Long Beach, CA, Jan 22-24, 2001.

[66] I. Progri, J.M. Hill and W. R. Michalson, "A Doppler-based navigation algorithm," *Institute of Navigation National Technical Meeting,* pp. 482-490, Long Beach, CA, Jan 22-24, 2001.

[67] I. Progri , J. M. Hill and W. R. Michalson, "Assessing the Accuracy of Navigation Algorithms Using a Combined System of GPS Satellites and Pseudolites," *Institute of Navigation National Technical Meeting,* Long Beach, CA, pp 473-481, Jan 22-24, 2001.

[68] J. M. Hill and W. R. Michalson, "Design of a Stable Discrete Time Costas Loop," *Institute of Navigation National Technical Meeting,* pp. 228-234, Jan 22-24, 2001.

[69] J. M. Hill and W. R. Michalson, "Techniques for Reducing the Near-Far Problem in Indoor Geolocation Systems," *Institute of Navigation National Technical Meeting,* pp. 860-865, Jan 22-24, 2001.

[70] I. Progri and W. R. Michalson, "Adaptive Spatial and Temporal Selective Attenuator in the Presence of Mutual Coupling and Channel Errors," *Institute of Navigation GPS 2000,* Salt Lake City, UT, pp. 462-470, Sep 19-22, 2000.

[71] W. R. Michalson, J. A. Orr and D. Cyganski, "A System for Tracking and Locating Emergency Personnel Inside Buildings," *Institute of Navigation GPS 2000,* Salt Lake City, UT, pp. 560-568, Sep 19-22, 2000.

[72] W. R. Michalson and I. Progri, "Assessing the Accuracy of Underground Positioning Using Pseudolites," *Institute of Navigation GPS 2000,* Salt Lake City, UT, pp. 1007-1015, Sep 19-22, 2000.

[73] I. Progri and W. R. Michalson, "Performance Evaluation of Category III Precision Landing Using Airport Pseudolites," *IEEE Conf. Position, Location, and Navigations Systems (PLANS),* Mar, pp. 262-269, 2000.

[74] R. Ludwig, G. Bogdanov, W. Michalson, and D. Apelian, "Instrumentation Development for Crack Detection of Surface and Subsurface Defects in Green-State P/M Compacts through Multi-Probe Electric Resistivity Testing," *Review of Progress in NDE*, Snowbird Ski and Summer Resort, Snowbird UT, Jul 19-24, 1998.

[75] W. R. Michalson, W. Cidela, et. al., "A GPS-Based Hazard Detection and Warning System," in review, " *ION GPS-96, 9th International Meeting of the Satellite Division of the Institute of Navigation*, pp. 167-175, Kansas City, MO, Sep 17-20, 1996.

[76] W. R. Michalson and R. L. Labonte, "Capstone Design in the ECE Curriculum: Assessing the Quality of Undergraduate Projects at WPI," *American Society of Engineering Educators 1996 Annual Conference (CD-ROM)*, session 1232 Washington, D.C., Jun, 1996.

[77] W. R. Michalson, J. Single, M. Spadazzi, and M. Wehr, "A Real-Time GPS System for Monitoring Forestry Operations," *Sixth Biennial Forest Service Remote Sensing Applications Conference,* pp. 264-269, Denver, CO, May 1-3, 1996.

[78] W. R. Michalson and D. Nicoletti, "Computers in Introductory and Upper-Level ECE Courses," *American Society of Engineering Educators 1995 Annual Conference*, Anaheim, CA, Jun 25-28, 1995, pp. 2795-99.

[79] W. R. Michalson, D. B. Cox, and H. Hua, "GPS Carrier-Phase RAIM," *ION GPS-95, 8th International Meeting of the Satellite Division of the Institute of Navigation*, Palm Springs, CA., pp. 1975-1984, Sep 12-15, 1995.

[80] J. Bernick and W. R. Michalson, "UDSRAIM: An Innovative Approach to Increasing RAIM Availability," *ION GPS-95, 8th International Meeting of the Satellite Division of the Institute of Navigation*, Sep 12-15, Palm Springs, CA., pp. 1965-1973, 1995.

[81] C. Easton and W. R. Michalson, "Effects of Worst Case Geometries on RAIM Testing," *ION GPS-95, 8th International Meeting of the Satellite Division of the Institute of Navigation*, Sep 12-15, Palm Springs, CA., pp. 2015-2022, 1995.

[82] D. B. Cox and W. R. Michalson, "Use of Uncorrected GPS Carrier Phase Measurements for Incremental RAIM with WAAS," ION 51st Annual Meeting, Jun 5-7, Colorado Springs, CO., pp. 515-520, 1995.

[83] W. Michalson, et. al., "RAIM Availability for Augmented GPS-Based Navigation Systems," *ION GPS-94, 7th International Meeting of the Satellite Division of the Institute of Navigation*, pp. 587-95, Sep 20-23, 1994.

[84] V. G. Virball, W. Michalson, et. al., "A GPS Integrity Channel Based Fault Detection and Exclusion Algorithm Using Maximum Solution Separation," *Proceedings of the 1994 IEEE Position Location and Navigation Symposium (PLANS-94)*, pp. 747-54, Las Vegas, Apr 11-15, 1994.

[85] W. Michalson and C. Easton, "Experiences Implementing the Supplemental MOPS Off-Line Test Procedure," *ION GPS-93, 6th International Meeting of the Satellite Division of the Institute of Navigation*, pp. 519-27, Sep 22-24, 1993.

[86] D. Beatovic, P.L. Levin, A. Meroth, M. Spasojevic, W. R. Michalson, A. Unstundag, "Iterative Matrix Solvers for Large Full Systems," *Eighth International Symposium on High Voltage Engineering*, Aug 23-27, Yokohama, Japan, 1993.

[87] E. Schnieder, W. R. Michalson, "A Comparison of Large Guided-Wave Interconnection Networks for Optical Computation Systems," SPIE Conf. on Optoelectronic Interconnects OE/LASE 93.

[88] E. Schnieder, W. R. Michalson, "Integrated Guided-Wave Crossbar Interconnection of SEED Arrays," SPIE Conf. on Optoelectronic Interconnects OE/LASE 93.

[89] W. R. Michalson, "The Application of Neural Networks to Nonlinear Filtering," Proc. SPIE Cooperative Intelligent Robotics in Space III, pp. 219-228, Boston, 1992.

[90] W. R. Michalson and C. S. Tocci, "Exploiting Programmable Devices in High-Performance System Design: Current Trends," *Proc. Electro 92*, vol 2, pp. 104-109, May 12-14, 1992.

[91] S. Clayton, R. J. Duckworth, W. Michalson, A. Wilson, "Determining Update Latency Bounds in Galactica Net," *Proc. IEEE Conf. on High-Performance Distributed Computing*, pp. 104-111, Sep 9-11, 1992.

[92] R. J. Duckworth, J. E. Lavallee, W. R. Michalson, L. Becker, and P. Green, "The Development of Intelligent Real-Time Systems Using Ada," *12th IEEE Symposium on Real-Time Systems*, Dec 3-6, 1991.

[93] W. Jessop, W. Michalson, and R. Some, "Fault Injection for Verifying Fault Tolerant System Behavior," *Workshop on Experimental Evaluation*, El Segundo, CA, May 1-3, 1990.

[94] W. R. Michalson and P. Heldt, "A Hybrid Architecture for the ART 2 Neural Model," *Proc. International Joint Conference on Neural Networks*, pp. 167-70, Washington D.C., Jan 15-19, 1990.

[95] W. R. Michalson, "A Review of the Current State of Logic Synthesis," *2nd Annual IEEE ASIC Seminar and Exhibit*, Rochester, NY, Sep 25- 28, 1989.

[96] P. E. Green and W. R. Michalson, "Real-Time Evidential Reasoning and Network Based Processing," *Proc. IEEE 1st International Conference on Neural Networks*, pp. 359-365, San Diego, CA, Jun 21-24, 1987.

[97] P. E. Green, R. J. Juels, and W. R. Michalson, "Real Time Artificial Intelligence Architecture," *Proc. Workshop on Future Directions in Computer Architecture and Software*, pp. 328-330, Charleston, SC, May 5-7, 1986.

3.3 Book Chapters

[1] W. Michalson and E. Schnieder, "An Approach for Implementing a Reconfigurable Optical Interconnection Network for Massively Parallel Computers," in *Optical Interconnection - Foundations and Approaches*, C. Tocci and H. J. Caulfield Eds., Artech House, proposed release January 1994.

3.4 Patents

Precision location methods and systems
United States Patent 8,928,459, Issued January 6, 2015

The invention describes systems and methods for determining the location of a transmitter by jointly and collectively processing the full sampled signal data from a plurality of receivers to form a single solution.

Apparatus and methods for addressable communication using voice-grade radios
United States Patent 8,284,711, Issued 9 October 2012
The invention relates to methods and apparatus for conducting directed communication using voice-grade radios. The methods and apparatus can be used to form a packet-switched wireless network using legacy analog transceivers, providing, e.g., both data and voice-over-Internet Protocol communication.

Multi-channel electrophysiologic signal data acquisition system on an integrated circuit.
United States Patent 7896807, issued 3 March 2011.
A physiologic data acquisition system includes an analog input, a sigma-delta front end signal conditioning circuit adapted to subtract out DC and low frequency interfering signals from and amplify the analog input before analog to digital conversion. The system can be programmed to acquire a selected physiologic signal, e.g., a physiologic signal characteristic of or originating from a particular biological tissue. The physiologic data acquisition system may include a network interface modulating a plurality of subcarriers with respective portions of an acquired physiologic signal. A receiver coupled to the network interface can receive physiologic data from, and send control signals and provide power to the physiologic data acquisition system over a single pair of wires. The network interface can modulate an RF carrier with the plurality of modulated subcarriers and transmit the resulting signal to the receiver across a wireless network. An integrated circuit may include the physiologic data acquisition system. Also included are methods for acquiring physiologic data comprising the step of selectively controlling an acquisition circuit to acquire the physiologic signal.

Methods and apparatus for high resolution positioning. United States Patent 7292189. The invention relates to a method of signal analysis that determines the location of a transmitter and to devices that implement the method. The method includes receiving by at least three receivers, from a transmitter, a first continuous-time signal having a first channel. The first channel includes a first plurality of signal carriers having known relative initial phases and having known frequencies which are periodically spaced and which are orthogonal to one another within a first frequency range. The signal analysis method also includes determining the phase shifts of the carriers of the first channel resulting from the distance the carriers traveled in reaching the first receiver. Analysis of the phase shifts yields time difference of arrival information amongst the receivers, which is further processed to determine the location of the transmitter. 6 Nov 2007.

A Reconfigurable Indoor Geolocation System, US Patent Number 7,079,025. A portable reconfigurable geolocation system is provided. The system includes a portable user node and one or more portable pseudolite nodes in communication one another and with the user node. Each of the user nodes and pseudolite nodes includes a transmitter that generates a signal on one or more carrier frequencies. Each signal is modulated with digital signals necessary to establish distances between the nodes and to convey data between the nodes. Each node also includes a receiver for receiving and demodulating the signals transmitted between the nodes, and a processor for receiving the demodulated signals, extracting data values and derived values from the

demodulated signals and determining a three-dimensional position of each node in the system. Issued 18 Jul 2006.

Auto-Calibrating Surround System, United States Patent 7158643. A multi-channel surround sound system and method is described that allows automatic and independent calibration and adjustment of the frequency, amplitude and time response of each channel of the surround sound system. The disclosed auto-calibrating surround sound (ACSS) system includes a processor that generates a test signal represented by a temporal maximum length sequence (MLS) and supplies the test signal as part of an electric input signal to a loudspeaker. A microphone coupled to the processor receives the signal in a listening environment. The processor correlates the received sound signal with the test signal in the time domain and determines from the correlated signals a whitened response of the audio channel in the listening environment. Issued 2 Jan 2007.

Hand-held GPS-mapping device, US. Patent Number 5,987,380. A hand-held navigation, mapping and positioning device contains a GPS receiver, a database capable of storing vector or bit mapped graphics, a viewing port, an embedded processor, a simplified user interface, a data compression algorithm, and other supporting electronics, The viewport is configured such that the data presented in the viewport if clearly visible in any ambient light condition. The database stores compressed image data which might include topographical map data, user annotations, building plans, or any other image. The system includes an interface to a personal computer which may be used to annotate or edit graphic information externally to the device for later upload. In addition, the device contains a simple menu-driven user interface which allows panning and zooming the image data, marking locations of interest, and other such functions. The device may be operated from an internal rechargeable battery, or powered externally. , Issued 16 Nov 1999.

Hand-held GPS-mapping device, US. Patent Number 5,902,347. A hand-held navigation, mapping and positioning device contains a GPS receiver, a database capable of storing vector or bit mapped graphics, a viewing port, an embedded processor, a simplified user interface, a data compression algorithm, and other supporting electronics. The viewpoint is configured such that the data presented in the viewport is clearly visible in any ambient light condition. The database stores compressed image data which might include topographical map data, user annotations, building plans, or any other image. The system includes an interface to a personal computer which may be used to annotate or edit graphic information externally to the device for later upload. In addition, the device contains a simple menu-driven user interface which allows panning and zooming the image data, marking locations of interest, and other such functions. The device may be operated from an internal rechargeable battery, or powered externally. Issued 11 May 19/99.

3.5 Professional Presentations

American Ambulance Association Annual Meeting: Low-cost VHF/UHF Interoperability for digital telemetry, Las Vegas, NV, Dec. 2005.

California Ambulance Association, Keynote address: Alternatives to solving interoperability problems in Land Mobile Radios, Lake Tahoe, NV, July 2005.

Museum of Science Lecture Series: The Next Generation of Information and Communications Technologies-What does the Future Hold?, William R. Michalson and Brian King, January 14, 2004.

Agilent Wireless Technology Summit: Dynamic Node Location in an Ad Hoc Indoor Communications and Positioning Network, William R. Michalson, January 27, 2004

Security & Technology Online (SATO) Security Leadership Council. Panel discussion on Smart Surveillance, Command and Control, Oct 28-29, 2004.

"*Worcester Polytechnic Institute Barcelona Summit: The Future of Information Technology*," delivered presentation entitled "Personal Navigation in the Information Age," Apr 2001.

4. Projects advised (undergraduate).

4.1 Major Qualifying Projects (current)

[1] *Voice Release System,* B. Waldron, WZM-MQP-1M10, in process.

4.2 Major Qualifying Projects (completed)

[2] *Aeacus*, N. Anderson, D. Praetorius and C. Roddy, co-advised with S. Nestinger, 2011.

[3] *Realization of Performance Advancements for WPI's UGV-Prometheus*, M. Akmanalp, R. Doherty, J. Gorges, P. Kalauskas, E. Peterson and F. Polido, co-advised with T. Padir, S. Nestinger, M. Ciraldi, K. Stafford, 2011.

[4] *Autonomous Underwater Vehicle*, J. Baker, C. Frumento, J. Grzyb and T. North, co-advised w/I. Hussein, 2011.

[1] *Tactical Vest*, V. Brisian, J. Fernando, A. Khandaker and J. Zorrilla DeLos Santos, 2011.

[2] *Marsupial AUV*, N. Smith, B. Berard and C. Pietre, Lincoln Laboratory Project Center, co-advised with G. Heiniman, 2010.

[3] *Voice Release System,* J. Low, WZM-MQP-1M10, 2010.

[4] *Design and Realization of an Intelligent Unmanned Ground Vehicle*, J. Barrett, B. Roy and D. Sacco, Co-Advised w/T. Padir, 2010.

[5] *Accurate Real-Time Audio Circuit Simulation*, B. Gleason, WZM-MB09, 2010.

[6] *Optimization and Control Design of an Autonomous Underwater Vehicle*, D. Moussette, A. Palooparambil, and J. Raymond, AE- IIH-0003, co-advised w/I. Hussein, 2010.

[7] *Design of Autonomous Underwater Vehicle and Optimization of Hydrodynamic Properties and Control*, R. David, WZM-3A08, 2009.

[8] *Robotic Bass Player, B. Kosherick*, M. Brown, and A. Teti, WZM-RB08, 2008.

[9] *Public Safety Radio System*, P. Lucia, I. Levin and M. Barone, WZM-1A08, 2008.

[10] *Aircraft Lasercom Terminal Compact Optical Module*, B. Scoville and S. Rose, Lincoln Laboratory Project Center, WZM-2A08, 2008.

[11] *GPS Attitude Determination System*, J.P. Salmon, Michael LaBossiere and Mark Minotaur, 2005.

[12] *FPGA-Based VHF Modem With Integrated Testability*, Andrew Dupont and Jack Coyne, 2005.

[13] *TMR Computer System*, Maulin Patel, Omar Moussa and Matthew Kwiatkowski, 2005.

[14] *GPS Signal Generator*, Tim Coffey, 2005.

[15] *Dipole Antenna Placement in a Falcon-20 Aircraft*, Emily Anesta and David Plourde, Lincoln Laboratories Project Center, A-Term, 2004.

[16] *GPS Attitude Determination System*, Joshua Holwell, Himanshu Agrawal and Andrew Coonradt, 2004.

[17] *TMR Computer System*, Ryan Angilly, Mitch Lauer and Dan Debiasio, 2004.

[18] *Personal Inertial Navigation System*, Jason DeChiaro and Christopher Strus, 2003.

[19] WZM-MQP-4A02: *PC I/O in High Stress Environments*, John Niesz and James Kent, 2003.

[20] WZM-MQP-2A02: *Vacuum Tube Amplifier*, Joseph Kambourakis and Gregory Molnar, 2003.

[21] *Container Tracking System*, Victoria Chaplick, 2003.

[22] WZM-MQP-2A03: *Heat Management System for PCs*, Ernest Cardin, Kevin Candiloro and Stephen Leavey, 2002.

[23] WZM-MQP-1313: *Digital Image Enhancement*, Julie Bolduc, Joeseph Perry, Wei Fu, 2002.

[24] WZM-MQP-2A01: *Synchronized Audio Sample Looper*, Joel Gottshalk, Robert Conrad and Sanford Freedman, 2002.

[25] WZM-MQP-1A01: *Springboard Digital Multimeter*, Pavel Loven and Andrew Young, 2002.

[26] *Ballistic Missile Defense Analysis Toolkit*, Winfield Peterson, Doug Tilkin and Benjamin Wilson, Lincoln Laboratories Project Center, 2002.

[27] HU-FB-CS01, *C Sound Synthesizer*, Peter W. DeBonte (co-advised).

[28] WZM-MQP-1A00: *StrongArm-Based Computer System*, Bradford Snow, 2001.

[29] WZM-MQP-1C01: *PC Controlled Laser Light Show Device*, Joel Smith, 2001.

[30] WZM-MQP-1E00: *PIC-based MIDI Sequencer* Malcolm Beaulieu, 2001.

[31] WZM-MQP-2A00: *Automotive PC Development Platform* Travis Pouliot and David Philips, 2001.

[32] *The RoadCom Automotive Computing System*, Benjamin Kennedy and John Pong, 2000.

[33] WZM-MQP-1A99: *Compressed Sample Wavetable Synthesizer*, Justin Brzozoski, 2000.

[34] WZM-MQP-3499, *Automatically Equalizing Monitor*, Fernando Braghin, Tenzin Lama, Rahul Bhan and Dion Soetadi, 2000.

[35] 99D163M: *Railroad Communication*, Benjamin Richards, 2000.

[36] CS-MXC-IE00: *PIC Real-Time Sequencer*, Alexander Goodrich, 2000.

[37] 99D514M: *Design of a Microphone Preamplifier*, Eric Reuter, 2000.

[38] WZM-MQP-1A99, *MPEG Audio Deck*, Justin Brzozoski, 2000.

[39] *Digital Image Enhancement*, Julie Bolduc, Wei Fu and Joseph Perry, 2000.

[40] WZM-MQP-4A98, *Railroad Communications System*, Matthew Lug, 1999.

[41] 99D078M: *Modular Effects Processor II*, Erik Neyland, 1999.

[42] 99D176M: *Portable Digital Audio Recorder* Eric Toledo and Duc Truong, 1999.

[43] EE-WZM-1A97, *C Sound Synthesizer*, Ross E. Borgeson, Michael W. Hamel and Matthew S. Walsh, 1998.

[44] EE-WZM-4A97, *Firewire Audio Device*, Daniel R. Stutzbach, 1998.

[45] EE-WZM-2A97, *Modular Effects Processor*, Michael J. Dellisanti, 1998.

[46] EE-WZM-3A97, *GPS Personal Navigation*, Jeffery A. Alderson and Helder Machado, 1998.

[47] HU-FB-CS01, *C Sound Synthesizer*, Peter W. DeBonte (co-advised).

[48] EE-WZM-1E97, *PM Measurement System*, Yevgeniy Bogdanov.

[49] EE-REL-C008, *Design and Development of a Microprocessor-Based Gaussmeter*, David M. Burnham.

[50] EE-WZM-RC01, *Acoustic Guitar Amplifier*, Christopher Thomas.

[51] EE-WZM-GSD1, *Guitar Sustaining Device*, Paul D'Ambra.

[52] EE-RXV-5260, *Audio Feedback Elimination System*, Ross D. Pease and John R. Pelliccio.

[53] EE-RJD-M963, *Embedded Systems Design*, Christopher A. Briggs and Anthony J. Viapiano.

[54] EE-WHE-9601, *GPS Hazard Detector*, Michael Roberts, William Cidela, and Chris Mangiarelli.

[55] EE-WZM-2C96, *Flexible Synthesis*, Noah T. Vawter and Luke Demoracski.

[56] EE-WZM-1A96, *Tap Dancer MIDI Interface*, Thomas Trela and William Dowell.

[57] EE-WZM-2A96, *GPS Hazard Detector II*, Will Brothers, Jon Day, and John Zaghi.

[58] EE-WZM-3A96, *Loudspeaker Data Acquisition System*, Adam Gross.

[59] EE-WZM-4A96, *Audio Morphing Processor*, William Butterfield and Ted Phipps.

[60] EE-WZM-5A96, *Acoustic Modeling*, Peter DeBonte.

[61] EE-WZM-1B96, *Distributed Audio Controller*, Stephen S. Richardson.

[62] EE-WZM-1A95, *Acoustic Hazard Meter*, Ronald D. Slack.

[63] EE-WZM-2A95, *Forest Service DGPS*, Joshua J. Single and Michael T. Spadazzi.

[64] EE-WZM-1C95, *Audio to MIDI Converter*, Jennifer R. Principe.

[65] EE-WZM-1D95, *Low Cost Auralizer*, Jason R. Hills and Mark R. Paulson.

[66] EE-WZM-3A95, *Passive Radiator Design*, Kevin R. Weldon.

[67] EE-WZM-1C94, *Char Model Generation*, Colin J. Florendo.

[68] EE-WZM-1D94, *Wide Area DGPS Simulator*, Daniel Cohen and Robert Schroter.

[69] EE-WZM-2D94, *Digital Soundcard*, Timothy Alsberg (Russian Project Center).

[70] EE-WZM-3D94, *Digital Univibe*, Andrew Willis and Daniel Toohey.

[71] EE-WZM-1A94, *DSP Based Real-Time Audio Feedback Eliminator*, Kevin M. Eddy.

[72] EE-WZM-2A94, *Digital LCD Oscilloscope*, William F. Brown and John F. Ebersole.

[73] 93D236M, *MIDI Mapper*, Jonathan Kemble and Brian Candiloro.

[74] EE-WZM-1C93, *Fault-Tolerant Computer*, Frederick N. Parmenter.

[75] EE-WZM-1A93, *Wireless MIDI Controller*, Sanjay Raja, Charles Cimalore, Ty Panagoplos.

[76] EE-WZM-2A93, *Multiple Pitch Detector*, Jeanne A. Sawtelle.

[77] EE-WZM-3A93, *Multiprocessor Cache Coherence*, Lauren C. Lind and Norman E. Rhodes.

[78] EE-WZM-1C92, *A Simulation of the DLX Architecture*, Lisa Harlow.

[79] EE-WZM-2C92, *A New Microprocessor Development System*, Gregory B. Burlingame, David J. Fortin, Kevin S. Pearson.

[80] EE-WZM-1A92, *Digital Audio Sampler*, Roger D. Gagnon and James M. Lach.

[81] EE-WZM-2A92, *Intelligent Harmonizer*, Prabhjot S. Anand and Aftab M. Yusuf.

[82] EE-WZM-3A92, *Computerized Audio Mixer*, Richard J. Wood.

[83] EE-WZM-1B92, *Real-Time Harmonizer*, Mohiuddin M. Kahn.

[84] EE-WZM-1A91, *Residue Number System Processor*, Ravdeep S. Anand and Christine A. Easton.

[85] EE-WZM-2A91, *SCSI Bus Analyzer*, Brian Costello, George Delouriero, Matthew Maguire, and Keith Nevins.

4.3 Graduate Theses Advised and Co-Advised

4.3.1 MS Theses (current)

[1] No Current MS Students

4.3.2 MS Theses (completed)

[1] Morin, Russell, "A Novel Localization System For Experimental Autonomous Underwater Vehicles," MS Thesis, co-advisor, Worcester Polytechnic Institute, 2010.

[2] Navalekar, Abhijit, "Design of an OFDM-Based VHF Modem," MS Thesis, primary advisor, Worcester Polytechnic Institute.

[3] Ahlehagh, Hasti, "Techniques for Communications and Geolocation Using Wireless Ad Hoc Networks," MS Thesis, primary advisor, Worcester Polytechnic Institute, 2004.

[4] Sebastian, Dalys, "Development of a Field-Deployable Ultrasound Scanner System," MS Thesis, co-advisor, Worcester Polytechnic Institute, 2004.

[5] Tobgay, Sonam, "Novel Concepts for RF Surface Coils with Integrated Receivers," MS Thesis, co-advisor, Worcester Polytechnic Institute, 2004.

[6] Breen, Daniel, "Characterization of Multi-Carrier Locator Performance," MS Thesis, co-advisor, 2004.

[7] Aghogho, Obi, "A Novel Radio Frequency Coil Design for Breast Cancer Screening in a Magnetic Resonance Imaging System," MS Thesis, co-advisor, Worcester Polytechnic Institute, 2003.

[8] Fei, Ming, "Electromagnetic Detection, Infrared Visualization and Image Processing Techniques for Non-Metallic Inclusions in Molten Aluminum," MS Thesis, co-advisor, 2002.

[9] Lavoie, Bruce, "Design and Implementation of an N-Channel Self Calibrating Audio System," MS Thesis, primary advisor, Worcester Polytechnic Institute, 2000.

[10] Bogdonov, Gene, "Theoretical and Practical Implementation of Electrical Impedance Material Inspection of Powder Metallurgy Compacts," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1999.

[11] Messier, Andrew, "Modeling the Effects of Terrain Masking on GPS Accuracy and Integrity," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1998.

[12] Antonescu, Bogdan, "Elliptic Curve Cryptosystems on Embedded Microprocessors," Bogdan Antonescu, MS Thesis, co-advisor, Worcester Polytechnic Institute, 1998.

[13] Lai, Qiang, "Ground-Penetrating Radar Data Processing System," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1998.

[14] Soria-Rodríguez, Pedro, "Multicast-Based Interactive-Group Object-Replication For Fault Tolerance," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1998.

[15] Hoy, William, "Audio Signal Denoising Using Wavelets," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1997.

[16] Progri, Ilir, "Harmonic Flow Monitoring by means of Global Positioning System," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1997.

[17] Bretchko, Pavel, "Pulsed Hysteresis Graph System," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1997.

[18] Repkin, Dmitry V., "A Hierarchical Neural Network Based Data Processing System for Ground Penetrating Radar," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1997.

[19] Metsis, Sophocles, "Design of a Real-Time Capable, Fault-Tolerant, Distributed System," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1996.

[20] Hill, Jonathan, "Efficient Implementation of Mesh Generation and FDTD Simulation of Electromagnetic Fields," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1996.

[21] Dunkelberg, John, "FEM Mesh Mapping to a SIMD Machine Using Genetic Algorithms," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1996.

[22] Leuenberger, Georg, "Design and Development of a Microprocessor Based Gauss Meter," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1995.

[23] Valentino, Ralph, "DISC: A Dynamic Instruction Set Coprocessor," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1995.

[24] Muley, Aalok, "A Fault Tolerant Network for a Real-Time Environment," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1994.

[25] Mohan, Surrender, "Automatic Surface Mesh Generation for 3D Solid Models Using Delaunay Algorithm," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1994.

[26] Petrangelo, John, "Experimental Preconditioners for Large Dense Systems," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1994.

[27] Schneider, Eric, "Design, Simulation, and Analysis of a 3D Integrated Optical Computer," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1993.

[28] Palmer, Bradley, "A Comparison of Three Protocols Supporting Time-Dependent and Time-Independent Communications," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1992.

[29] Clayton, Shawn, "An Analysis of the Real-Time Behavior of Galactica Net," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1992.

[30] Levergood, Thomas, "An Experimental Evaluation of Split User/Supervisor Cache Memories," MS Thesis, primary advisor, Worcester Polytechnic Institute, 1992.

[31] Lavalee, James, "The Design and Development of Real-Time Systems Using Ada and the Activation Framework," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1992.

[32] Velazques, Javier, "The Development of a Real-Time Environment Using the Activation Framework," MS Thesis, co-advisor, Worcester Polytechnic Institute, 1992.

4.3.3 Ph. D. Dissertations (current)

[1] Jitesh, "Ad-Hoc Networking for Bandwith Limited LMR Systems," primary advisor.

4.3.4 Ph. D. Dissertations (completed)

[1] Iyer, Vishwanath, "Broadband Impedance Matching of Antenna Radiators," Ph.D. Dissertation, co-advisor, Worcester Polytechnic Institute, 2010.

[2] Navalekar, Abhijit, "Distributed Digital Radios For Land Mobile Radio Applications," Ph.D. Dissertation, primary advisor, Worcester Polytechnic Institute, 2009.

[3] Parikh, Hemish, "Design of an OFDM Transmitter and Receiver for Precision Personnel Location," primary advisor.

[4] Progri, Ilir, "An Assessment of Indoor Geolocation Systems," Ph.D. Dissertation, primary advisor, Worcester Polytechnic Institute, 2003.

[5] Li, Xinrong, "Super-Resolution TOA Estimation with Diversity Techniques for Indoor Applications," Ph.D. Dissertation, co-advisor, Worcester Polytechnic Institute, 2003.

[6] Leuenberger, Gerog H. W., "Electrostatic Density Measurements in Green-State PM Parts," Ph.D. Dissertation, co-advisor, Worcester Polytechnic Institute, 2003.

[7] Bogdanov, Gene, "Radio-Frequency Coil Design for High Field Magnetic Resonance Imaging," Ph.D. Dissertation, co-advisor, Worcester Polytechnic Institute, 2002.

[8] Elbirt, Adam J., "Reconfigurable Computing for Symmetric-Key Algorithms," Ph.D. Dissertation, co-advisor, Worcester Polytechnic Institute, 2002.

[9] Bretchko, Pavel, "Design and Development of Ultra-wideband DC-Coupled Amplifier," Ph.D. Dissertation, co-advisor, Worcester Polytechnic Institute, 2001.

[10] Hill, Jonathan, "Development of an Experimental Global Positioning System (GPS) Receiver Platform for Navigation Algorithm Evaluation," Ph.D. Dissertation, primary advisor, 2001.

[11] Spasojević, Mirko, "Creation of Sparse Boundary Element Matricies for 2-D and Axi-symmetric Electrostatic Problems Using a Bi-orthogonal Wavelet," Ph.D. Dissertation, co-advisor, Worcester Polytechnic Institute, 1997.

[12] Shi, Funan, "Optimal Designs of Gradient and RF Coils for Magnetic Resonance Imaging (MRI) Instrument," Ph.D. Dissertation, co-advisor, Worcester Polytechnic Institute, 1996.

5. Proposals and Funding (past 5 years):

5.1 In Review

$ 199,996 A National Model Robotics Curriculum, NSF (PI: Dr. M. Gennert, Co-PIs: Drs. T. Padir, W.R. Michalson, G. Fischer and C. Demetry), May 2009.

$ 199,052 A National Model Robotics Capstone, NSF (PI: Dr. W.R. Michalson, Co-PIs: Drs.T. Padir, C. Demetry, G. Tryggvason and F. Looft), May 2009.

$ 399,791 Modular System for Teaching Robotics Engineering (MySTRE), NSF (PI: Dr. G. Fischer, Co-PIs: Drs. W.R. Michalson and T. Padir), March 2009.

5.2 Funding Received

$ 50,524 PCGO Broadband Modem, Powerwave Technologies, Inc., (PI: Dr. W.R. Michalson), May 2009.

$ 1,245,000 Real-Time Troop Status Monitoring System, US Army Telemedicine and Advanced Technology Research Center. (PI: Dr. Peder Pedersen, Co-PIs: Drs. William R. Michalson and Yitzhak Mendelson). Third year of funding. Projected funding period: Oct 1, 2004 to Sep 30, 2005.

$ 148,422 Precision Personnel Locator System, National Institute of Justice (PI: Dr. John Orr, Co-PIs: Drs. David Cyganski and William R. Michalson). Second year of funding. Projected funding period: Sep 1, 2004 to Oct 31, 2005. Grant code 219240.

$ 74,048 High-Speed VHF Modem, US Army Telemedicine and Advanced Technology Research Center (PI: Dr. William R. Michalson). Funding period: Mar 1, 2004 to Dec 31, 2004. Grant code 214370.

$ 81,499 WPI Nanosat Program, Air Force Office of Scientific Research (PI: Dr. Fred Looft, Co-PIs: Drs. William R. Michalson and Diran Apelian). Funding period: Apr 1, 2003 to Mar 31, 2005. Grant code 214400.

$ 996,000 Precision Personnel Locator System, National Institute of Justice (PI: Dr. David Cyganski, Co-PIs: Drs. William R. Michalson and John Orr). Second year of funding. Projected funding period: Sep 1, 2004 to Aug 31, 2005. Grant code 219240.

$ 813,141 Real-Time Troop Status Monitoring System, US Army Telemedicine and Advanced Technology Research Center, (PI: Dr. William R. Michalson, Co-PIs: Drs. Peder Pedersen and Yitzhak Mendelson). Second year of funding. Funding period Oct 1, 2003 to Sep 30, 2004. Grant code 214370.

6. Honors, Awards, and Recognitions:

Elected Senior Member of the IEEE.

Joseph Samuel Satin Distinguished Fellowship awarded for the 1994-1995 academic year.

Aldo Miccioli Fellowship recipient from Raytheon Equipment Division.

ION Best Paper Award - GPS-96 for W. R. Michalson, W. Cidela, et. al., "A GPS-Based Hazard Detection and Warning System," in review, " *ION GPS-96, 9th International Meeting of the Satellite Division of the Institute of Navigation*, pp. 167-175, Kansas City, MO, Sep 17-20, 1996

2nd Place - 2004 ECE Department MQP Award / Provost's MQP Award for *GPS-Based Orbit and Attitude Determination System for PANSAT*, Joshua Holwell, Andrew Coonradt and Himanshu Agrawal.

1st Place - 2003 ECE Department MQP Award / Provost's MQP Award for *Personal Inertial Navigation System*, Jason DeChiaro and Chris Struus.

1st Place - 2002 ECE Department MQP Award / Provost's MQP Award for *Handspring Digital Voltmeter*, Andrew Young and Pavel Loven.

3rd Place - 1998 ECE Department MQP Award / Provost's MQP Award for *Design of a Personal Handheld GPS Receiver*, Jeffery Alderson and Helder Machado.

2nd Place - 1997 ECE Department MQP Award / Provost's MQP Award for *Distributed Audio Controller*, EE-WZM-1B96, Stephen S. Richardson.

3rd Place - 1997 ECE Department MQP Award / Provost's MQP Award for *GPS Hazard Detector II*, EE-WZM-2A96, Will Brothers, Jon Day, and John Zaghi.

1st Place - 1996 ECE Department MQP Award / Provost's MQP Award for *GPS Hazard Detector*, EE-WHE-9601, Michael Roberts, William Cidela, and Chris Mangiarelli.

6.1 Memberships and offices held in professional society

Institute of Electrical and Electronic Engineers, Senior Member

Institute of Navigation

Royal Institute of Navigation

American Society of Engineering Educators

6.2 Professional Service

Massachusetts Board of Bar Overseers Hearing Committee Member, 2010-Present.

Steering Committee – 2009 First Annual Robotics Innovations Competition and Conference (RICC '09), Nov 7-8, Worcester, MA, 2009.

Conference Technical Co-Chair – 2009 IEEE International Conference on Technologies of Practical Robot Applications (TePRA 2009), Nov. 9-11, Woburn, MA, 2009.

Reviewer – Proposal number CRDPJ 379622-08, Natural Sciences and Engineering Research Council of Canada (NSERC), Mar. 2009

Co-Chair – Urban and Indoor Geolocation, Institute of Navigation International Technical Meeting (ITM2009), Anaheim CA, Jan. 2009.

Reviewer – Proposal number CRDPJ 379622-08, Natural Sciences and Engineering Research Council of Canada (NSERC), Mar. 2009

APPENDIX B

The Design and Analysis of Spatial Data Structures

The Design and Analysis of Spatial Data Structures

Hanan Samet
UNIVERSITY OF MARYLAND

ADDISON - WESLEY PUBLISHING COMPANY, INC.
Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam
Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan

This book is in the Addison-Wesley Series in Computer Science
Michael A. Harrison: Consulting Editor

Library of Congress Cataloging-in-Publication Data

Samet, Hanan.
The Design and analysis of spatial data structures/by Hanan Samet.
p. cm.
Bibliography: p.
Includes index.
ISBN 0-201-50255-0
1. Data structures (Computer science) 2. Computer graphics.
I. Title.
QA76.9.D35S26 1989 89-30382
005.7'3—dc19 CIP

Reprinted with corrections January, 1994

4 5 6 7 8 9 10 11 12 13 14-MA-97 96 95 94

Credits:
Thor Bestul created the cover art.

Gyuri Fekete generated Figure 1.16; Daniel DeMenthon, Figures 1.20, 1.21, and 1.23; Jiang-Hsing Chu, Figures 2.48 and 2.52; and Walid Aref, Figures 4.38 through 4.40.

Figures 1.1, 4.9, and 4.10 are from H. Samet and R. E. Webber, On encoding boundaries with quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 6*, 3 (May 1984), 365–369. © 1984 IEEE. Reprinted by permission of IEEE.

Figures 1.2, 1.3, 1.5 through 1.10, 1.12, 1.14, 1.25, 1.26, 2.3, 2.4, 2.18, 2.20, 2.30, 2.32, 2.53, 2.54, 2.57, 2.58, 3.20, 3.21, 4.1 through 4.5, 4.7, 4.8, 4.11, and 5.2 are from H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys 16*, 2 (June 1984), 187–260. Reprinted by permission of ACM.

Figures 1.4 and 5.6 are from H. Samet and R. E. Webber, Hierarchical data structures and algorithms for computer graphics. Part I. Fundamentals, *IEEE Computer Graphics and Applications 8*, 3 (May 1988), 48–68. © 1988 IEEE. Reprinted by permission of IEEE.

Figure 1.30 is from M. Li, W. I. Grosky, and R. Jain, Normalized quadtrees with respect to translations, *Computer Graphics and Image Processing 20*, 1 (September 1982), 72–81. Reprinted by permission of Academic Press.

Figures 2.7 and 2.10 through 2.15 are from H. Samet, Deletion in two-dimensional quad trees, *Communications of the ACM 23*, 12 (December 1980), 703–710. Reprinted by permission of ACM.

Figures 2.26 and 2.27 are from D. T. Lee and C. K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and quad trees, *Acta Informatica 9*, 1 (1977), 23–29. Reprinted by permission of Springer Verlag.
Continued on p. 493

To my parents, Julius and Lotte

PREFACE

Spatial data consist of points, lines, rectangles, regions, surfaces, and volumes. The representation of such data is becoming increasingly important in applications in computer graphics, computer vision, database management systems, computer-aided design, solid modeling, robotics, geographic information systems (GIS), image processing, computational geometry, pattern recognition, and other areas. Once an application has been specified, it is common for the spatial data types to be more precise. For example, consider a geographic information system (GIS). In such a case, line data are differentiated on the basis of whether the lines are isolated (e.g., earthquake faults), elements of tree-like structures (e.g., rivers and their tributaries), or elements of networks (e.g., rail and highway systems). Similarly region data are often in the form of polygons that are isolated (e.g., lakes), adjacent (e.g., nations), or nested (e.g., contours). Clearly the variations are large.

Many of the data structures currently used to represent spatial data are hierarchical. They are based on the principle of recursive decomposition (similar to *divide and conquer* methods [Aho74]). One such data structure is the quadtree (octree in three dimensions). As we shall see, the term *quadtree* has taken on a generic meaning. In this book, it is my goal to show how a number of hierarchical data structures used in different domains are related to each other and to quadtrees. My presentation concentrates on these different representations and illustrates how a number of basic operations that use them are performed.

Hierarchical data structures are useful because of their ability to focus on the interesting subsets of the data. This focusing results in an efficient representation and in improved execution times. Thus they are particularly convenient for performing set operations. Many of the operations described can often be performed as efficiently, or more so, with other data structures. Nevertheless hierarchical data structures are attractive because of their conceptual clarity and ease of implementation. In addition, the use of some of them provides a spatial index. This is very useful in applications involving spatial databases.

As an example of the type of problems to which the techniques described in this book are applicable, consider a cartographic database consisting of a number of maps and some typical queries. The database contains a contour map, say at 50-foot elevation intervals, and a land use map classifying areas according to crop growth. Our goal is to determine all regions between 400- and 600-foot elevation levels where wheat is grown. This will require an intersection operation on the two maps. Such an analysis could be rather costly, depending on the way the maps are represented. For example, since areas where corn is grown are of no interest, we wish to spend a minimal amount of effort searching such regions. Yet traditional region representations such as the boundary code [Free74] are very local in application, making it difficult to avoid examining a corn-growing area that meets the desired elevation criterion. In contrast, hierarchical representations such as the region quadtree are more global in nature and enable the elimination of larger areas from consideration.

Another query might be to determine whether two roads intersect within a given area. We could check them point by point; however, a more efficient method of analysis would be to represent them by a hierarchical sequence of enclosing rectangles and to discover whether in fact the rectangles do overlap. If they do not, the search is terminated. If an intersection is possible, more work may have to be done, depending on which method of representation is used.

A similar query can be constructed for point data — for example, to determine all cities within 50 miles of St. Louis that have a population in excess of 20,000. Again we could check each city individually. However, using a representation that decomposes the United States into square areas having sides of length 100 miles would mean that at most four squares need to be examined. Thus California and its adjacent states can be safely ignored.

Finally, suppose we wish to integrate our queries over a database containing many different types of data (e.g., points, lines, areas). A typical query might be, "Find all cities with a population in excess of 5,000 people in wheat-growing regions within 20 miles of the Mississippi River." In this book we will present a number of different ways of representing data so that such queries and other operations can be efficiently processed.

This book is organized as follows. There is one chapter for each spatial data type, in which I present a number of different data structures. The aim is to gain the ability to evaluate them and to determine their applicability. Two problems are treated in great detail: the rectangle intersection problem, discussed in the context of the representation of collections of small rectangles (Chapter 3), and the point location problem, discussed in the context of the representation of curvilinear data (Chapter 4). A comprehensive treatment of the use of quadtrees and octrees in other applications in computer graphics, image processing, and geographic information systems (GIS) can be found in [Same90b].

Chapter 1 gives a general introduction to the principle of recursive decomposition with a concentration on two-dimensional regions. Key properties, as well as a historical overview, are presented.

Chapter 2 discusses hierarchical representations of multidimensional point data. These data structures are particularly useful in applications in database management systems because they are designed to facilitate responses to search queries.

Chapter 3 examines the hierarchical representation of collections of small rectangles. Such data arise in applications in computational geometry, very large-scale integrations (VLSI), cartography, and database management. Examples from these fields (e.g., the rectangle intersection problem) are used to illustrate their differences. Many of the representations are closely related to those used for point data. This chapter is an expansion of [Same88a].

Chapter 4 treats the hierarchical representation of curvilinear data. The primary focus is on the representation of polygonal maps. The goal is to be able to solve the point location problem. Quadtree-like solutions are compared with those from computational geometry such as the K-structure [Kirk83] and the layered dag [Edel86a].

Chapter 5 looks at the representation of three-dimensional region data. In this case, a number of octree variants are examined, as well as constructive solid geometry (CSG) and the boundary model (BRep). Algorithms are discussed for converting between some of these representations. The representation of surfaces (i.e., 2.5-dimensional data) is also briefly discussed in this chapter.

There are a number of topics for which justice requires a considerably more detailed treatment. However, due to space limitations, I have omitted a detailed discussion of them and instead refer interested readers to the appropriate literature. For example, surface representations are discussed briefly with three-dimensional data in Chapter 5 (also see Chapter 7 of [Same90b]). The notion of a pyramid is presented only at a cursory level in Chapter 1 so that it can be contrasted with the quadtree. In particular, the pyramid is a multiresolution representation, whereas the quadtree is a variable resolution representation. Readers are referred to Tanimoto and Klinger [Tani80] and the collection of papers edited by Rosenfeld [Rose83a] for a more comprehensive exposition on pyramids.

Results from computational geometry, although related to many of the topics covered in this book, are discussed only in the context of representations for collections of small rectangles (Chapter 3) and curvilinear data (Chapter 4). For more details on early work involving some of these and related topics, interested readers should consult the surveys by Bentley and Friedman [Bent79b], Overmars [Over88a], Edelsbrunner [Edel84], Nagy and Wagle [Nagy79], Peuquet [Peuq84], Requicha [Requ80], Srihari [Srih81], Samet and Rosenfeld [Same80d], Samet [Same84b, Same88a], Samet and Webber [Same88c, Same88d], and Toussaint [Tous80].

There are also a number of excellent texts containing material related to the topics that I cover. Rosenfeld and Kak [Rose82a] should be consulted for an encyclopedic treatment of image processing. Mäntylä [Mänt87] has written a comprehensive introduction to solid modeling. Burrough [Burr86] provides a survey of geographic information systems (GIS). Overmars [Over83] has produced a particularly good treatment of multidimensional point data. In a similar vein, see Mehlhorn's [Mehl84] unified treatment of multidimensional searching and computational geometry. For thorough introductions to computational geometry, see Preparata and

K-structure and the layered dag in Section 4.3 are relevant to computational geometry. Bucket methods such as linear hashing, spiral hashing, grid file, and EXCELL, in Section 2.8, and R-trees in Section 3.5.3 are important in the study of database management systems. Methods for multidimensional searching that are discussed include k–d trees in Section 2.4, range trees and priority search trees in Section 2.5, and point-based rectangle representations in Section 3.4. The discussions of the representation of two-dimensional regions in Chapter 1, polygonal representations in Chapter 4, and use of point methods for focussing the Hough Transform are relevant to image processing. Finally the rectangle-representation methods and plane-sweep methods of Chapter 3 are important in the field of VLSI design.

The natural home for courses that use this book is in a computer science department, but the book could also be used in a curriculum in geographic information systems (GIS). Such a course is offered in geography departments. The emphasis for a course in this area would be on the use of quadtree-like methods for representing spatial data.

Shamos [Prep85] and Edelsbrunner [Edel87] (also see [Prep83, ORou88]). A broader view of the literature can be found in related bibliographies such as the ongoing collective effort coordinated by Edelsbrunner [Edel83c, Edel88], and Rosenfeld's annual collection of references in the journal *Computer Vision, Graphics, and Image Processing* (e.g., [Rose88]).

Nevertheless, given the broad and rapidly expanding nature of the field, I am bound to have omitted significant concepts and references. In addition at times I devote a disproportionate amount of attention to some concepts at the expense of others. This is principally for expository purposes; I feel that it is better to understand some structures well rather than to give readers a quick runthrough of buzzwords. For these indiscretions, I beg your pardon and hope you nevertheless bear with me.

My approach is an algorithmic one. Whenever possible, I have tried to motivate critical steps in the algorithms by a liberal use of examples. I feel that it is of paramount importance for readers to see the ease with which the representations can be implemented and used. In each chapter, except for the introduction (Chapter 1), I give at least one detailed algorithm using pseudo-code so that readers can see how the ideas can be applied. The pseudo-code is a variant of the ALGOL [Naur60] programming language that has a data structuring facility incorporating pointers and record structures. Recursion is used heavily. This language has similarities to C [Kern78], PASCAL [Jens74], SAIL [Reis76], and ALGOL W [Baue68]. Its basic features are described in the Appendix. However, the actual code is not crucial to understanding the techniques, and it may be skipped on a first reading. The index indicates the page numbers where the code for each algorithm is found.

In many cases I also give an analysis of the space and time requirements of different data structures and algorithms. The analysis is usually of an asymptotic nature and is in terms of *big O* and Ω notation [Knut76]. The *big O* notation denotes an upper bound. For example, if an algorithm takes $O(\log_2 N)$ time, then its worst-case behavior is never any worse than $\log_2 N$. The Ω notation denotes a lower bound. As an example of its use, consider the problem of sorting N numbers. When we say that sorting is $\Omega(N \cdot \log_2 N)$ we mean that given any algorithm for sorting, there is some set of N input values for which the algorithm will require at least this much time.

At times I also describe implementations of some of the data structures for the purpose of comparison. In such cases counts, such as the number of fields in a record, are often given. These numbers are meant only to amplify the discussion. They are not to be taken literally, as improvements are always possible once a specific application is analyzed more carefully.

Each chapter contains a substantial number of exercises. Many of the exercises develop further the material in the text as a means of testing the reader's understanding, as well as suggesting future directions. When the exercise or its solution is not my own, I have preceded it with the name of its originator. The exercises have not been graded by difficulty. They rarely require any mathematical skills beyond the undergraduate level for their solution. However, while some of the exercises are quite straightforward, others require some ingenuity. Solutions, or references to papers that

contain the solution, are provided for a substantial number of the exercises that do not require programming. Readers are cautioned to try to solve the exercises before turning to the solutions. It is my belief that much can be learned this way (for the student and, even more so, for the author). The motivation for undertaking this task was my wonderful experience on my first encounter with the rich work on data structures by Knuth [Knut73a, Knut73b].

An extensive bibliography is provided. It contains entries for both this book and the companion text [Same90b]. Not all of the references that appear in the bibliography are cited in the two texts. They are retained for the purpose of giving readers the ability to access the entire body of literature relevant to the topics discussed in them. Each reference is annotated with a key word(s) and a list of the numbers of the sections in which it is cited in either of the texts (including exercises and solutions). In addition, a name and credit index is provided that indicates the page numbers in this book on which each author's work is cited or a credit is made.

ACKNOWLEDGMENTS

Over the years I have received help from many people, and I am extremely grateful to them. In particular Robert E. Webber, Markku Tamminen, and Michael B. Dillencourt have generously given me much of their time and have gone over critical parts of the book. I have drawn heavily on their knowledge of some of the topics covered here. I have also been extremely fortunate to work with Azriel Rosenfeld over the past ten years. His dedication and scholarship have been a true inspiration to me. I deeply cherish our association.

I was introduced to the field of spatial data structures by Gary D. Knott who asked "how to delete in point quadtrees." Azriel Rosenfeld and Charles R. Dyer provided much interaction in the initial phase of my research. Those discussions led to the discovery of the neighbor-finding principle. It is during that time that many of the basic conversion algorithms between quadtrees and other image representations were developed as well. I learned much about image processing and computer vision from them. Robert E. Webber taught me computer graphics, Markku Tamminen taught me solid modeling and representations for multiattribute data, and Michael B. Dillencourt taught me about computational geometry.

During the time that this book was written, my research was supported, in part, by the National Science Foundation, the Defense Mapping Agency, the Harry Diamond Laboratory, and the Bureau of the Census. In particular I would like to thank Richard Antony, Y. T. Chien, Su-shing Chen, Hank Cook, Phil Emmerman, Joe Rastatter, Alan Saalfeld, and Larry Tokarcik. I am appreciative of their support.

Many people helped me in the process of preparing the book for publication. Acknowledgments are due to Rene McDonald for coordinating the day-to-day matters

of getting the book out and copyediting; to Scott Carson, Emery Jou, and Jim Purtilo for TROFF assistance beyond the call of duty; to Marisa Antoy and Sergio Antoy for designing and implementing the algorithm formatter used to typeset the algorithms; to Barbara Burnett, Michael B. Dillencourt, and Sandra German for help with the index; to Jay Weber for setting up the TROFF macrofiles so that I can keep track of symbolic names and thus be able to move text around without worrying about the numbering of exercises, sections, and chapters; to Liz Allen for early TROFF help; to Nono Kusuma, Mark Stanley, and Joan Wright Hamilton for drawing the figures; to Richard Muntz and Gerald Estrin for providing temporary office space and computer access at UCLA; to Sandy German, Gwen Nelson, and Janet Salzman for help in initial typing of the manuscript; to S. S. Iyengar, Duane Marble, George Nagy, and Terry Smith who reviewed the book; and to Peter Gordon, John Remington, and Keith Wollman at Addison-Wesley Publishing Company for their encouragement and confidence in this project.

Aside from the individuals named above, I have also benefited from discussions with many other people over the past years. They have commented on various parts of the book and include Chuan-Heng Ang, Walid Aref, James Arvo, Harvey H. Atkinson, Thor Bestul, Sharat Chandran, Chiun-Hong Chien, Jiang-Hsing Chu, Leila De Floriani, Roger Eastman, Herbert Edelsbrunner, Claudio Esperanca, Christos Faloutsos, George (Gyuri) Fekete, Kikuo Fujimura, John Gannon, John Goldak, Erik Hoel, Liuqing Huang, Frederik W. Jansen, Ajay Kela, David Kirk, Per Åke Larson, Dani Lischinski, Don Meagher, David Mount, Randal C. Nelson, Glenn Pearson, Ron Sacks-Davis, Timos Sellis, Clifford A. Shaffer, Deepak Sherlekar, Li Tong, Brian Von Herzen, Peter Widmayer, and David Wise. I deeply appreciate their help.

A GUIDE TO THE INSTRUCTOR

This book can be used in a second data structures course, one with emphasis on the representation of spatial data. The focus is on the use of the principle of divide-and-conquer for which hierarchical data structures provide a good demonstration. Throughout the book both worst-case optimal methods and methods that work well in practice are emphasized in conformance with my view that the well-rounded computer scientist should be conversant with both types of algorithms. This material is more than can be covered in one semester; but the instructor can reduce it as necessary. For example, the detailed examples can be skipped or used as a basis of a term project or programming assignments.

The book can also be used to organize a course to be prerequisite to courses in computer graphics and solid modeling, computational geometry, database management systems, multidimensional searching, image processing, and VLSI design. The discussions of the representations of two-dimensional regions in Chapter 1, polygonal representations in Chapter 4, and most of Chapter 5 are relevant to computer graphics and solid modeling. The discussions of plane-sweep methods and their associated data structures such as segment trees, interval trees, and priority search trees in Sections 3.2 and 3.3 and point location and associated data structures such as the

CONTENTS

Preface **vii**

1 INTRODUCTION **1**
- 1.1 Basic Definitions 1
- 1.2 Overview of Quadtrees and Octrees 2
- 1.3 History of the Use of Quadtrees and Octrees 10
- 1.4 Space Decomposition Methods 16
  - 1.4.1 Polygonal Tilings 17
  - 1.4.2 Nonpolygonal Tilings 26
- 1.5 Space Requirements 32

2 POINT DATA **43**
- 2.1 Introduction 44
- 2.2 Nonhierarchical Data Structures 46
- 2.3 Point Quadtrees 48
  - 2.3.1 Insertion 49
  - 2.3.2 Deletion 54
  - 2.3.3 Search 64
- 2.4 k-d Trees 66
  - 2.4.1 Insertion 68
  - 2.4.2 Deletion 73
  - 2.4.3 Search 77
  - 2.4.4 Comparison with Point Quadtrees 80
- 2.5 Range Trees and Priority Search Trees 80
- 2.6 Region-based Quadtrees 85
  - 2.6.1 MX Quadtrees 86
  - 2.6.2 PR Quadtrees 92

2.6.3 Comparison of Point and Region-based Quadtrees 104
2.7 Bit Interleaving 105
2.8 Bucket Methods 110
2.8.1 Hierarchical Bucket Methods 111
2.8.2 Nonhierarchical Bucket Methods 116
2.8.2.1 Linear Hashing 117
2.8.2.2 Spiral Hashing 125
2.8.2.3 Grid File 135
2.8.2.4 EXCELL 141
2.9 Conclusion 147

3 COLLECTIONS OF SMALL RECTANGLES 153
3.1 Introduction 155
3.2 Plane-Sweep Methods and the Rectangle Intersection Problem 158
3.2.1 Segment Trees 160
3.2.2 Interval Trees 165
3.2.3 Priority Search Trees 171
3.2.4 Alternative Solutions and Related Problems 174
3.3 Plane-Sweep Methods and the Measure Problem 178
3.4 Point-based Methods 186
3.5 Area-based Methods 199
3.5.1 MX-CIF Quadtrees 200
3.5.1.1 Insertion 202
3.5.1.2 Deletion 206
3.5.1.3 Search 209
3.5.2 Multiple Quadtree Block Representations 213
3.5.3 R-trees 219

4 CURVILINEAR DATA 227
4.1 Strip Trees, Arc Trees, and BSPR 228
4.2 Methods Based on the Region Quadtree 235
4.2.1 Edge Quadtrees 235
4.2.2 Line Quadtrees 237
4.2.3 PM Quadtrees 239
4.2.3.1 The PM_1 Quadtree 240
4.2.3.2 The PM_2 Quadtree 257
4.2.3.3 The PM_3 Quadtree 261
4.2.3.4 PMR Quadtrees 264
4.2.3.5 Fragments 269
4.2.3.6 Maintaining Labels of Regions 275
4.2.4 Empirical Comparisons of the Different Representations 278
4.3 Methods Rooted in Computational Geometry 286
4.3.1 The K-structure 287

4.3.2 Separating Chains and Layered Dags 293
4.3.3 Comparison with PM Quadtrees 306
4.4 Conclusion 312

5 VOLUME DATA 315
5.1 Solid Modeling 316
5.2 Region Octrees 318
5.3 PM Octrees 326
5.4 Boundary Model (BRep) 331
5.5 Constructive Solid Geometry (CSG) 338
5.5.1 CSG Evaluation by Bintree Conversion 340
5.5.1.1 Algorithm for a Single Halfspace 341
5.5.1.2 Algorithm for a CSG Tree 346
5.5.1.3 Incorporation of the Time Dimension 355
5.5.2 PM-CSG Trees 360
5.6 Surface-based Object Representations 365
5.7 Prism Trees 370
5.8 Cone Trees 374

Solutions to Exercises 377
Appendix: Description of Pseudo-Code Language 411
References 415
Name and Credit Index 465
Subject Index 477

1 INTRODUCTION

There are numerous hierarchical data structuring techniques in use for representing spatial data. One commonly used technique is the quadtree, which has evolved from work in different fields. Thus it is natural that a number of adaptations of it exist for each spatial data type. Its development has been motivated to a large extent by a desire to save storage by aggregating data having identical or similar values. We will see, however, that this is not always the case. In fact, the savings in execution time that arise from this aggregation are often of equal or greater importance.

In this chapter we start with a historical overview of quadtrees, including definitions. Since the primary focus in this book is on the representation of regions, what follows is a discussion of region representation in the context of different space decomposition methods. This is done by examining polygonal and nonpolygonal tilings of the plane. The emphasis is on justifying the use of a decomposition into squares. We conclude with a detailed analysis of the space requirements of the quadtree representation.

Most of the presentation in this chapter is in the context of two-dimensional regions. The extension of the topics in this chapter, and remaining chapters, to three-dimensional region data, and higher, is straightforward and, aside from definitions, is often left to the exercises. Nevertheless, the concept of an octree, a quadtree-like representation of three-dimensional regions, is defined and a brief explanation is given of how some of the results described here are applicable to higher-dimensional data.

1.1 BASIC DEFINITIONS

First, we define a few terms with respect to two-dimensional data. Assume the existence of an array of picture elements (termed *pixels*) in two dimensions. We use the term *image* to refer to the original array of pixels. If its elements are black or

white, then it is said to be *binary*. If shades of gray are possible (i.e., gray levels), the image is said to be a *gray–scale* image. In the discussion, we are primarily concerned with binary images. Assume that the image is on an infinite background of white pixels. The *border* of the image is the outer boundary of the square corresponding to the array.

Two pixels are said to be 4-*adjacent* if they are adjacent to each other in the horizontal or vertical direction. If the concept of adjacency also includes adjacency at a corner (i.e., diagonal adjacencies), then the pixels are said to be 8-*adjacent*. A set S is said to be *four–connected* (*eight–connected*) if for any pixels p, q in S there exists a sequence of pixels $p = p_0, p_1, \cdots, p_n = q$ in S, such that p_{i+1} is 4-adjacent (8-adjacent) to p_i, $0 \leq i < n$.

A black *region*, or black four-connected *component*, is a maximal four-connected set of black pixels. The process of assigning the same label to all 4-adjacent black pixels is called *connected component labeling* (see Chapter 5 of [Same90b]). A white *region* is a maximal *eight–connected* set of white pixels defined analogously. The complement of a black region consists of a union of eight-connected white regions. Exactly one of these white regions contains the infinite background of white pixels. All the other white regions, if any, are called *holes* in the black region. The black region, say R, is surrounded by the infinite white region and R surrounds the other white regions, if any.

A pixel is said to have four edges, each of which is of unit length. The *boundary* of a black region consists of the set of edges of its constituent pixels that also serve as edges of white pixels. Similar definitions can be formulated in terms of rectangular blocks, all of whose pixels are identically colored. For example, two disjoint blocks, P and Q, are said to be 4-*adjacent* if there exists a pixel p in P and a pixel q in Q such that p and q are 4-adjacent. Eight-adjacency for blocks (as well as connected component labeling) is defined analogously.

1.2 OVERVIEW OF QUADTREES AND OCTREES

The term *quadtree* is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. They can be differentiated on the following bases:

1. The type of data they are used to represent
2. The principle guiding the decomposition process
3. The resolution (variable or not)

Currently they are used for point data, areas, curves, surfaces, and volumes. The decomposition may be into equal parts on each level (i.e., regular polygons and termed a *regular decomposition*), or it may be governed by the input. In computer graphics this distinction is often phrased in terms of image-space hierarchies versus object-space hierarchies, respectively [Suth74]. The resolution of the decomposition

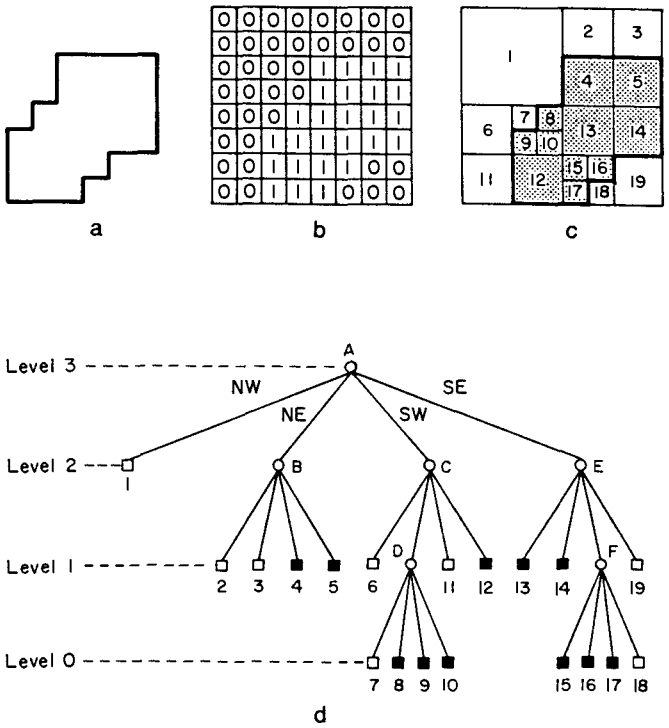


Figure 1.1 An example of (a) a region, (b) its binary array, (c) its maximal blocks (blocks in the region are shaded), and (d) the corresponding quadtree

(i.e., the number of times that the decomposition process is applied) may be fixed beforehand, or it may be governed by properties of the input data. For some applications we can also differentiate the data structures on the basis of whether they specify the boundaries of regions (e.g., curves and surfaces) or organize their interiors (e.g., areas and volumes).

The first example of a quadtree representation of data is concerned with the representation of two-dimensional binary region data. The most studied quadtree approach to region representation, called a *region quadtree* (but often termed a *quadtree* in the rest of this chapter), is based on the successive subdivision of a bounded image array into four equal-sized quadrants. If the array does not consist entirely of 1s or entirely of 0s (i.e., the region does not cover the entire array), then it is subdivided into quadrants, subquadrants, and so on, until blocks are obtained that consist entirely of 1s or entirely of 0s; that is, each block is entirely contained in the region or entirely disjoint from it. The region quadtree can be characterized as a variable resolution data structure.

As an example of the region quadtree, consider the region shown in Figure 1.1a represented by the $2^3 \times 2^3$ binary array in Figure 1.1b. Observe that the 1s correspond to picture elements (i.e., pixels) in the region, and the 0s correspond to picture elements outside the region. The resulting blocks for the array of Figure 1.1b are shown in Figure 1.1c. This process is represented by a tree of degree 4 (i.e., each nonleaf node has four sons).

In the tree representation, the root node corresponds to the entire array. Each son of a node represents a quadrant (labeled in order NW, NE, SW, SE) of the region represented by that node. The leaf nodes of the tree correspond to those blocks for which no further subdivision is necessary. A leaf node is said to be black or white depending on whether its corresponding block is entirely inside (it contains only 1s) or entirely outside the represented region (it contains no 1s). All nonleaf nodes are said to be gray (i.e., its block contains 0s and 1s). Given a $2^n \times 2^n$ image, the root node is said to be at level n while a node at level 0 corresponds to a single pixel in the image.[1] The region quadtree representation for Figure 1.1c is shown in Figure 1.1d. The leaf nodes are labeled with numbers, while the nonleaf nodes are labeled with letters. The levels of the tree are also marked.

Our definition of the region quadtree implies that it is constructed by a top-down process. In practice, the process is bottom-up, and one usually uses one of two approaches. The first approach [Same80b] is applicable when the image array is not too large. In such a case, the elements of the array are inspected in the order given by the labels on the array in Figure 1.2 (which corresponds to the image of Figure 1.1a). This order is also known as a Morton order [Mort66] (discussed in Section 1.3). By using such a method, a leaf node is never created until it is known to be maximal. An equivalent statement is that the situation does not arise in which four leaf nodes of the same color necessitate the changing of the color of their parent from gray to black or white as is appropriate. (For more details, see Section 4.1 of [Same90b].)

The second approach [Same81a] is applicable to large images. In this case, the elements of the image are processed one row at a time—for example, in the order given by the labels on the array in Figure 1.3 (which corresponds to the image of Figure 1.1a). This order is also known as a row or raster-scan order (discussed in Section 1.3). A quadtree is built by adding pixel-sized nodes one by one in the order in which they appear in the file. (For more details, see Section 4.2.1 of [Same90b].) This process can be time-consuming due to the many merging and node insertion operations that need to take place.

The above method has been improved by using a predictive method [Shaf86a, Shaf87a], which only makes a single insertion for each node in the final quadtree and performs no merge operations. It is based on processing the image in row order (top to bottom, left to right), always inserting the largest node (i.e., block) for which the current pixel is the first (upper leftmost) pixel. Such a policy avoids the necessity of merging since the upper leftmost pixel of any block is inserted before any other pixel of that block. Therefore it is impossible for four sibling nodes to be of the same color. This method makes use of an auxiliary array of size $O(2^n)$ for a $2^n \times 2^n$ image. (For more details, see Section 4.2.3 of [Same90b].)

The region quadtree is easily extended to represent three-dimensional binary region data and the resulting data structure is called a *region octree* (termed an *octree*

[1] Alternatively we can say that the root node is at depth 0 while a node at depth n corresponds to a single pixel in the image. In this book both concepts of level and depth are used to describe the relative position of nodes. The one that is chosen is context dependent.

1	2	5	6	17	18	21	22
3	4	7	8	19	20	23	24
9	10	13	14	25	26	29	30
11	12	15	16	27	28	31	32
33	34	37	38	49	50	53	54
35	36	39	40	51	52	55	56
41	42	45	46	57	58	61	62
43	44	47	48	59	60	63	64

Figure 1.2 Morton order for the pixels of Figure 1.1

in the rest of this chapter). We start with a $2^n \times 2^n \times 2^n$ object array of unit cubes (termed *voxels* or *obels*). The octree is based on the successive subdivision of an object array into octants. If the array does not consist entirely of 1s or entirely of 0s, it is subdivided into octants, suboctants, and so on until cubes (possibly single voxels) are obtained that consist of 1s or of 0s; that is, they are entirely contained in the region or entirely disjoint from it.

This subdivision process is represented by a tree of degree 8 in which the root node represents the entire object and the leaf nodes correspond to those cubes of the array for which no further subdivision is necessary. Leaf nodes are said to be black or white (alternatively, full or void) depending on whether their corresponding cubes are entirely within or outside the object, respectively. All nonleaf nodes are said to be gray. Figure 1.4a is an example of a simple three-dimensional object, in the form of a staircase, whose octree block decomposition is given in Figure 1.4b and whose tree representation is given in Figure 1.4c.

The region quadtree is a member of a class of representations characterized as being a collection of maximal (according to an appropriate definition) blocks, each of which is contained in a given region and whose union is the entire region. The simplest such representation is the runlength code, where the blocks are restricted to $1 \times m$ rectangles [Ruto68]. A more general representation treats the region as a union of maximal square blocks (or blocks of any other desired shape) that may possibly overlap. Usually the blocks are specified by their centers and radii. This representation is called the *medial axis transformation* (*MAT*) [Blum67, Rose66]. Of course, other approaches are also possible (e.g., rectangular coding [Kim83, Kim86], TID [Scot85, Scot86]).

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Figure 1.3 Raster-scan order for the pixels of Figure 1.1

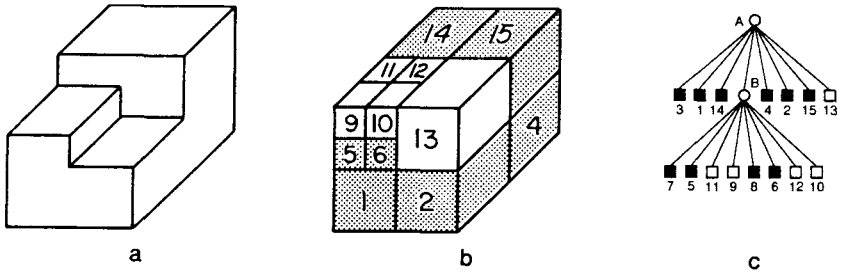


Figure 1.4 (a) Example three-dimensional object; (b) its octree block decomposition; (c) its tree representation

The region quadtree is a variant on the maximal block representation. It requires the blocks to be disjoint and to have standard sizes (i.e., sides of lengths that are powers of two) and standard locations. The motivation for its development is a desire to obtain a systematic way to represent homogeneous parts of an image. Thus to transform the data into a region quadtree, a criterion must be chosen for deciding that an image is homogeneous (i.e., uniform).

One such criterion is that the standard deviation of its gray levels is below a given threshold t. Using this criterion, the image array is successively subdivided into quadrants, subquadrants, and so on until homogeneous blocks are obtained. This process leads to a regular decomposition. If one associates with each leaf node the mean gray level of its block, the resulting region quadtree will then completely specify a piecewise approximation to the image where each homogeneous block is represented by its mean. The case where $t=0$ (i.e., a block is not homogeneous unless its gray level is constant) is of particular interest since it permits an exact reconstruction of the image from its quadtree.

Note that the blocks of the region quadtree do not necessarily correspond to maximal homogeneous regions in the image. Most likely there exist unions of the blocks that are still homogeneous. To obtain a segmentation of the image into maximal homogeneous regions, we must allow merging of adjacent blocks (or unions of blocks) as long as the resulting region remains homogeneous. This is achieved by a 'split-and-merge' algorithm [Horo76]. However, the resulting partition will no longer be represented by a quadtree; instead the final representation is in the form of an adjacency graph. Thus the region quadtree is used as an initial step in the segmentation process.

For example, Figure 1.5b–d demonstrates the results of the application, in sequence, of merging, splitting, and grouping to the initial image decomposition of Figure 1.5a. In this case, the image is initially decomposed into 16 equal-sized square blocks. Next the 'merge' step attempts to form larger blocks by recursively merging groups of four homogeneous 'brothers' (the four blocks in the NW and SE quadrants of Figure 1.5b). The 'split' step recursively decomposes blocks that are not homogeneous (the NE and SW quadrants of Figure 1.5c) until a particular homogeneity criterion is satisfied or a given level is encountered. Finally the 'grouping' step aggregates all homogeneous 4-adjacent black blocks into one region apiece;

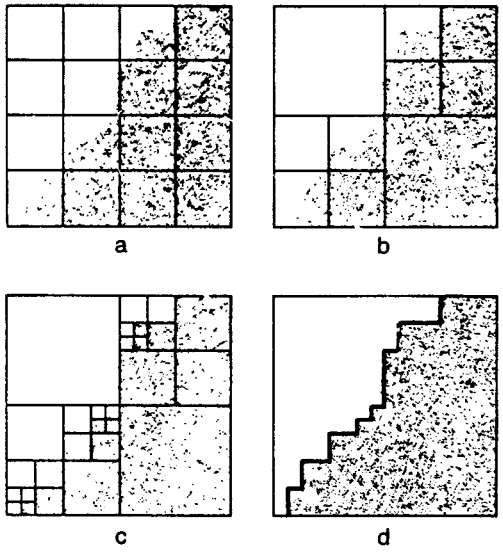


Figure 1.5 Example illustrating the 'split-and-merge' segmentation procedure: (a) start, (b) merge, (c) split, (d) grouping

the 8-adjacent white blocks are similarly aggregated into white regions (Figure 1.5d).

An alternative to the region quadtree representation is to use a decomposition method that is not regular (i.e., rectangles of arbitrary size rather than squares). This alternative has the potential of requiring less space. Its drawback is that the determination of optimal partition points may be computationally expensive (see Exercise 1.10). A closely related problem, decomposing a region into a minimum number of rectangles, is known to be NP-complete[2] [Gare79] if the region is permitted to contain holes [Ling82].

The homogeneity criterion ultimately chosen to guide the subdivision process depends on the type of region data represented. In the remainder of this chapter we shall assume that the domain is a $2^n \times 2^n$ binary image with 1, or black, corresponding to foreground and 0, or white, corresponding to background (e.g., Figure 1.1).

---

[2] A problem is in NP if it can be solved nondeterministically in polynomial time. A nondeterministic solution process proceeds by 'guessing' a solution and then verifying that the solution is correct. Assume that n is the size of the problem (e.g., for sorting, n is the number of records to be sorted). Intuitively, then, a problem is in NP if there is a polynomial $P(n)$ such that if one guesses a solution, it can be verified in $O(P(n))$ time, whether the guess is indeed a correct solution. Thus the verification process is the key to determining whether a problem is in NP, not the actual solution of the problem.

A problem is NP-complete if it is 'at least as hard' as any other problem in NP. Somewhat more formally, a problem P_1 in NP is NP-complete if the following property holds: for all other problems P_i in NP, if P_1 can be solved deterministically in $O(f(n))$ time, then P_i can be solved in $O(P(f(n)))$ time for some polynomial P. It has been conjectured that no NP-complete problem can be solved deterministically in polynomial time, but this is not known for sure. The theory of NP-completeness is discussed in detail in [Gare79].

Nevertheless the quadtree and octree can be used to represent multicolored data (e.g., a landuse class map associating colors with crops [Same87a]).

It is interesting to note that Kawaguchi, Endo, and Matsunaga [Kawa83] use a sequence of m binary-valued quadtrees to encode image data of 2^m gray levels, where the various gray levels are encoded by use of Gray codes (see, e.g., [McCl65]). This should lead to compaction (i.e., larger-sized blocks) since the Gray code guarantees that the binary representation of the codes of adjacent gray level values differ by only one binary digit.[3] Note, though, that if the primary interest is in image compression, there exist even better methods (see, e.g., [Prat78]); however, they are beyond the scope of this book (but see Chapter 8 of [Same90b]). In another context, Kawaguchi, Endo, and Yokota [Kawa80b] point out that a sequence of related images (e.g., in an animation application) can be stored compactly as a sequence of quadtrees such that the i^{th} element is the result of exclusive oring the first i images (see Exercise 1.7).

Unfortunately the term *quadtree* has taken on more than one meaning. The region quadtree, as described earlier, is a partition of space into a set of squares whose sides are all a power of two long. This formulation is due to Klinger [Klin71] and Klinger and Dyer, who used the term *Q-tree* [Klin76], whereas Hunter [Hunt78] was the first to use the term *quadtree* in such a context. Actually a more precise term would be *quadtrie*, as it is really a trie structure [Fred60] in two dimensions.[4] A similar partition of space into rectangular quadrants, also termed a *quadtree*, was used by Finkel and Bentley [Fink74]. It is an adaptation of the binary search tree [Knut73b] to two dimensions (which can be easily extended to an arbitrary number of dimensions). It is primarily used to represent multidimensional point data, and we shall refer to it as a *point quadtree* where confusion with a region quadtree is possible.

As an example of a point quadtree, consider Figure 1.6, which is built for the sequence Chicago, Mobile, Toronto, Buffalo, Denver, Omaha, Atlanta, and Miami[5]

---

[3] The Gray code is motivated by a desire to reduce errors in transitions between successive gray level values. Its one bit difference guarantee is achieved by the following encoding. Consider the binary representation of the integers from 0 to $2^m - 1$. This representation can be obtained by constructing a binary tree, say T, of height m where each left branch is labeled 0 while each right branch is labeled 1. Each leaf node, say P, is given the label formed by concatenating the labels of the branches taken by the path from the root to P. Enumerating the leaf nodes from left to right yields the binary integers 0 to $2^m - 1$. The Gray codes of the integers are obtained by constructing a new binary tree, say T', such that the labels of some of the branches in T' are the reverse of what they were in T. The algorithm is as follows. Initially, T' is a copy of T. Next, traverse T in preorder (i.e., visit the root node, followed by the left and right subtrees). For each branch in T labeled 1, exchange the labels of the two descendant branches of its corresponding branch in T'. No action is taken for descendants of branches in T labeled 0. Enumerating the leaf nodes in T' from left to right yields the Gray codes of the integers 0 to $2^m - 1$. For example, for 8 gray levels (i.e., $m = 3$), we have 000, 001, 011, 010, 110, 111, 101, 100.

[4] In a one-dimensional *trie* structure, each data item or key is treated as a sequence of characters where each character has M possible values. A node at depth i in the trie represents an M-way branch depending on the i^{th} character. The data are stored in the leaf nodes, and the shape of the trie is independent of the order in which the data are processed. Such a structure is also known as a *digital tree* [Knut73b].

[5] The correspondence between coordinate values and city names is not geographically correct. This liberty has been taken so that the same example can be used throughout the text to illustrate a variety of concepts.

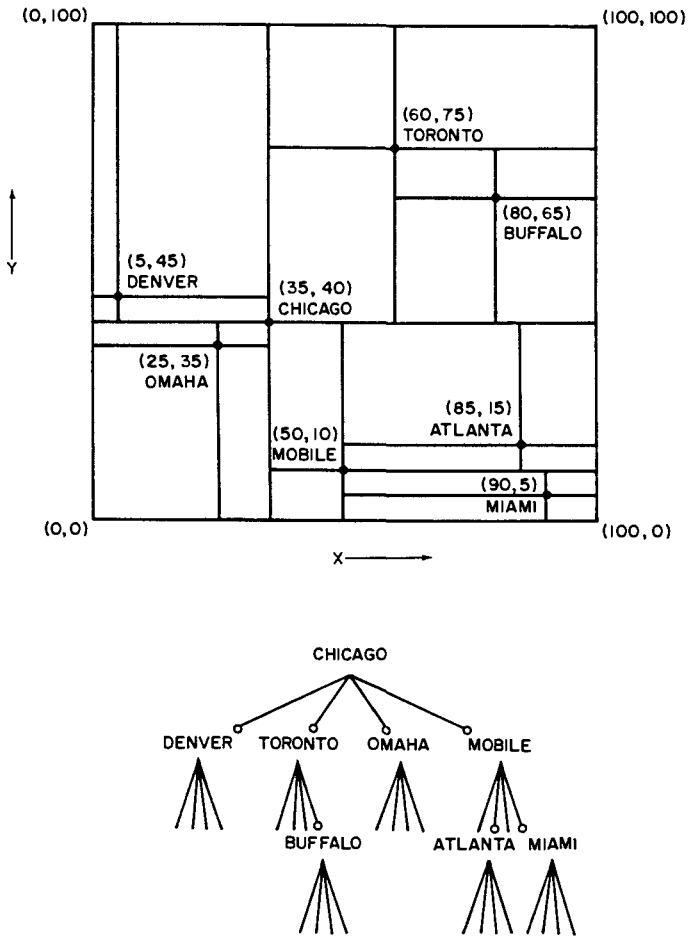


Figure 1.6 A point quadtree and the records it represents

in the order in which they are listed here.[6] Its shape is highly dependent on the order in which the points are added to it. Of course, trie-based point representations also exist (see Sections 2.6.1 and 2.6.2).

Exercises

1.1. The region quadtree is an alternative to an image representation that is based on the use of an array or even a list. Each of these image representations may be biased in favor of the computation of a particular adjacency relation. Discuss these biases for the array, list, and quadtree representations.

1.2. Given the array representation of a binary image, write an algorithm to construct the corresponding region quadtree.

[6] Refer to Figure 2.5 to see how the point quadtree is constructed in an incremental fashion for Chicago, Mobile, Toronto, and Buffalo.

1.3. Given an image represented by a region quadtree with B black and W white nodes, how many additional nodes are necessary for the nonleaf nodes?

1.4. Given an image represented by a region octree with B black and W white nodes, how many additional nodes are necessary for the nonleaf nodes?

1.5. Suppose that an octree is used to represent a collection of disjoint spheres. What would you use as a leaf criterion?

1.6. The quadtree can be generalized to represent data in arbitrary dimensions. As we saw, the octree is its three-dimensional analog. The renowned artist Escher [Coxe86] is noted for etchings of unusual interpretations of geometric objects such as staircases. How would you represent one of Escher's staircases?

1.7. Let $\oplus$ denote an exclusive or operation. Given a sequence of related images, $<P_n, P_{n-1}, \cdots, P_0>$, define another sequence $<Q_n, Q_{n-1}, \cdots, Q_0>$ such that $Q_0 = P_0$ and $Q_i = P_i \oplus Q_{i-1}$ for $i > 0$. Show that when the sequences P and Q are represented as quadtrees, replacing sequence P by sequence Q results in fewer nodes.

1.8. Prove that in Exercise 1.7 the sequence P can be reconstructed from the sequence Q. In particular, given Q_i and Q_{i-1}, determine P_i.

1.9. Write an algorithm to construct the Gray codes of the integers 0 to 2^m-1.

1.10. Find a polynomial-time algorithm to decompose a region optimally so that its quadtree representation uses a minimum amount of space (i.e., a minimum number of nodes). In this case, you can assume that the decomposition lines can be placed in arbitrary positions so that the space requirement is reduced. In other words, the decomposition lines need not split the space into four squares of equal size. Thus the decomposition is similar to that induced by a point quadtree.

1.3 HISTORY OF THE USE OF QUADTREES AND OCTREES

The origin of the principle of recursive decomposition, upon which all quadtrees are based, is difficult to ascertain. Below, to give some indication of the uses of the region quadtree, some of its applications to geometric data are traced briefly. Most likely it was first seen as a way of aggregating blocks of zeros in sparse matrices. Indeed Hoare [Hoar72] attributes a one-level decomposition of a matrix into square blocks to Dijkstra. Morton [Mort66] used it as a means of indexing into a geographic database (i.e., it acts as a spatial index).

Warnock, in a pair of reports that serve as landmarks in computer graphics [Warn68, Warn69b], described the implementation of hidden-line and hidden-surface elimination algorithms using a recursive decomposition of the picture area. The picture area is repeatedly subdivided into rectangles that are successively smaller while searching for areas that are sufficiently simple to be displayed. Klinger [Klin71] and Klinger and Dyer [Klin76] applied these ideas to pattern recognition and image processing, while Hunter [Hunt78] used them for an animation application.

The SRI robot project [Nils69] used a three-level decomposition of space to represent a map of the robot's world. Eastman [East70] observes that recursive decomposition might be used for space planning in an architectural context and presents a simplified version of the SRI robot representation. A quadtree-like representation in the form of production rules called DF-expressions (denoting 'depth-first') is discussed by Kawaguchi and Endo [Kawa80a] and Kawaguchi, Endo, and Yokota

[Kawa80b] (see also Section 1.5). Tucker [Tuck84a] uses quadtree refinement as a control strategy for an expert vision system.

The three-dimensional variant of the region quadtree—the octree—was developed independently by a number of researchers. Hunter [Hunt78] mentioned it as a natural extension of the quadtree. Reddy and Rubin [Redd78] proposed the octree as one of three representations for solid objects. The second is a three-dimensional generalization of the point quadtree of Finkel and Bentley [Fink74]—that is, a decomposition into rectangular parallelepipeds (as opposed to cubes) with planes perpendicular to the x, y, and z axes. The third breaks the object into rectangular parallelepipeds that are not necessarily aligned with an axis. The parallelepipeds are of arbitrary sizes and orientations. Each parallelepiped is recursively subdivided into parallelepipeds in the coordinate space of the enclosing parallelepiped. Reddy and Rubin prefer the third approach for its ease of display.

Situated somewhere between the second and third approaches of Reddy and Rubin is the method of Brooks and Lozano-Perez [Broo83] (see also [Loza81]), who use a recursive decomposition of space into an arbitrary number of rectangular parallelepipeds, with planes perpendicular to the x, y, and z axes, to model space in solving the *findpath* or *piano movers* problem [Schw88] in robotics. This problem arises when planning the motion of a robot in an environment containing known obstacles and the desired solution is a collision-free path obtained by use of a search. Faverjon [Fave84] discusses an approach to this problem that uses an octree, as do Samet and Tamminen [Same85g] and Fujimura and Samet [Fuji89].

Jackins and Tanimoto [Jack80] adapted Hunter and Steiglitz's quadtree translation algorithm [Hunt78, Hunt79b] to objects represented by octrees. Meagher [Meag82a] developed numerous algorithms for performing solid modeling operations in an environment where the octree is the underlying representation. Yau and Srihari [Yau83] extended the octree to arbitrary dimensions in the process of developing algorithms to handle medical images.

Both quadtrees and octrees are frequently used in the construction of meshes for finite element analysis. The use of recursive decomposition for meshes was initially suggested by Rheinboldt and Mesztenyi [Rhei80]. Yerry and Shephard [Yerr83] adapted the quadtree and octree to generate meshes automatically for three-dimensional solids represented by a superquadric surface-based modeler. This has been extended by Kela, Voelcker, and Goldak [Kela84b] (see also [Kela86]) to mesh boundary regions directly, rather than through discrete approximations, and to facilitate incremental adaptive analysis by exploiting the spatial index nature of the quadtree and octree.

Parallel to the development of the quadtree and octree data structures, there has been related work by researchers in the field of image understanding. Kelly [Kell71] introduced the concept of a *plan*, which is a small picture whose pixels represent gray-scale averages over 8×8 blocks of a larger picture. Needless effort in edge detection is avoided by first determining edges in the plan and then using these edges to search selectively for edges in the larger picture. Generalizations of this idea motivated the development of multiresolution image representations—for example,

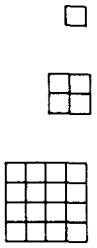


Figure 1.7 Structure of a pyramid having three levels

the recognition cone of Uhr [Uhr72], the preprocessing cone of Riseman and Arbib [Rise77], and the pyramid of Tanimoto and Pavlidis [Tani75]. Of these representations, the pyramid is the closest relative of the region quadtree.

Given a $2^n \times 2^n$ image array, say $A(n)$, a *pyramid* is a sequence of arrays $\{A(i)\}$ such that $A(i-1)$ is a version of $A(i)$ at half the scale of $A(i)$. $A(0)$ is a single pixel. Figure 1.7 shows the structure of a pyramid having three levels. It should be clear that a pyramid can also be defined in a more general way by permitting finer scales of resolution than the power of two scale.

At times, it is more convenient to define a pyramid in the form of a tree. Again, assuming a $2^n \times 2^n$ image, a recursive decomposition into quadrants is performed, just as in quadtree construction, except that we keep subdividing until we reach the individual pixels. The leaf nodes of the resulting tree represent the pixels, while the nodes immediately above the leaf nodes correspond to the array $A(n-1)$, which is of size $2^{n-1} \times 2^{n-1}$. The nonleaf nodes are assigned a value that is a function of the nodes below them (i.e., their sons) such as the average gray level. Thus we see that a pyramid is a multiresolution representation, whereas the region quadtree is a variable

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Figure 1.8 Example pyramid A(3)

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

Figure 1.9 A(2) corresponding to Figure 1.8

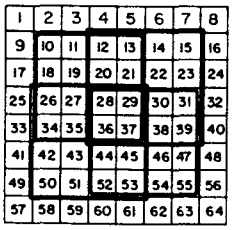


Figure 1.10 The overlapping blocks in which pixel 28 participates

resolution representation. Another analogy is that the pyramid is a complete quadtree [Knut73a].

The above definition of a pyramid is based on nonoverlapping 2×2 blocks of pixels. An alternative definition, termed an *overlapping pyramid*, uses overlapping blocks of pixels. One of the simplest schemes makes use of 4×4 blocks that overlap by 50% in both the horizontal and vertical directions [Burt81]. For example, Figure 1.8 is a $2^3 \times 2^3$ array, say $A(3)$, whose pixels are labeled 1-64. Figure 1.9 is $A(2)$ corresponding to Figure 1.8 with elements labeled A-P. The 4×4 neighborhood corresponding to element F in Figure 1.9 consists of pixels 10–13, 18–21, 26–29, and 34–37. This method implies that each block at a given level participates in four blocks at the immediately higher level. Thus the containment relations between blocks no longer form a tree. For example, pixel 28 participates in blocks F, G, J, and K in the next higher level (see Figure 1.10 where the four neighborhoods corresponding to F, G, J, and K are drawn as squares).

To avoid treating border cases differently, each level in the overlapped pyramid is assumed to be cyclically closed (i.e., the top row at each level is adjacent to the bottom row and similarly for the columns at the extreme left and right of each level). Once again we say that the value of a node is the average of the values of the nodes in its block on the immediately lower level. The overlapped pyramid may be compared with the Quadtree Medial Axis Transform (see Section 9.3.1 of [Same90b]) in the sense that both may result in nondisjoint decompositions of space.

Pyramids have been applied to the problems of feature detection and extraction since they can be used to limit the scope of the search. Once a piece of information of interest is found at a coarse level, the finer resolution levels can be searched. This approach was followed by Davis and Roussopoulos [Davi80] in approximate pattern matching. Pyramids can also be used for encoding information about edges, lines, and curves in an image [Shne81c, Krop86]. One note of caution: the reduction of resolution has an effect on the visual appearance of edges and small objects [Tani76]. In particular, at a coarser level of resolution, edges tend to get smeared, and region separation may disappear. Pyramids have also been used as the starting point for a 'split-and-merge' segmentation algorithm [Piet82].

Quadtree-like decompositions are useful as space-ordering methods. The purpose is to optimize the storage and processing sequences for two-dimensional data by mapping them into one dimension (i.e., linearizing them). This mapping should pre-

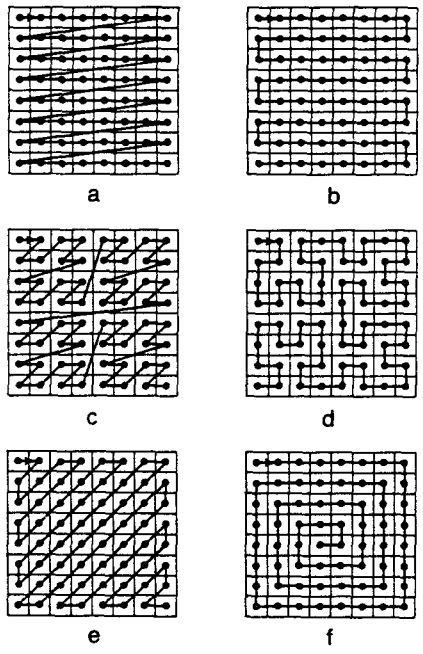


Figure 1.11 The result of applying a number of different space-ordering methods to an 8 × 8 image whose first element is in the upper left corner of the image: (a) row order, (b) row-prime order, (c) Morton order, (d) Peano-Hilbert order, (e) Cantor-diagonal order, (f) spiral order

serve the spatial locality of the original two-dimensional image in one dimension. The result of the mapping is also known as a *space-filling curve* [Gold81, Witt83] because it passes through every point in the image.

Goodchild and Grandfield [Good83] discuss a number of space-ordering methods, some of which are illustrated in Figure 1.11. Each has different characteristics. The row (Figure 1.11a), also known as raster-scan, and row-prime orders (Figure 1.11b) are similar in the same way as are the Morton [Mort66, Pean90] (Figure 1.11c) and the Peano-Hilbert [Hilb91] (Figure 1.11d) orders. The primary difference is that in both the row-prime and Peano-Hilbert orders every element is a 4-adjacent neighbor of the previous element in the sequence, and thus they have a slightly higher degree of locality than the row and Morton orders, respectively. Both the Morton and Peano-Hilbert orders exhaust a quadrant or subquadrant of a square image before exiting it. They are both related to quadtrees; however, as we saw above, the Morton order does not traverse the image in a spatially contiguous manner (the result has the shape of the letter 'N' or 'Z' and is also known as N order [Whit82] and Z order [Oren84]).

For both the Morton and Peano-Hilbert orders, there is no need to know the maximum values of the coordinates. The Morton order is symmetric, while the Peano-Hilbert order is not. One advantage of the Morton order is that the position of each element in the ordering (termed its *key*) can be determined by interleaving the

bits of the x and y coordinates of the element; this is not easy for the Peano-Hilbert order. Another advantage of the Morton order is that the recursion necessary for its generation is quite easy to specify.

Other orders are the Cantor-diagonal order (Figure 1.11e) and the spiral order (Figure 1.11f). The Cantor-diagonal order proceeds outward from the origin and visits the elements in an order similar to row-prime with the difference that elements are visited in order of their increasing 'Manhattan' (or 'city block') distance.[7] Thus it is good for ordering a space that is unbounded in the two directions emanating from the origin which has been relocated to the center of the image. On the other hand, the spiral order is attractive when ordering a space that is unbounded in the four directions emanating from the origin.

The most interesting orders, as far as we are concerned, are the Morton and Peano-Hilbert orders since they can also be used to order a space that has been aggregated into squares. Of these two orderings, the Morton order is by far the more frequently used as a result of the simplicity of the conversion process between the key and its corresponding element in the multidimensional space. In this book we are primarily interested in Morton orderings. (For further discussion of some of the properties of these two orderings, see [Patr68, Butz71, Alex79, Alex80, Laur85].)

Exercises

1.11. Write an algorithm to extract the x and y coordinates from a Peano-Hilbert order key.

1.12. Write an algorithm to construct the Peano-Hilbert key for a given point (x, y). Try to make it optimal.

1.13. Suppose that you are given a $2^n \times 2^n$ array of points such that the horizontal and vertical distances between 4-adjacent points are 1. What is the average distance between successive points when the points are ordered according to the orders illustrated in Figure 1.11? What about a random order?

1.14. Suppose that you are given a $2^n \times 2^n$ image. Assume that the image is stored on disk in pages of size $2^m \times 2^m$ where n is much larger than m. What is the average cost of retrieving a pixel and its 4-adjacent neighbors when the image is ordered according to the orders illustrated in Figure 1.11?

1.15. The traveling salesman problem [Lawl85] is one where a set of points is given and it is desired to find the path of minimum distance such that each point is visited only once. This is an NP-complete problem [Gare79] and thus there is a considerable amount of work in formulating approximate solutions to it [Bent82]. For example, consider the following approximate solution. Assume that the points are uniformly distributed in the unit square. Let d be the expected Euclidean distance between two independent points. Now, sort the points using the row order and the Morton order. Laurini [Laur85] simulated the average Euclidean distance between successive points in these orders and found it to be $d/2$ for the row order and $d/3$ for the Morton order. Can you derive these averages analytically? What are the average values for the other orders illustrated in Figure 1.11? What about a random order?

[7] The Manhattan distance between points (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$ (for more details, see Section 9.1 of [Same90b]).

1.16. Suppose that the traveling salesman problem is solved using a traversal of the points in Morton order as discussed in Exercise 1.15. In particular, assume that the set of points is decomposed in such a way that each square block contains just one point. This yields a point representation that is analogous to the region quadtree (termed a PR quadtree and discussed in Section 2.6.2). How close does such a solution come to optimality?

1.4 SPACE DECOMPOSITION METHODS

In general, any planar decomposition used as a basis for an image representation should possess the following two properties:

1. The partition should be an infinitely repetitive pattern so that it can be used for images of any size.
2. The partition should be infinitely decomposable into increasingly finer patterns (i.e., higher resolution).

In this section, the discussion is restricted to two-dimensional data. Thus we are dealing with planar space decompositions. Space decompositions can be classified into two categories, depending on the nature of the pattern. The pattern can consist of polygonal shapes or nonpolygonal shapes. The polygonal shapes are generally computationally simpler since their sides can be expressed in terms of linear relations (e.g., equations of lines). They are good for approximating the interior of a region. The nonpolygonal shapes are more flexible since they provide good approximations, in terms of measures, of the boundaries (e.g., perimeter) of regions as well as their interiors (e.g., area).[8]

Moreover, the normals to the boundaries of nonpolygonal shapes are not restricted to a fixed set of directions. For example, in the case of rectangular tiles, there is a 90 degree discontinuity between the normals to boundaries of adjacent tiles. This lack of continuity is a drawback in applications in fields such as computer graphics where such tasks as shading make use of the directions of the surface. However, working with nonpolygonal shapes generally requires use of floating point arithmetic, and hence it is usually more complex.

The remainder of this section expands on a number of polygonal decompositions and compares them. It also contains a brief discussion of one nonpolygonal decomposition that consists of a collection of sector-like objects whose arcs are not necessarily part of a circle. This method is based on polar coordinates where the arc joining two distinct points is formed by linear interpolation. The term *sector tree* is used to describe it. This discussion is of an advanced nature and can be skipped on an initial reading.

[8] Recall the statement in Section 1.2 that hierarchical data structures are often differentiated on the basis of whether they specify the boundaries of regions or organize their interiors.

1.4.1 Polygonal Tilings

Bell, Diaz, Holroyd, and Jackson [Bell83] discuss a number of polygonal tilings of the plane (i.e., tessellations) that satisfy property 1. Figure 1.12 illustrates some of these tessellations. They also present a taxonomy of criteria to distinguish between the various tilings. The tilings, consisting of polygonal tiles, are described by use of a notation based on the degree of each vertex as the edges (i.e., sides) of the 'atomic' tile are visited in order, forming a cycle. For example, the tiling described by $[4.8^2]$ (Figure 1.12c) has the shape of a triangle where the first vertex has degree four while the remaining two vertices have degree eight apiece.

A tiling is said to be *regular* if the atomic tiles are composed of regular polygons (i.e., all sides are of equal length as are the interior angles). A *molecular tile* is an aggregation of atomic tiles to form a hierarchy. It is not necessarily constrained to have the same shape as the atomic tile. When a tile at level k (for all $k>0$) has the same shape as a tile at level 0 (i.e., it is a scaled image of a tile at level 0), then the tiling is said to be *similar*.

Bell et al. focus on the isohedral tilings where a tiling is said to be *isohedral* if all the tiles are equivalent under the symmetry group of the tiling. A more intuitive

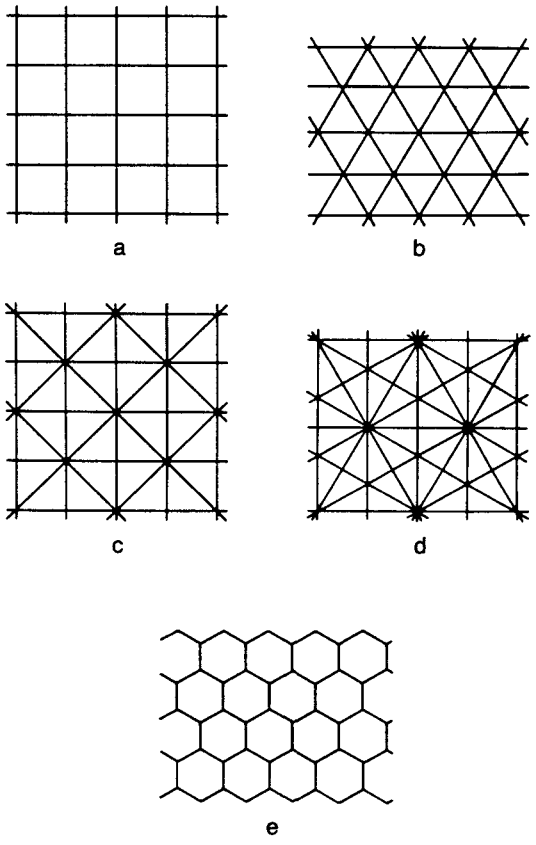


Figure 1.12 Sample tessellations: (a) $[4^4]$ square; (b) $[6^3]$ equilateral triangle; (c) $[4.8^2]$ isoceles triangle; (d) [4.6.12] 30–60 right triangle; (e) $[3^6]$ hexagon

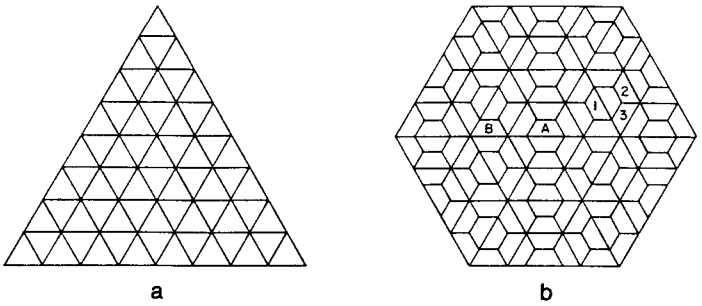


Figure 1.13 Examples of (a) isohedral and (b) nonisohedral tilings

way to conceptualize this definition is to assume the position of an observer who stands in the center of a tile having a given orientation and scans the surroundings. If the view is independent of the tile, the tiling is isohedral. For example, consider the two tilings in Figure 1.13 consisting of triangles (Figure 1.13a) and trapezoids (Figure 1.13b). The triangles are isohedral, whereas the trapezoids are not, as can be seen by the view from tiles A and B.

In the case of the trapezoidal tiling, the viewer from A is surrounded by an infinite number of concentric hexagons, whereas this is not the case for B. In other words, the trapezoidal tiling is not periodic. Also note that all of the tiles in Figure 1.13a are described by $[6^3]$, while those in Figure 1.13b are either $[3^2.4^2]$, $[3^2.6^2]$, or $[3.4.6^2]$ (i.e., tiles labeled 1, 2, and 3, respectively, in Figure 1.13b). When the isohedral tilings are classified by the action of their symmetry group, there are 81 different types [Grün77, Grün87]. When they are classified by their adjacency structure, as done here, there are 11 types.

The most relevant criterion to the discussion is the distinction between limited and unlimited hierarchies of tilings. A *limited* tiling is not similar. A tiling that satisfies property 2 is said to be *unlimited*. Equivalently, in a limited tiling, no change of scale lower than the limit tiling can be made without great difficulty. An alternative characterization of an unlimited tiling is that each edge of a tile lies on an infinite straight line composed entirely of edges. Interestingly the hexagonal tiling $[3^6]$ is limited. Bell et al. claim that only four tilings are unlimited. These are the tilings given in Figure 1.12a–d. Of these, $[4^4]$, consisting of square atomic tiles (Figure 1.12a), and $[6^3]$, consisting of equilateral triangle atomic tiles (Figure 1.12b), are well-known regular tessellations [Ahuj83]. For these two tilings we consider only the molecular tiles given in Figures 1.14a and 1.14b.

The tilings $[4^4]$ and $[6^3]$ can generate an infinite number of different molecular tiles where each molecular tile at the first level consists of n^2 atomic tiles ($n > 1$). The remaining nonregular unlimited triangular tilings, $[4.8^2]$ (Figure 1.12c) and [4.6.12] (Figure 1.12d), are less well understood. One way of generating $[4.8^2]$ and [4.6.12] is to join the centroids of the tiles of $[4^4]$ and $[6^3]$, respectively, to both their vertices and midpoints of their edges. Each of the tilings $[4.8^2]$ and [4.6.12] has two

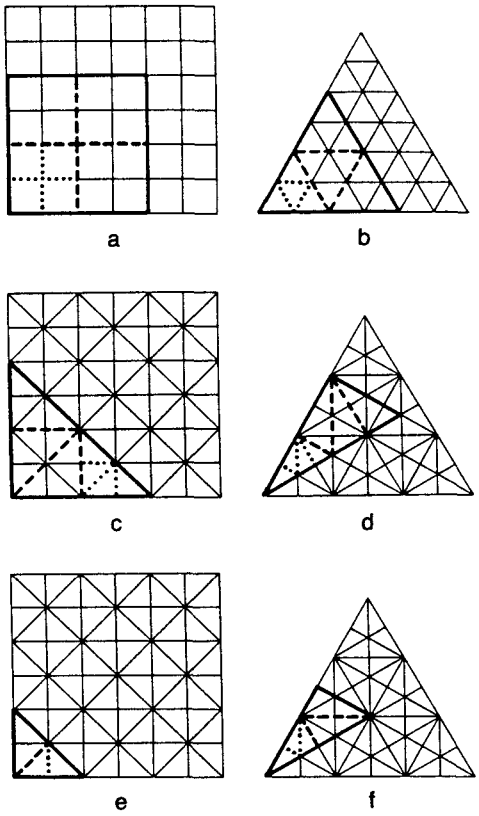


Figure 1.14 Examples illustrating unlimited tilings: (a) [4^4] hierarchy, (b) [6^3] hierarchy, (c) ordinary [4.8^2] hierarchy, (d) ordinary [4.6.12] hierarchy, (e) rotation [4.8^2] hierarchy, (f) reflection [4.6.12] hierarchy

types of hierarchy. [4.8^2] has an ordinary (Figure 1.14c) and a rotation hierarchy (Figure 1.14e) requiring a rotation of 135 degrees between levels. [4.6.12] has an ordinary (Figure 1.14d) and a reflection hierarchy (Figure 1.14f), which requires a reflection of the basic tile between levels.

The distinction between the two types of hierarchies for [4.8^2] and [4.6.12] is necessary because the tiling is not similar without a rotation or a reflection when the hierarchy is not ordinary. This can be seen by observing the use of dots in Figure 1.14 to delimit the atomic tiles in the first molecular tile. Similarly broken lines are used to delimit the components of tiles at the second level (assuming atomic tiles are at level 0). For the ordinary [4.8^2] and [4.6.12] hierarchies, each molecular tile at the first level consists of n^2 ($n > 1$) atomic tiles. In the reflection hierarchy of [4.6.12], each molecular tile at the first level consists of $3 \cdot n^2$ ($n > 1$) atomic tiles, while for the rotation hierarchy of [4.8^2], $2 \cdot n^2$ ($n > 1$) atomic tiles comprise a molecular tile at the first level.

To represent data in the Euclidean plane, any of the unlimited tilings could have been chosen. For a regular decomposition, the tilings [4.8^2] and [4.6.12] are ruled out. Comparing 'square' [4^4] and 'triangular' [6^3] quadtrees, we find that they differ in terms of adjacency and orientation. Let us say that two tiles are *neighbors* if they are

adjacent either along an edge or at a vertex. A tiling is *uniformly adjacent* if the distances between the centroid of one tile and the centroids of all its neighbors are the same. The adjacency number of a tiling is the number of different intercentroid distances between any one tile and its neighbors. In the case of $[4^4]$, there are only two adjacency distances, whereas for $[6^3]$ there are three adjacency distances.

A tiling is said to have *uniform orientation* if all tiles with the same orientation can be mapped into each other by translations of the plane that do not involve rotation or reflection. Tiling $[4^4]$ displays uniform orientation, while $[6^3]$ does not. Under the assumption that uniform orientation and a minimal adjacency distance is preferable, we say that $[4^4]$ is more useful than $[6^3]$. It is also very easy to implement. Nevertheless, $[6^3]$ has its uses. For example, Yamaguchi, Kunii, Fujimura, and Toriya [Yama84] use a triangular quadtree to generate an isometric view from an octree representation of an object (see Section 7.1.4 of [Same90b]).

Of the *limited* tilings, many types of hierarchies may be generated [Bell83]; however, in general, they cannot be decomposed beyond the atomic tiling without changing the basic tile shape. This is a serious deficiency of the hexagonal tessellation $[3^6]$ (Figure 1.12e) since the atomic hexagon can be decomposed only into triangles. Nevertheless the hexagonal tessellation is of considerable interest. It is regular, has a uniform orientation, and, most important, displays a uniform adjacency (i.e., each neighbor of a tile is at the same distance from it).

There are a number of different hexagonal hierarchies distinguished by classifying the shape of the first-level molecular tile on the basis of the number of hexagons that it contains. Three of these tiling hierarchies are given in Figure 1.15 and are called *n-shapes* where *n* denotes the number of atomic tiles in the first-level molecular tile. Of course, these n-shapes are not unique.

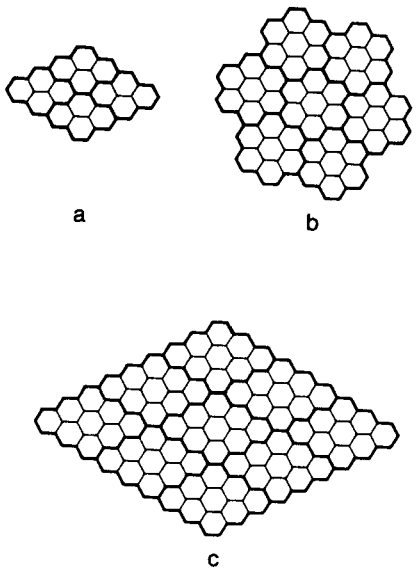


Figure 1.15 Three different hexagonal tiling hierarchies: (a) 4-shape, (b) 7-shape, (c) 9-shape

The 4-shape and the 9-shape have an unusual adjacency property in the sense that no matter how large the molecular tile becomes, contact with two of the tiles (i.e., the one above and the one below) is along only one edge of a hexagonal atomic tile, while contact with the remaining four molecular tiles is along nearly one-quarter of the perimeter of the corresponding molecular tile. The hexagonal pattern of the 4-shape and 9-shape molecular tiles has the shape of a rhombus. In contrast, a 7-shape molecular tile has a uniform contact with its six neighboring molecular tiles.

The type of quadtree used often depends on the grid formed by the image sampling process. Square quadtrees are appropriate for square grids and triangular quadtrees for triangular grids. In the case of a hexagonal grid [Burt80], the 7-shape hierarchy is frequently used since the shape of its molecular tile is more like a hexagon. It is usually described as *rosette*–like (i.e., a *septree*). Note that septrees have jagged edges as they are merged to form larger units (e.g., Figure 1.15b). The septree is used by Gibson and Lucas [Gibs82] (who call it a *generalized balanced ternary* or *GBT* for short) in the development of algorithms analogous to those existing for quadtrees.

Although the septree can be built up to yield large septrees, the smallest resolution in the septree must be decided upon in advance since its primitive components (i.e., hexagons) cannot later be decomposed into septrees. Therefore the septree yields only a partial hierarchical decomposition in the sense that the components can always be merged into larger units, but they cannot always be broken down. For region data, a pixel is generally an indivisible unit, and thus unlimited decomposition is not absolutely necessary. However, in the case of other data types such as points (see Chapter 2) and lines (see Chapter 4), we will see that the decomposition rules of some representations require that two entities be separated, which may lead to a level of decomposition not known in advance (e.g., a decomposition rule that restricts each square to contain at most one point). In this book the discussion is limited to square quadtrees and their variants.

When the data are spherical, a number of researchers have proposed the use of a representation based on an icosahedron (a 20-faced polyhedron whose faces are regular triangles) [Dutt84, Feke84]. The icosahedron is attractive because, in terms of the number of faces, it is the largest possible regular polyhedron. Each of the triangular faces can be further decomposed in a recursive manner into n^2 ($n > 1$) spherical triangles (the $[6^3]$ tiling).

Fekete and Davis [Feke84] let $n = 2$, which means that at each level of decomposition, three new vertices are generated by halving each side of the triangle; connecting them together yields four triangles. They use the term *property sphere* to describe their representation. The property sphere has been used in object recognition; it is also of potential use in mapping the globe because it can enable accurate modeling of regions around the poles. For example, see Figure 1.16, which is a property sphere representation of some spherical data. In contrast, planar quadtrees are less attractive the farther we get from the equator due to distortions in planarity caused by the earth's curvature. Of course, for true applicability for mapping, we need a closer approximation to a sphere than is provided by the 20 triangles of the icosahedron. Moreover, we want a way to distinguish between different elevations.

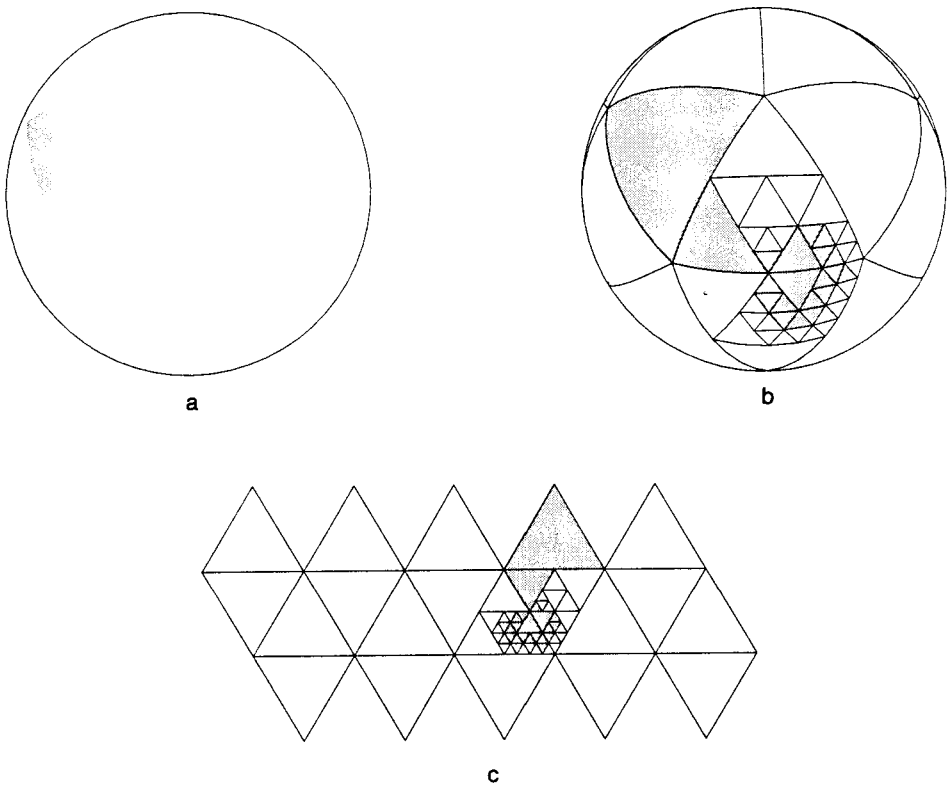


Figure 1.16 Property sphere representation of some spherical data: (a) data, (b) decomposition on a sphere, (c) decomposition on a plane

Dutton [Dutt84] lets $n = \sqrt{3}$, which means that at each level of decomposition, one new vertex is created by connecting the centroid of the triangle to its vertices. The result is an alternating sequence of triangles so that each level is fully contained in the level that was created two steps previously and has nine times as many triangles as that level. Dutton uses the term *triacon* to describe the resulting hierarchy. As an example, consider Figure 1.17, which illustrates four levels of a triacon decomposition. The initial and odd-numbered decompositions are shown with heavy lines, and the even-numbered decompositions are shown with broken and thin lines.

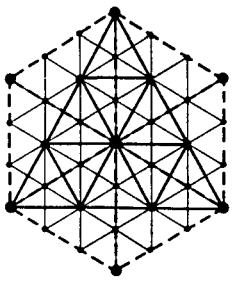


Figure 1.17 Example of a triacon hierarchy

The icosahedron is not the only regular polyhedron that can be used to model spherical data. Others include the tetrahedron, hexahedron, octahedron, and dodecahedron, which have 4, 6, 8, and 12 faces, respectively. Collectively these five polyhedra are known as the *Platonic solids* [Peuq84]. The faces of the tetrahedron and octahedron are equilateral triangles, while the faces of the hexahedron and dodecahedron are squares and regular pentagons, respectively.

The dodecahedron is not an appropriate primitive because the pentagonal faces cannot be further decomposed into pentagons or other similar shapes. The tetrahedron and hexahedron (the basis of the octree) have internal angles that are too small to model a sphere properly, thereby leading to shape distortions.

Dutton [Dutt84] points out that the octahedron is attractive for modeling spherical data such as the globe because it can be aligned so that the poles are at opposite vertices and the prime meridian and the equator intersect at another vertex. In addition, one subdivision line of each face is parallel to the equator. Of course, for all of the Platonic solids, only the vertices of the solids touch the sphere; the facets of the solids are interior to the sphere.

Other decompositions for spherical data are also possible. Tobler and Chen [Tobl86] point out the desirability of a close relationship to the commonly used system of latitude and longitude coordinates. In particular, any decomposition that is chosen should enable the use of meridians and parallels to refer to the data. An additional important goal is for the partition to be into units of equal area, which rules out the use of equally spaced lines of latitude (of course, the lines of longitude are equally spaced). In this case, the sphere is projected into a plane using Lambert's cylindrical projection [Adam49], which is locally area preserving. Authalic coordinates [Adam49], which partition the projection into rectangles of equal area, are then derived. (For more details, see [Tobl86].)

The quadtree decomposition has the property that at each subdivision stage, the image is subdivided into four equal-sized parts. When the original image is a square, the result is a collection of squares, each of which has a side whose length is a power of 2. The binary image tree (termed *bintree*) [Know80, Tamm84a, Same88b] is an alternative decomposition defined in a manner analogous to the region quadtree except that at each subdivision stage we subdivide the image into two equal-sized parts. In two dimensions, at odd stages, we partition along the x coordinate, and at even stages, along the y coordinate. The bintree is equivalent to the region quadtree if we replace all leaf nodes at odd stages of subdivision by two identically colored sons.

The bintree is related to the region quadtree in the same way as the k-d tree [Bent75b] (see Section 2.4) is related to the point quadtree [Fink74]. The difference is that region quadtrees and bintrees are used to represent region data with fixed subdivision points, while point quadtrees and k-d trees are used to represent point data where the values of the points determine the subdivision. For example, Figure 1.18 is the bintree representation corresponding to the image of Figure 1.1. We assume that for the x (y) partition, the left subtree corresponds to the west (south) half of the image and the right subtree corresponds to the east (north) half. Once again, as in Figure 1.1, all leaf nodes are labeled with numbers, and the nonleaf nodes are labeled with letters.

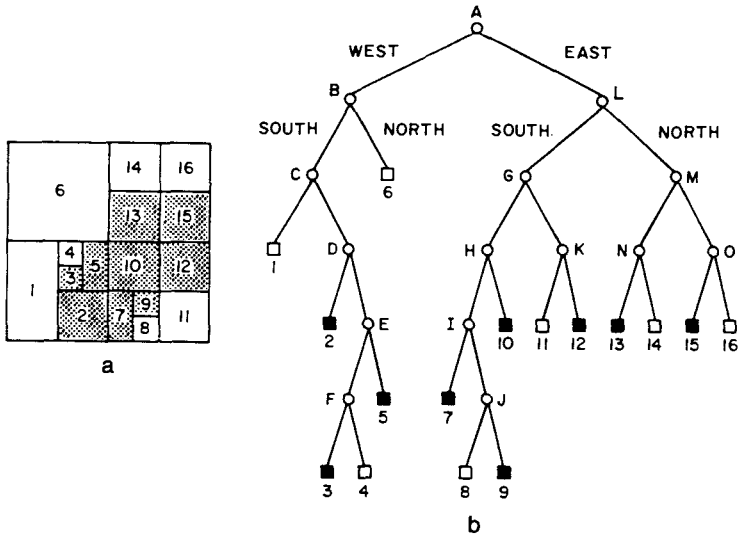


Figure 1.18 Bintree representation corresponding to Figure 1.1: (a) block decomposition, (b) bintree representation of blocks in (a)

The quadtree and bintree decompose a region into equal-sized parts. Kanatani [Kana85] suggests using splitting rules based on the Fibonacci sequence of numbers. The Fibonacci numbers consist of the sequence of numbers f_i that satisfy the relation $f_i = f_{i-1} + f_{i-2}$, with $f_0 = 1$ and $f_1 = 1$. We can try to devise both quadtree and bintree splitting rules based on such a sequence. Generally for a decomposition scheme to be useful in geometric applications, it must have pixel-sized squares (i.e., 1×1) as the primitive tiles. At first glance, it appears that the Fibonacci sequence gives quite a bit of leeway in deciding on a splitting sequence and on the sizes of the regions corresponding to the subtrees and the primitive tiles.

One possible quadtree splitting rule is to restrict all shapes to squares with sides whose lengths are Fibonacci numbers. Clearly not all the shapes can be squares since we cannot aggregate these squares into larger squares that obey this rule. Another possibility is to restrict the shapes to rectangles the length of whose sides are either equal Fibonacci numbers or are successive Fibonacci numbers (see Exercise 1.26). We term this condition the *2-d Fibonacci condition.*

In this discussion, we have assumed splitting rules that ensure that vertical subdivision lines at the same level are colinear as well as for horizontal lines at the same level. For example, when using a quadtree splitting rule, the vertical lines that subdivide the NW and SW quadrants are colinear, as well as for the horizontal lines that subdivide the NW and NE quadrants. An alternative is to relax the colinearity restriction; however, the sides of the shapes must still satisfy the 2-d Fibonacci condition (see Exercise 1.27).

As can be seen in Exercises 1.26 and 1.27, neither a quadtree nor a bintree can

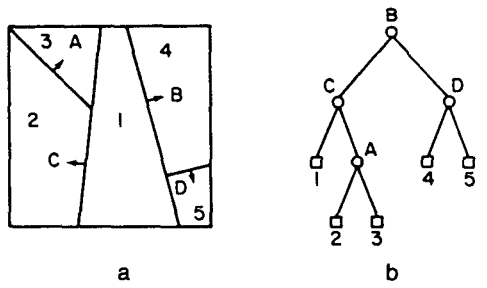


Figure 1.19 (a) An arbitrary space decomposition and (b) its BSP tree. The arrows indicate the direction of the positive halfspaces.

be used by itself as a basis for Fibonacci-based space decomposition; however, a combination of the two structures could be used. When the lengths of the sides of a rectangle are equal, the rectangle is split into four rectangles such that the lengths of the sides satisfy the 2-d Fibonacci condition. When the lengths of the sides of a rectangle are not equal, the rectangle is split into two rectangles with the split along a line (an axis) parallel to the shorter (longer) of the two sides. Interestingly the dimensions of the A-series of European paper are based on a Fibonacci sequence—that is, the elements of the series are of dimension $f_i \times f_{i-1}$ multiplied by an appropriate scale factor.

Another variation on the bintree idea, termed *adaptive hierarchical coding* (*AHC*), is proposed by Cohen, Landy, and Pavel [Cohe85b]. In this case, the image is again split into two equal-sized parts at each stage, but there is no need to alternate between the x and y coordinates. The decision as to the coordinate on which to partition depends on the image. This technique may require some work to get the optimal partition from the point of view of a minimum number of nodes (see Exercise 1.29).

An even more general variation on the bintree is the *BSP* tree of Fuchs, Kedem, and Naylor [Fuch80, Fuch83]. Its variants are used in some hidden-surface elimination algorithms (see Section 7.1.5 of [Same90b]) and in some implementations of beam tracing (see Section 7.3 of [Same90b]). It is applicable to data of arbitrary dimension, although here it is explained in the context of two-dimensional data. At each subdivision stage, the image is subdivided into two parts of arbitrary size. Note that successive subdivision lines need be neither orthogonal nor parallel. Therefore the resulting decomposition consists of arbitrarily shaped convex polygons.

The BSP tree is a binary tree. To be able to assign regions to the left and right subtrees, we associate a direction with each subdivision line. In particular, the subdivision lines are treated as separators between two halfspaces.[9] Let the line have the

[9] A (linear) *halfspace* in d-space is defined by the inequality $\sum_{i=0}^{d} a_i \cdot x_i \geq 0$ on the $d+1$ homogeneous coordinates ($x_0 = 1$). The halfspace is represented by a column vector a. In vector notation, the inequality is written as $a \cdot x \geq 0$. In the case of equality, it defines a hyperplane with a as its normal. It is important to note that halfspaces are volume, not boundary, elements.

equation $a \cdot x + b \cdot y + c = 0$. We say that the right subtree is the 'positive' side and contains all subdivision lines formed by separators that satisfy $a \cdot x + b \cdot y + c \geq 0$. Similarly we say that the left subtree is 'negative' and contains all subdivision lines formed by separators that satisfy $a \cdot x + b \cdot y + c < 0$. As an example, consider Figure 1.19a, which is an arbitrary space decomposition whose BSP tree is given in Figure 1.19b. Notice the use of arrows to indicate the direction of the positive halfspaces.

Exercises

1.17. Given a $[6^3]$ tiling such that each side of an atomic tile has a unit length, compute the three adjacency distances from the centroid of an atomic tile.

1.18. Repeat Exercise 1.17 for $[3^6]$ and $[4^4]$, again assuming that each side of an atomic tile has a unit length.

1.19. Suppose that you are given an image in the form of a binary array of pixels. The result is a square grid. How can you view this grid as a hexagonal grid?

1.20. Show how the property sphere data structure can be used to model the earth. In particular, discuss how to represent landmass features, such as mountain ranges and crevices.

1.21 Suppose that you use an icosahedron to model spherical data. Initially there are 20 faces. How many faces are there after the first level of decomposition when $n = 2$? $n = \sqrt{3}$?

1.22. What is the ratio of leaf nodes to nonleaf nodes in a bintree for a d-dimensional image?

1.23. What is a lower bound on the ratio of leaf nodes in a bintree to that in a quadtree for a d-dimensional image? What is an upper bound? What is the average?

1.24. Is it true that the total number of nodes in a bintree is always less than that in the corresponding quadtree?

1.25. The Fibonacci numbers are defined by the relation $f_n = f_{n-1} + f_{n-2}$. Devise a two-dimensional analog of this relation to correspond to a splitting rule that would have to be satisfied in a Fibonacci-based space decomposition that yields four parts. Generalize this result to n dimensions.

1.26. Give a counterexample to the use of a quadtree splitting rule in a Fibonacci-based space decomposition.

1.27. Give a counterexample to the use of a bintree splitting rule in a Fibonacci-based space decomposition.

1.28. Suppose that you use the combination quadtree-bintree approach to a Fibonacci-based space decomposition. Prove that any image such that the lengths of its sides satisfy the 2-d Fibonacci condition can be decomposed into subimages whose sides obey this property and with a primitive tile of size 1×1.

1.29. Suppose that you use the AHC method. How many different rectangles and positions must be examined in building such a structure for a $2^n \times 2^n$ image?

1.4.2 Nonpolygonal Tilings

In the previous section we focused on space decompositions based on polygonal tiles. This is the prevalent method in use today. For certain applications, however, the use of polygonal tiles can lead to problems. For example, suppose that we have a decomposition based on square tiles. In this case, as the resolution is increased, the area of the approximated region approaches the true value of the area; however, this is not

true for a boundary measure such as the perimeter. To see this, consider a quadtree approximation of an isosceles right triangle where the ratio of the approximated perimeter to the true perimeter is $4/(2+\sqrt{2})$ (see Exercise 1.30). Other problems include the discontinuity of the normals to the boundaries of adjacent tiles.

There are a number of ways of attempting to overcome these problems. The *hierarchical probe model* of Chen [Chen85b] is an approach based on treating space as a polar plane and recursively decomposing it into sectors. We say that each sector consists of an origin, two sides (labeled 1 and 2 corresponding to the order in which they are encountered when proceeding in a counterclockwise direction), and an arc. The points at which the sides of the sector intersect (or touch) the object are called *contact points*. (ρ,θ) denotes a point in the polar plane. Let (ρ_i, θ_i) be the contact point with the maximum value of ρ in direction θ_i. Each sector represents a region bounded by the points (0,0), (ρ_1,θ_1), and (ρ_2,θ_2), where $\theta_1 = 2k\pi/2^n$ and $\theta_2 = \theta_1 + 2\pi/2^n$ such that k and n are nonnegative integers ($k < 2^n$). The arc between the two nonorigin contact points (ρ_1, θ_1) and (ρ_2, θ_2) of a sector is approximated by the linear parametric equations ($0 \le t \le 1$):

$$\rho(t) = \rho_1 + (\rho_2 - \rho_1) \cdot t \qquad \theta(t) = \theta_1 + (\theta_2 - \theta_1) \cdot t.$$

Note that the interpolation curves are arcs of spirals due to the linear relation between ρ and θ.

The *sector tree* is a binary tree that represents the result of recursively subdividing sectors in the polar plane into two sectors of equal angular intervals. Thus the recursive decomposition is only with respect to θ, not ρ. The decomposition stops whenever the approximation of a part of an object by a sector is deemed to be adequate. The computation of the stopping condition is implementation dependent. For example, it can be the maximum deviation in the value of ρ between a point on the boundary and the corresponding point (i.e., at the same value of θ) on the approximating arc. Initially the universe is the interval $[0,2\pi)$.

In the presentation, we assume that the origin of the polar plane is contained within the object. See Exercise 1.36 for a discussion of how to represent an object that does not contain the origin of the polar plane. The simplest case arises when the object is convex. The result is a binary tree where each leaf node represents a sector and contains the contact points of its corresponding arc. For example, consider the object in Figure 1.20. The construction of its sector tree approximation is shown in

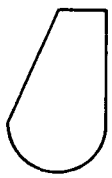


Figure 1.20 Example convex object

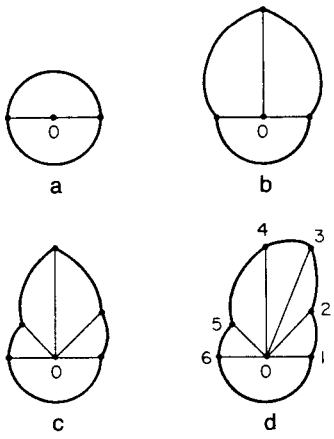


Figure 1.21 Successive sector tree approximations for the object of Figure 1.20: (a) π intervals, (b) π/2 intervals, (c) π/4 intervals, (d) π/8 intervals

Figures 1.21a–d. The final binary tree is given in Figure 1.22 with interval endpoints labeled according to Figure 1.21d.

The situation is more complex when the object is not convex. This means that each side of a sector may intersect the boundary of the object at an arbitrary, and possibly different, number of contact points. In the following, each sector will be seen to consist of a set of alternating regions within and outside the object. These regions are three-sided or four-sided and have at least one side that is colinear with a side of the sector. The discussion is illustrated with the object of Figure 1.23a whose sector tree decomposition is given in Figure 1.23b. The final binary tree is given in Figure 1.24. A better indication of the quality of the approximation can be seen by examining Figure 1.23c, which contains an overlay of Figures 1.23a and 1.23b.

When the boundary of the object intersects a sector at two successive contact points, say P and Q, that lie on the same side, say S, of the sector, then the region

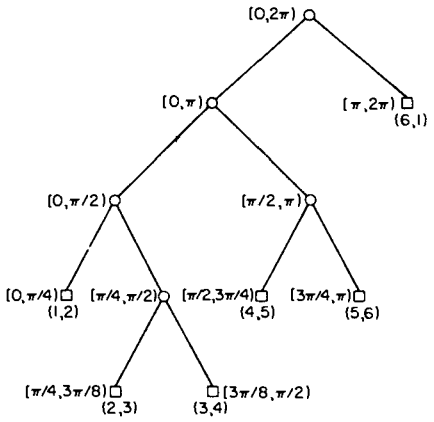


Figure 1.22 Binary tree representation of the sector tree of Figure 1.20

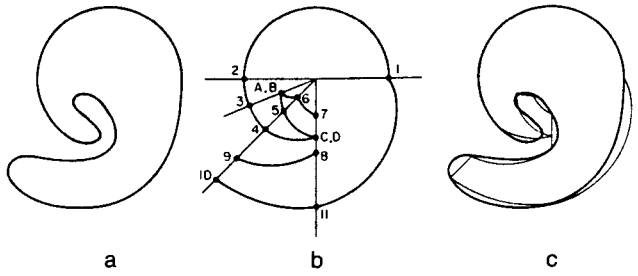


Figure 1.23 (a) Example object, (b) its sector tree description, and (c) a comparison of the sector tree approximation (thin lines) with the original object (thick lines). Note the creation of a hole corresponding to the region formed by points A, B, 6, 7, C, D, and 5

bounded by S and PQ must be approximated. Without loss of generality, assume that the region is inside the object. There are two choices. An inner approximation ignores the region by treating the segment of S between P and Q as part of the approximated boundary (e.g., the region between points 9 and 10 in sector $[9\pi/8, 5\pi/4)$ in Figure 1.23b).

An outer approximation inserts two identical contact points, say R and T, on the other side of the sector and then approximates the region by the three-sided region formed by the segment of S between P and Q and the spiral arc approximations of PR and QT. The value of R (and hence T) is equal to the average of the value of ρ at P and Q. For example, the region between points 4 and 5 in sector $[5\pi/4, 3\pi/2)$ in Figure 1.23b is approximated by the region formed with points C and D.

Of course, the same approximation process is applied to the part of the region outside the object. In Figure 1.23b, we have an inner approximation for the region between points 7 and 8 in sector $[3\pi/2, 2\pi)$, and an outer approximation for the region between points 5 and 6 in sector $[9\pi/8, 5\pi/4)$, by virtue of the introduction of points A and B.

One of the problems with the sector tree is that its use can lead to the creation of holes that do not exist in the original object. This situation arises when the decomposition is not carried out to a level of sufficient depth. For example, consider Figure 1.23b, which has a hole bounded by the arcs formed by points A, B, 6, 7, C, D, and 5. This is a result of the inner approximation for the region between points 7 and 8 in sector $[3\pi/2, 2\pi)$ and an outer approximation for the region between points 4 and 5 in sector $[5\pi/4, 3\pi/2)$. This situation can be resolved by further decomposition in either or both of sectors $[3\pi/2, 2\pi)$ and $[5\pi/4, 3\pi/2)$.

The result of the approximation process is that each sector consists of a collection of three-sided and four-sided regions that approximate the part of the object contained in the sector. This collection is stored in the leaf node of the sector tree as a list of pairs of points in the polar plane. It is interesting to observe that the boundaries of the interpolated regions are not stored explicitly in the tree. Instead each pair of points corresponds to the boundary of a region. Since the origin of the polar plane is within the object, an odd number of pairs of points is associated with each leaf node. For

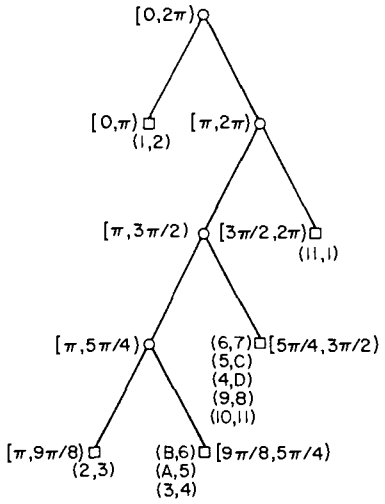


Figure 1.24 Binary tree representation of the sector tree of Figure 1.23

example, consider the leaf node in Figure 1.24 corresponding to the sector $[5\pi/4, 3\pi/2)$. The first pair, together with the origin, defines the first region (e.g., (6,7)). The next two pairs of points define the second region (e.g., (5,C) and (4,D)), with each successive two pairs of points defining the remaining regions.

The sector tree is a partial polar decomposition, as the subdivision process is based only on the value of θ. A total polar decomposition would partition the polar plane on the basis of both ρ and θ. The result is analogous to a quadtree, and it is termed a *polar quadtree*. There are a number of possible rules for the decomposition process (see Exercise 1.42). For example, consider a decomposition that recursively halves both ρ and θ at each level. In general, the polar quadtree is a variant of a maximal block representation. As in the sector tree, the blocks are disjoint. Unlike the sector tree, blocks in the polar quadtree do have standard sizes. In particular, all blocks in the polar quadtree are either three sided (i.e., sectors) or four sided (i.e., quadrilaterals, two of whose sides are arcs). Thus the sides of polar quadtree blocks are not based on interpolation.

The primary motivation for presenting the sector tree is to show that space decompositions could also be based on nonpolygonal tiles. In the rest of this book the primary concern is with space decompositions based on rectangles (especially squares) and showing how a number of operations can be performed when they serve as the underlying representation. The techniques are quite general and can be applied to most space decomposition methods. Thus the sector tree is not discussed further except in the context of its adaptation to the representation of three-dimensional data (see Section 5.6). Nevertheless, the following contains a brief mention of some of the operations to which the sector tree lends itself.

Set operations such as union and intersection are straightforward. Scaling is trivial as the sector tree need not be modified; all values of ρ are interpreted as scaled

by the appropriate scale factor. The number of nodes in a sector tree is dependent on its orientation—that is, on the points chosen as the origin and the contact point chosen to serve as (ρ,0). Rotation is not so simple; it cannot be implemented by simply rearranging pointers (but see Exercise 1.40). Translation is computationally expensive since the change in the relative position of the object with respect to the origin means that the entire sector tree must be reconstructed.

Exercises

1.30. Prove that for an isosceles right triangle represented by a region quadtree, the ratio of the approximated perimeter to the true perimeter is $4/(2+\sqrt{2})$.

1.31. Repeat Exercise 1.30 for a circle (i.e., find the ratio).

1.32. When the objects have linear sides, polygonal tiles are superior. How would you use the sector tree decomposition method with polygonal tiles?

1.33. In the discussion of the situation arising when the boundary of the object intersects a sector at two successive contact points, say P and Q, that lie on the same side, say S, of the sector, we assumed that the region bounded by S and PQ was inside the object. Suppose that this region is outside the object. How does this affect the inner and outer approximations?

1.34. Can you traverse the boundary of an object represented by a sector tree by visiting each leaf node just once?

1.35. When using a sector tree, how would you handle the situation that the boundary of the object just touches the side of a sector without crossing it (i.e., a tangent if the boundary is differentiable)?

1.36. How would you use a sector tree to represent an object that does not contain the origin of the polar plane?

1.37. The outer approximation used in building a sector tree always yields a three-sided region. Two of the sides are arcs of spirals with respect to a common origin. This implies a sharp discontinuity of the derivative at the point at which they meet. Can you devise a way to smoothe this discontinuity?

1.38. Does the inner approximation used in building a sector tree always underestimate the area? Similarly does the outer approximation always overestimate the area?

1.39. Compare the inner and outer approximations used in building a sector tree. Is there ever a reason for the outer approximation to be preferred over the inner approximations (or vice-versa)?

1.40. Define a complete sector tree in an analogous manner to a complete binary tree—that is, all leaf nodes are at the same level, say n. Prove that a complete sector tree is invariant under rotation in multiples of $2\pi/2^n$.

1.41. Write an algorithm to trace the boundary of an object represented by a sector tree.

1.42. Suppose that it is desired to decompose space into nonpolygonal shapes. Develop a quadtree-like data structure based on polar coordinates (i.e., ρ and θ). Investigate different splitting rules for polar quadtrees. In particular, you do not need to alternate the splits—that is, you could split on ρ several times in a row, and so on. This technique is used in the adaptive k–d tree [Frie77] (see Section 2.4.1) by decomposing the quartering process into two splitting operations—one for the x coordinate and one for the y coordinate. What are the possible shapes for the quadrants of such trees (e.g., a torus, doughnut, wheels with spokes)?

1.5 SPACE REQUIREMENTS

The primary motivation for the development of the quadtree was the desire to reduce the amount of space necessary to store data through the use of aggregation of homogeneous blocks. As we will see in subsequent chapters, an important by-product of this aggregation is the reduction of the execution time of a number of operations (e.g., connected component labeling, component counting). However, a quadtree implementation does have overhead in terms of the nonleaf nodes. For an image with B and W black and white blocks, respectively, $4 \cdot (B + W)/3$ nodes are required. In contrast, a binary array representation of a $2^n \times 2^n$ image requires only 2^{2n} bits; however, this quantity grows quite quickly. Furthermore, if the amount of aggregation is minimal (e.g., a checkerboard image), the quadtree is not very efficient.

The overhead for the nonleaf nodes can be reduced at times by using a pointer-less representation. Pointer-less representations can be grouped into two categories. The first, termed a *DF-expression*, represents the quadtree as a traversal of its constituent nodes [Kawa80a]. For example, letting 'B', 'W', and 'G' correspond to black, white, and gray nodes, respectively, and assuming a traversal in the order NW, NE, SW, and SE, the quadtree of Figure 1.1 would be represented by GWGWWBBGWGW BBBWBGBBGBBBWW.

The second approach treats the quadtree as a collection of the leaf nodes comprising it. Each node is represented by a pair of numbers [Garg82c]. The first number is the level of the tree at which the node is located. The second number is termed a *locational code*. It is formed by a concatenation of base 4 digits corresponding to directional codes that locate the node along a path from the root of the quadtree. The directional codes take on the values 0, 1, 2, 3 corresponding to quadrants NW, NE, SW, SE, respectively. For example, node 15 in Figure 1.1 is represented by the pair of numbers (0,320), which is decoded as follows. The base 4 locational code is 320. The pair denotes a node at level 0 that is reached by a sequence of transitions, SE, SW, and NW, starting at the root. A quadtree representation based on the use of locational codes is called *linear quadtree* by Gargantini [Garg82a, Garg82c] (because the addresses are keys in a linear list of nodes). Pointer-less representations are discussed in greater detail in Chapter 2 of [Same90b].

The worst case for a quadtree of a given depth in terms of storage requirements occurs when the region corresponds to a checkerboard pattern as in Figure 1.25. The amount of space required is obviously a function of the resolution (i.e., the number of levels in the quadtree), the size of the image (i.e., its perimeter), and its positioning in the grid within which it is embedded. As a simple example, Dyer [Dyer82] has shown that arbitrarily placing a square of size $2^m \times 2^m$ at any position in a $2^n \times 2^n$ image requires an average of $O(2^{m+2} + n - m)$ quadtree nodes. An alternative characterization of this result is that the average amount of space necessary is $O(p + n)$ where p is the perimeter (in pixel widths) of the block.

Dyer's $O(p + n)$ result for a square image is merely an instance of the earlier work of Hunter and Steiglitz [Hunt78, Hunt79a] who proved some fundamental theorems on the space requirements of images represented by quadtrees. In their

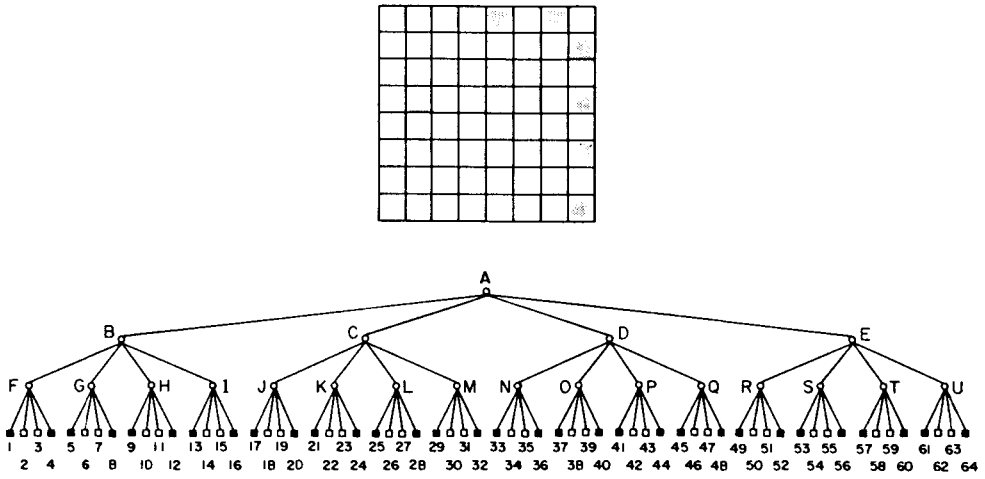


Figure 1.25 A checkerboard and its quadtree

studies, Hunter and Steiglitz used simple polygons (polygons with nonintersecting edges and without holes); however, these theorems have been observed to hold in arbitrary images (see [Rose82b] for empirical results in a cartographic environment).

In Hunter and Steiglitz's formulation, a polygon is represented by a three-color variant of the quadtree. In essence, there are three types of nodes: interior, boundary, and exterior. A node is said to be of type *boundary* if an edge of the polygon passes through it. *Interior* and *exterior* nodes correspond to areas within, and outside, respectively, the polygon and can be merged to yield larger nodes. The resulting quadtree is analogous to the MX quadtree representation of point data described below (for more details, see Section 2.6.1), and this term will be used to describe it. In particular, boundary nodes are analogous to black nodes, while interior and exterior nodes are analogous to white nodes.

Figure 1.26 illustrates a sample polygon and its MX quadtree. One disadvantage of the MX quadtree representation for polygonal lines is that a width is associated with them, whereas in a purely technical sense these lines have a width of zero. Also shifting operations may result in information loss. (For more appropriate representations of polygonal lines, see Chapter 4.)

An upper bound on the number of nodes in such a representation of a polygon can be obtained in the following manner. First, we observe that a curve of length $d + \varepsilon$ ($\varepsilon > 0$) can intersect at most six squares of side width d. Now consider a polygon, say G, having perimeter p, that is embedded in a grid of squares each of side width d. Mark the points at which G enters and exits each square. Choose one of these points, say P, as a starting point for a decomposition of G into a sequence of curves. Define the first curve in G to be the one extending from P until six squares have been intersected and a crossing is made into a different seventh square. This is the starting point for another curve in G that intersects six new squares, not counting those intersected by any previous curve.

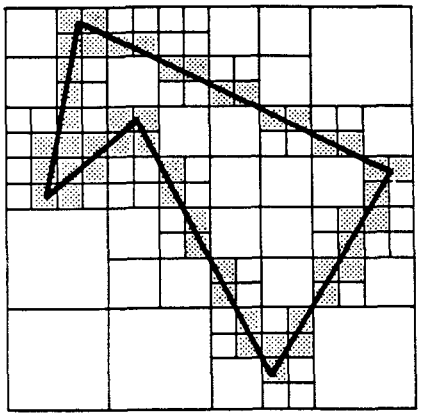


Figure 1.26 Hunter and Steiglitz's quadtree representation of a polygon

We now decompose G into a series of such curves. Since each curve adds at most six new squares and has length of at least d, we see that a polygon with perimeter p cannot intersect more than $6 \cdot \lceil p/d \rceil$ squares. Given a quadtree with a root at level n (i.e., the grid of squares is of width 2^n), at level i each square is of width 2^i. Therefore polygon G cannot intersect more than $B(i) = 6 \cdot \lceil p/2^i \rceil$ quadrants at level i. Recall that our goal is to derive an upper bound on the total number of nodes. This bound is attained when each boundary node at level i has three brother nodes that are not intersected. Of course, only boundary nodes can have sons, and thus no more than $B(i)$ nodes at level i have sons. Since each node at level i is a son of a node at level $i+1$, there are at most $4 \cdot B(i+1)$ nodes at level i. Summing up over n levels (accounting for a root node at level n and four sons), we find that the total number of nodes in the tree is bounded by

$$1 + 4 + \sum_{i=0}^{n-2} 4 \cdot B(i+1)$$

$$\leq 5 + 24 \cdot \sum_{i=0}^{n-2} \lceil \frac{p}{2^{i+1}} \rceil$$

$$\leq 5 + 24 \cdot \sum_{i=0}^{n-2} (1 + \frac{p}{2^{i+1}})$$

$$\leq 5 + 24 \cdot (n-1) + 24 \cdot p \cdot \sum_{i=0}^{n-2} \frac{1}{2^{i+1}}$$

$$\leq 24 \cdot n - 19 + 24 \cdot p.$$

Therefore, we have proved:

Theorem 1.1 The quadtree corresponding to a polygon with perimeter p embedded in a $2^n \times 2^n$ image has a maximum of $24 \cdot n - 19 + 24 \cdot p$ (i.e., $O(p+n)$) nodes. □

The proof of Theorem 1.1 is based on a decomposition of the polygon into a sequence of curves, each of which intersects at most six squares. This bound can be tightened by examining patterns of squares to obtain minimum lengths and corresponding ratios of possible squares per unit length. For example, observe that once a curve intersects six squares, the next curve of length d in the sequence can intersect at most two new squares. In contrast, it is easy to construct a sequence of curves of length $d + \varepsilon$ ($\varepsilon > 0$) such that almost each curve intersects two squares of side length d. Such a construction leads to an upper bound of the form $a \cdot n + b + 8 \cdot p$ where a and b are constants (see Exercise 1.48). Hunter and Steiglitz use a slightly different construction to obtain a bound of $16 \cdot n - 11 + 16 \cdot p$ (see Exercise 1.49).

Nevertheless, the bound of Theorem 1.1 is attainable as demonstrated by the following examples. First, consider a square of side width 2 that consists of the central four squares in a $2^n \times 2^n$ image (see Figure 1.27). Its quadtree has $16 \cdot n - 11$ nodes (see Exercise 1.50). Second, consider a curve that follows a vertical line through the center of a $2^n \times 2^n$ image. Now, make it a bit longer by making it intersect all of the pixels on either side of the vertical line (see Figure 1.28). As n increases, the total number of nodes in the quadtree approaches $8 \cdot p$ where $p = 2^n$ (see Exercise 1.51). A polygon having a number of nodes approaching $8 \cdot p$ can be constructed in a similar manner by approximating a square in the center of the image whose side is one-fourth the side of the image (see Exercise 1.52). In fact, it has been shown by Hunter [Hunt78] that $O(p+n)$ is a least upper bound on the number of nodes in a quadtree corresponding to a polygon (see Exercise 1.53).

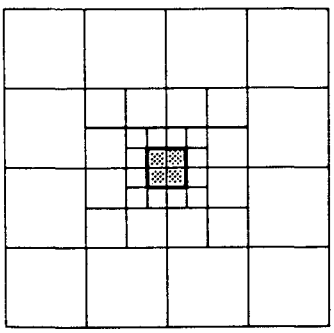


Figure 1.27 Example quadtree with $16 \cdot n - 11$ nodes

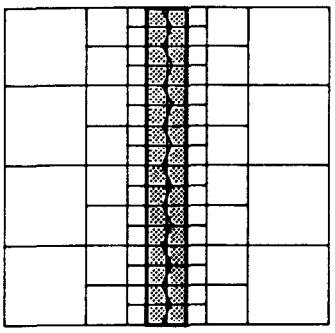


Figure 1.28 Example quadtree with approximately 8 · p nodes

Theorem 1.1 can be recast by measuring the perimeter p in terms of the length of a side of the image in which the polygon is embedded—i.e., for a $2^n \times 2^n$ image $p = p' \cdot 2^n$. Thus the value of the perimeter no longer depends on the resolution of the image. Restating Theorem 1.1 in terms of p' results in a quadtree having $O(p' \cdot 2^n + n)$ nodes. This leads to the following important corollary:

> **Corollary 1.1** The maximum number of nodes in a quadtree corresponding to an image is directly proportional to the resolution of the image. □

The significance of Corollary 1.1 is that when using quadtrees, increasing the image resolution leads to a linear growth in the number of nodes. This is in contrast to the binary array representation where doubling the resolution leads to a quadrupling of the number of pixels.

Since in most practical cases the perimeter, p, dominates the resolution, n, the results of Theorem 1.1 are usually interpreted as stating that the number of nodes in a quadtree is proportional to the perimeter of the regions contained therein.[10] Meagher [Meag80] has shown that this theorem also holds for three-dimensional data (i.e., for polyhedra represented by octrees) when the perimeter is replaced by the surface area. The perimeter and the surface area correspond to the size of the boundary of the polygon and polyhedron—that is, in two and three dimensions, respectively. In d dimensions this result can be stated as follows:

> **Theorem 1.2:** The size of a d-dimensional quadtree of a d-dimensional polyhedron is proportional to the sum of the resolution and the size of the boundary of the object. □

[10] Of course, the storage used by runlength codes is also proportional to the perimeter of the regions. However, runlength codes do not facilitate access to different parts of the regions (i.e., they have poor spatial indexing properties).

Aside from their implications on the storage requirements, Theorems 1.1 and 1.2 also directly affect the analysis of the execution time of algorithms. In particular, most algorithms that execute on a quadtree representation of an image instead of an array representation have an execution time proportional to the number of blocks in the image rather than the number of pixels. In its most general case, this means that the application of a quadtree algorithm to a problem in d-dimensional space executes in time proportional to the analogous array-based algorithm in the $(d-1)$-dimensional space of the surface of the original d-dimensional image. Thus quadtrees are somewhat like dimension-reducing devices.

Theorem 1.2 assumes that the image consists of a polyhedron. Walsh [Wals85] lifts this restriction and obtains a weaker complexity bound. Assuming an image of resolution n and measuring the perimeter, say p, in terms of the number of border pixels, he proves that the total number of nodes in a d-dimensional quadtree is less than or equal to $4 \cdot n \cdot p$. Furthermore he shows that the number of black nodes is less than or equal to $(2^d - 1) \cdot n \cdot p/d$.

The complexity measures discussed above do not explicitly reflect the fact that the amount of space occupied by a quadtree corresponding to a region is extremely sensitive to its orientation (i.e., where it is partitioned). For example, in Dyer's experiment, the number of nodes required for the arbitrary placement of a square of size $2^m \times 2^m$ at any position in a $2^n \times 2^n$ image ranged between $4 \cdot (n-m) + 1$ and $4 \cdot p + 16 \cdot (n-m) - 27$, with the average being $O(p+n-m)$. Clearly shifting the image within the space in which it is embedded can reduce the total number of nodes. The problem of finding the optimal position for a quadtree can be decomposed into two parts. First, we must determine the optimal grid resolution and, second, the partition points.

Grosky and Jain [Gros83] have shown that for a region such that w is the maximum of its horizontal and vertical extent (measured in pixel widths) and $2^{n-1} < w \leq 2^n$, the optimal grid resolution is either n or $n+1$. In other words embedding the region in a larger area than $2^{n+1} \times 2^{n+1}$ and shifting it around will not result in fewer nodes. Using similar reasoning, it can be shown that translating a region by 2^k pixels in any direction does not change the number of black or white blocks of size less than $2^k \times 2^k$ [Li82].

Armed with the above results, Li, Grosky, and Jain [Li82] developed the following algorithm that treats the image as a binary array and finds the configuration of the region in the image so that its quadtree requires a minimum number of nodes. First, enlarge the image to be $2^{n+1} \times 2^{n+1}$, and place the region within it so that the region's northernmost and westernmost pixels are adjacent to the northern and western borders, respectively, of the image. Next apply successive translations to the image of magnitude power of two in the vertical, horizontal, and corner directions and keep count of the number of leaf nodes required. Initially 2^{2n+2} leaf nodes are necessary. The following is a more precise statement of the algorithm:

1. Attempt to translate the image by (x,y) where x and y correspond to unit translations in the horizontal and vertical directions, respectively. Each of x and y takes on the values 0 or 1.

2. For the result of each translation in step 1, construct a new array at one-half the resolution. Each entry in the new array corresponds to a 2×2 block in the translated array. For each entry in the new array that corresponds to a single color (not gray) 2×2 block in the translated array, decrement the leaf node count by 3.
3. Recursively apply steps 1 and 2 to each result of steps 1 and 2. This process stops when no single-color 2×2 block is found in step 2 (i.e., they are all gray) or if the new array is a 1×1 block. Record the total translation and the minimum leaf node count.

Step 2 makes use of the property that for a translation of 2^k, there is a need to check only if single-color blocks of size $2^k \times 2^k$ or more are formed. In fact, because of the recursion, at each step we check only for the formation of blocks of size $2^{k+1} \times 2^{k+1}$. Note that the algorithm tries every possible translation since any integer can be decomposed into a summation of powers of two (i.e., use its binary representation). In fact this is why a translation of (0,0) is part of step 1. Although the algorithm computes the positioning of the quadtree with the minimum number of leaf nodes, it is also the positioning of the quadtree with the minimum total number of nodes since the number of nonleaf nodes in a quadtree of T leaf nodes is $(T-1)/3$.

As an example of the algorithm, consider the region given in Figure 1.29a whose block decomposition is shown in Figure 1.29b. Its quadtree requires 52 leaf nodes. The first step is to enlarge the image, place the region in the upper left corner, and form the array (Figure 1.30). The optimal positioning is such that Figure 1.30 is shifted 7 units in the horizontal direction and 3 units in the vertical direction. This corresponds to a sequence of translations (1,1), (1,1), and (1,0). The intermediate translated arrays are shown in Figure 1.31. All gray nodes in the translated arrays are labeled with a 'G' while black nodes are shaded. The optimal quadtree contains 46 leaf nodes and is given in Figure 1.32.

Now let us trace the algorithm as it applies the optimal sequence of translations, in more detail. Initially the leaf node count is 256. A translation of (1,1) leads to Figure 1.31a where 58 of the array entries correspond to single-color 2×2 blocks in the translated array. The leaf node count is decremented by $58 \cdot 3 = 174$, resulting in

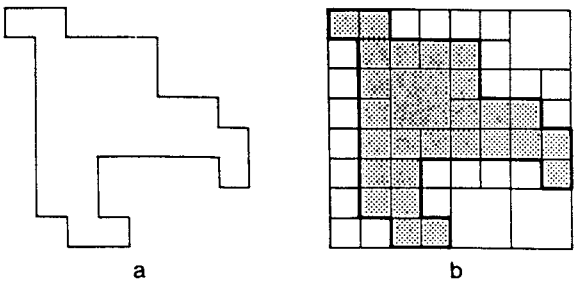


Figure 1.29 Example (a) image and (b) its block decomposition used to demonstrate the optimal positioning process

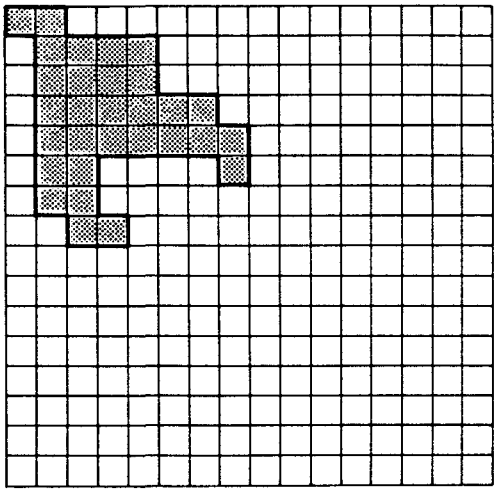


Figure 1.30 The array corresponding to the image in Figure 1.29 prior to the start of the optimal positioning process

82. The next translation of (1,1) leads to Figure 1.31b, where 11 of the array entries correspond to single-color 2×2 blocks. Therefore $11 \cdot 3 = 33$ is subtracted from 82, and the leaf node count is now 49. The final translation of (1,0) leads to Figure 1.31c, where only one of the array entries corresponds to a single-color 2×2 block in the translated array. Decrementing the leaf node count results in 46 nodes, and the process terminates. Of course, we have failed to describe the remaining $4^n - 3$ translations that were also attempted.

Despite trying all possible translations, the algorithm is quite efficient. The key is that for each translation, only the blocks whose motion can lead to space saving need to be considered. This is a direct consequence of the property that a translation of 2^k does not change the number of blocks of size less than $2^k \times 2^k$. For an image that has been enlarged to fit in a $2^{n+1} \times 2^{n+1}$ array, the algorithm will have a maximum depth of recursion of n. Since at each level of recursion we need an array at half the resolution of the previous level, the total amount of space required is $(4/3) \cdot 2^{2n+2}$.

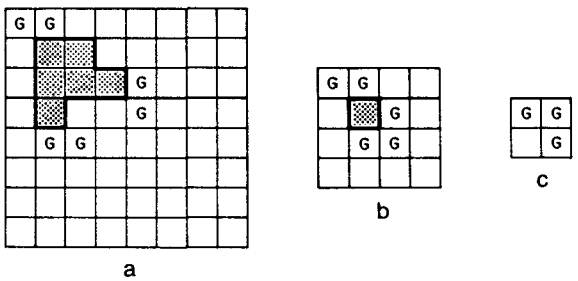


Figure 1.31 The successive translated arrays at half-resolution after application of (a) (1,1) and (b) (1,1), and (c) (1,0) to the original image array of Figure 1.30

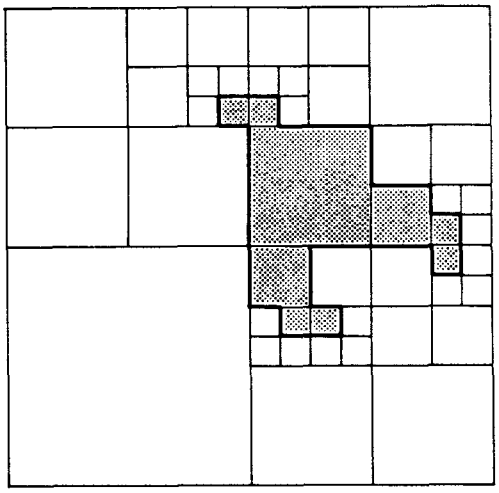


Figure 1.32 Optimal positioning of the quadtree of Figure 1.29

The basic computational task of the algorithm is to count 2×2 blocks of a single color. It can be shown that $4 \cdot n \cdot 2^{2n+2}$ array elements are examined in this process (see Exercise 1.63). Thus the algorithm uses $O(2^{2n})$ space and takes $O(n \cdot 2^{2n})$ time. Nevertheless experiments with typical images show that the algorithm has little effect (e.g., [Same84c]).

Exercises

1.43. Consider the arbitrary placement of a square of size $2^m \times 2^m$ at any position in a $2^n \times 2^n$ image. Prove that in the best case $4 \cdot (n-m) + 1$ nodes are required, while the worst case requires $4 \cdot p + 16 \cdot (n-m) - 27$ nodes. How many of these nodes are black and white, assuming that the square is black? Prove that on the average, the number of nodes that is required is $O(p+n-m)$.

1.44. What are the worst-case storage requirements of storing an arbitrary rectangle in a quadtree corresponding to a $2^n \times 2^n$ image? Give an example of the worst case and the number of nodes it requires.

1.45 Assume that the probability of a particular pixel's being black is one-half and likewise for being white. Given a $2^n \times 2^n$ image represented by a quadtree, what is the expected number of nodes, say $E(n)$, in the quadtree? Also compute the expected number of black, white, and gray nodes.

1.46 Suppose that instead of knowing the probability a particular pixel is black or white, we know the percentage of the total pixels in the image that are black. Given a $2^n \times 2^n$ image represented by a quadtree, what is the expected number of nodes in the quadtree?

1.47. The proof of Theorem 1.1 and the subsequent discussion raise the question of how N squares should be arranged so that each is intersected by a curve of minimum length extending to the outside of the squares on each end. Such a configuration leads to a minimal curve in the sense that it has a maximal ratio of squares to length. For which value of N is this ratio the smallest?

1.48. Try to prove that the upper bound of Theorem 1.1 can be tightened to be $a \cdot n + b + 8 \cdot p$ where a and b are constants.

1.49. Decompose the polygon used in the proof of Theorem 1.1 into a sequence of curves in the following manner. Mark the points where G enters and exits each square of side width d. Choose one of these points, say P, and define the first curve in G as extending from P until four squares have been intersected and a crossing is made into a different fifth square. This is the starting point for another curve in G that intersects four new squares, not counting those intersected by any previous curve. Prove that all of the curves, except for the last one, must be at least of length d. Using this result, prove that the upper bound on the number of nodes in the quadtree is $16 \cdot n - 11 + 16 \cdot p$.

1.50. Prove that the quadtree corresponding to a square of side width 2 consisting of the central four squares in a $2^n \times 2^n$ image has $16 \cdot n - 11$ nodes (see Figure 1.27).

1.51. Take a curve that follows a vertical line through the center of a $2^n \times 2^n$ image and lengthen it slightly by making it intersect all of the pixels on either side of the vertical line (see Figure 1.28). Prove that as n increases, the total number of nodes in the quadtree approaches $8 \cdot p$ where $p = 2^n$.

1.52. Using a technique analogous to that used in Exercise 1.51, construct a polygon of perimeter p by approximating a square in the center of the image whose side is one-fourth the side of the image. Prove that its quadtree has approximately $8 \cdot p$ nodes.

1.53. Prove that $O(p+n)$ is a least upper bound on the number of nodes in a quadtree corresponding to a polygon. Assume that $p \leq 2^{2n}$ (i.e., the number of pixels in the image). Equivalently the polygon boundary can touch all of the pixels in the most trivial way but can be no longer. Decompose your proof into two parts depending on whether p is greater than $4 \cdot n$.

1.54. Can you prove that for an arbitrary quadtree (not necessarily a polygon), the number of nodes doubles as the resolution is doubled?

1.55. Derive a result analogous to Theorem 1.1 for a three-dimensional polyhedron represented as an octree. In this case the perimeter corresponds to the surface area.

1.56. Prove Theorem 1.2.

1.57. Assuming an image of resolution n and measuring the perimeter, say p, in terms of the number of border pixels, prove that the total number of nodes in a d-dimensional quadtree is less than or equal to $4 \cdot n \cdot p$.

1.58. Assuming an image of resolution n and measuring the perimeter, say p, in terms of the number of border pixels, prove that the total number of black nodes in a d-dimensional quadtree is less than or equal to $(2^d - 1) \cdot n \cdot p / d$.

1.59. How tight are the bounds obtained in Exercises 1.57 and 1.58 for the number of nodes in a d-dimensional quadtree for an arbitrary region? Are they realizable?

1.60. Prove that for a region such that w is the maximum of its horizontal and vertical extent (measured in pixel widths) and $2^{n-1} < w \leq 2^n$, the optimal grid resolution is either n or $n+1$.

1.61. Prove that translating a region by 2^k pixels in any direction does not change the number of black or white blocks of size less than $2^k \times 2^k$.

1.62. Can you formally prove that the method described in the text does indeed yield the optimal quadtree?

1.63. Prove that $4 \cdot n \cdot 2^{2n+2}$ array elements are examined in the process of constructing the optimal quadtree.

1.64. How would you find the optimal bintree?

US005263136A

United States Patent [19]

DeAguiar et al.

[11] Patent Number: 5,263,136

[45] Date of Patent: Nov. 16, 1993

[54] **SYSTEM FOR MANAGING TILED IMAGES USING MULTIPLE RESOLUTIONS**

[75] Inventors: **John R. DeAguiar**, Sebastopol; **Ross M. Larkin**, Rollings Hills, both of Calif.

[73] Assignee: **Optigraphics Corporation**, San Diego, Calif.

[21] Appl. No.: **694,416**

[22] Filed: **Apr. 30, 1991**

[51] **Int. Cl.**[5] **G06F 15/20**

[52] **U.S. Cl.** **395/164**; 345/201

[58] **Field of Search** 395/162, 164, 166, 128-130; 340/798, 799; 358/452, 455

[56] **References Cited**

U.S. PATENT DOCUMENTS

Re. 31,200	4/1983	Sukonick et al.	395/162
4,878,183	10/1989	Ewart	395/128 X
4,920,504	4/1990	Sawada et al.	395/166
4,951,230	8/1990	Dalrymple et al.	395/166
5,020,003	5/1991	Moshenberg	395/164
5,150,462	9/1992	Takeda et al.	395/166

Primary Examiner—Dale M. Shaw
Assistant Examiner—Kee M. Tung
Attorney, Agent, or Firm—Knobbe, Martens, Olson & Bear

[57] **ABSTRACT**

An image memory management system for tiled images. The system defines an address space for a virtual memory that includes an image data cache and a disk. An image stack for each source image is stored as a full resolution image and a set of lower-resolution subimages. Each tile of an image may exist in one or more of five different states as follows: uncompressed and resident in the image data cache, compressed and resident in the image data cache, uncompressed and resident on disk, compressed and resident on disk and not loaded but re-creatable using data from higher-resolution image tiles.

17 Claims, 39 Drawing Sheets

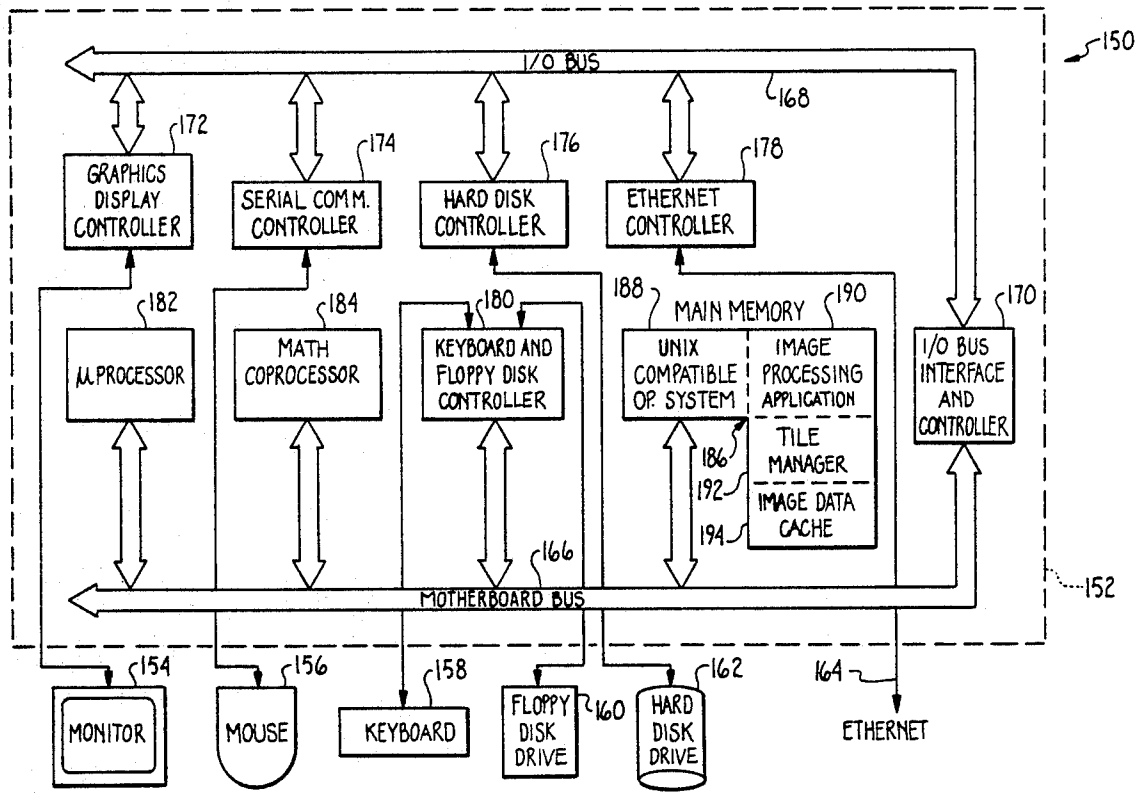
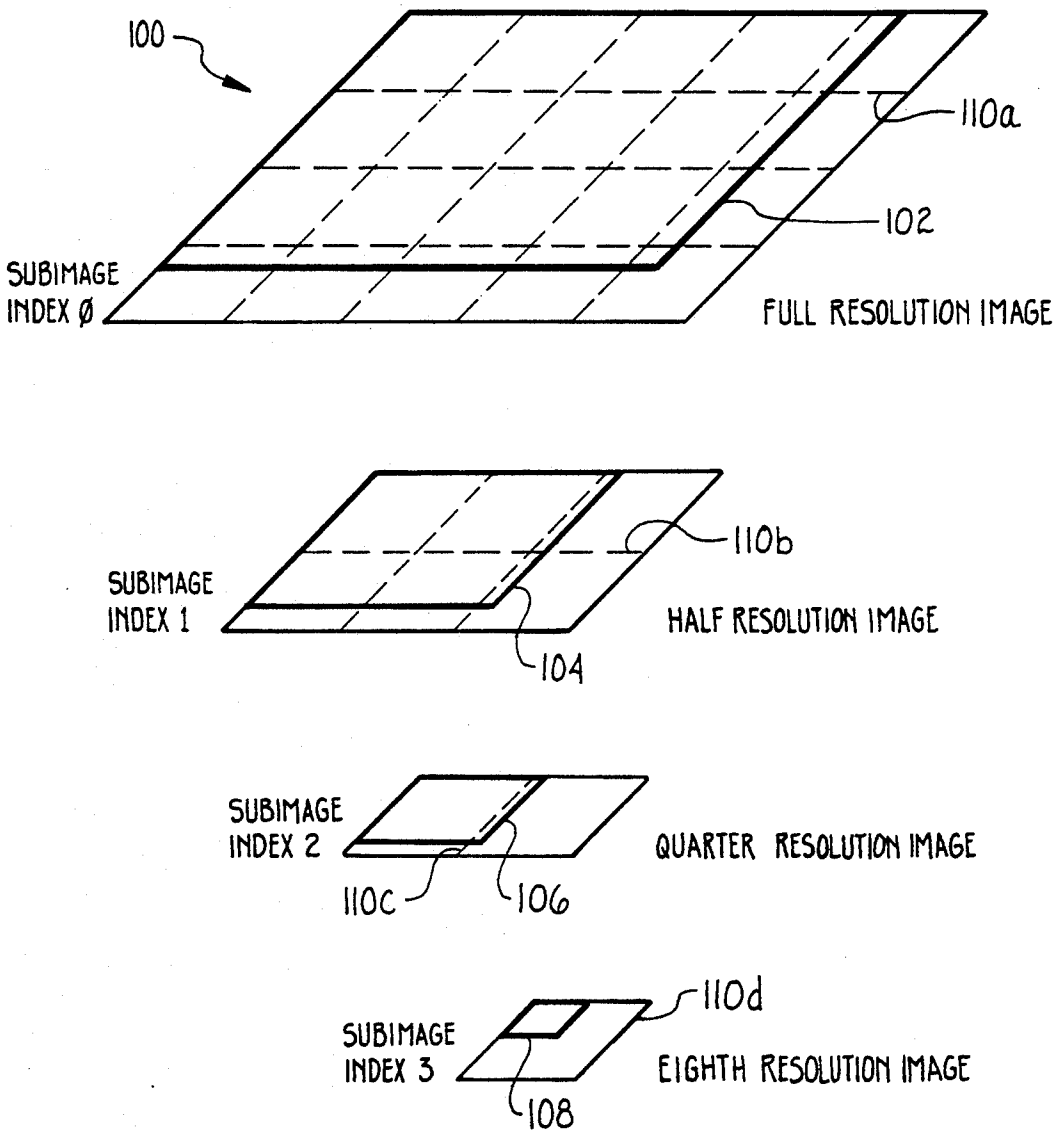


Image Stack

Fig. 1



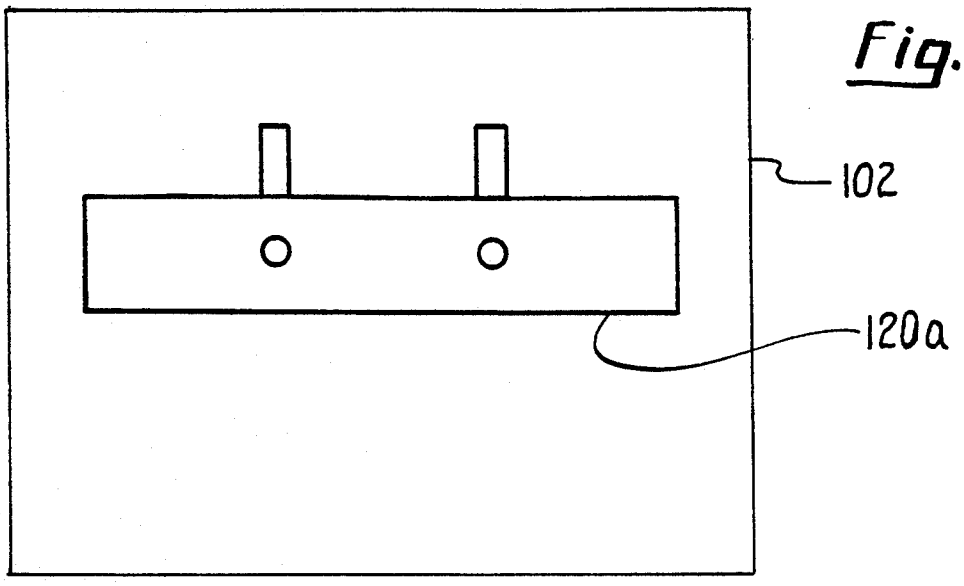


Fig. 2

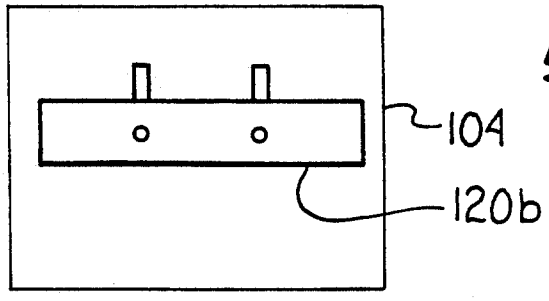


Fig. 3

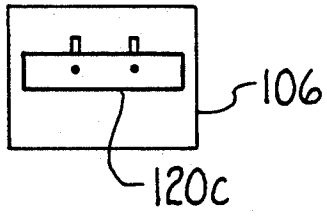


Fig. 4

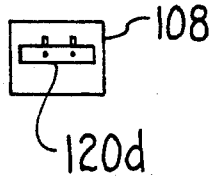


Fig. 5

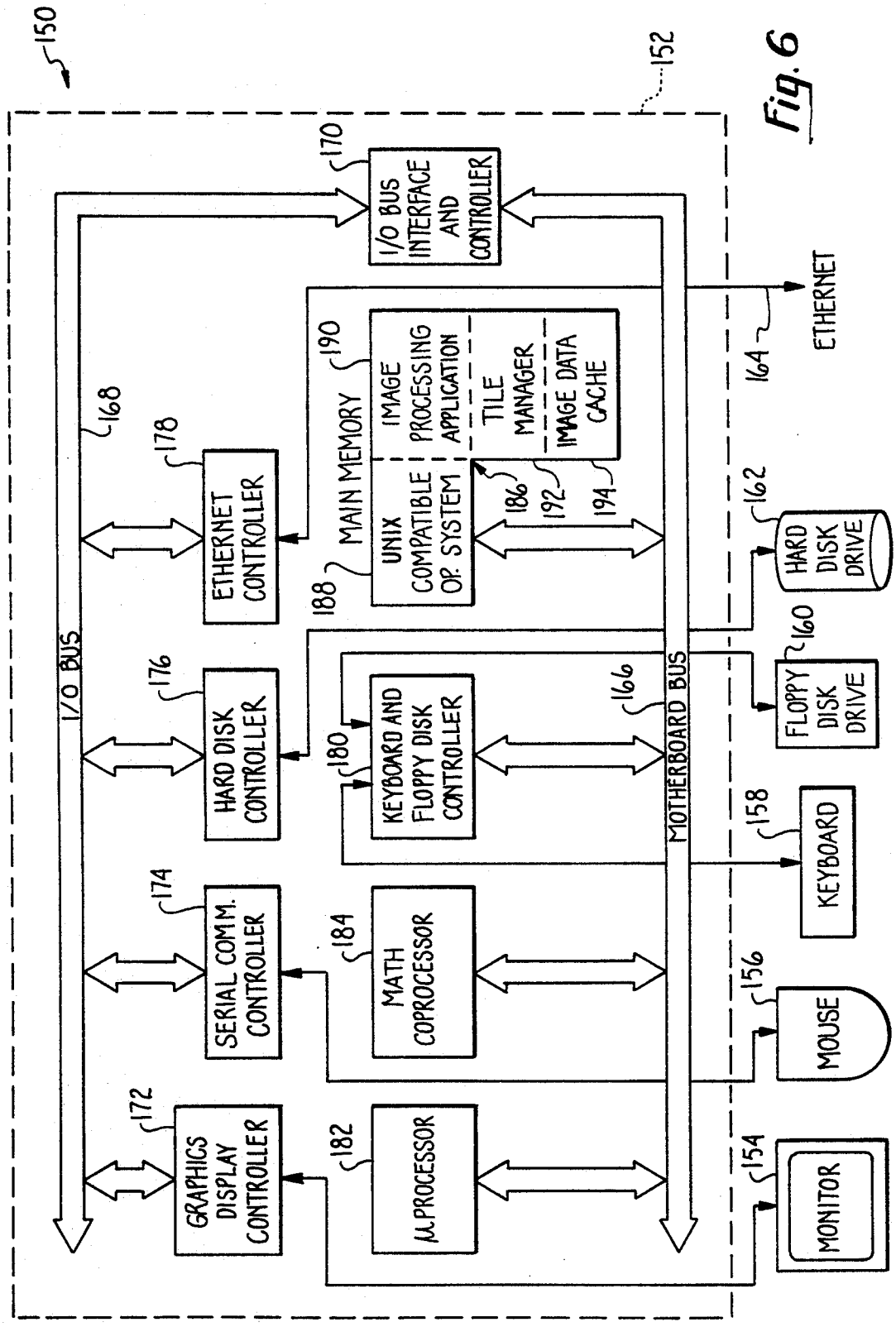

150
152
170
I/O BUS INTERFACE AND CONTROLLER
ETHERNET
164
168
178
ETHERNET CONTROLLER
190
MAIN MEMORY
IMAGE PROCESSING APPLICATION
TILE MANAGER
IMAGE DATA CACHE
UNIX COMPATIBLE OP. SYSTEM
186
192
194
188
162
HARD DISK DRIVE
160
FLOPPY DISK DRIVE
I/O BUS
176
HARD DISK CONTROLLER
180
KEYBOARD AND FLOPPY DISK CONTROLLER
166
MOTHERBOARD BUS
158
KEYBOARD
174
SERIAL COMM. CONTROLLER
184
MATH COPROCESSOR
156
MOUSE
172
GRAPHICS DISPLAY CONTROLLER
182
μ PROCESSOR
154
MONITOR

Fig. 6

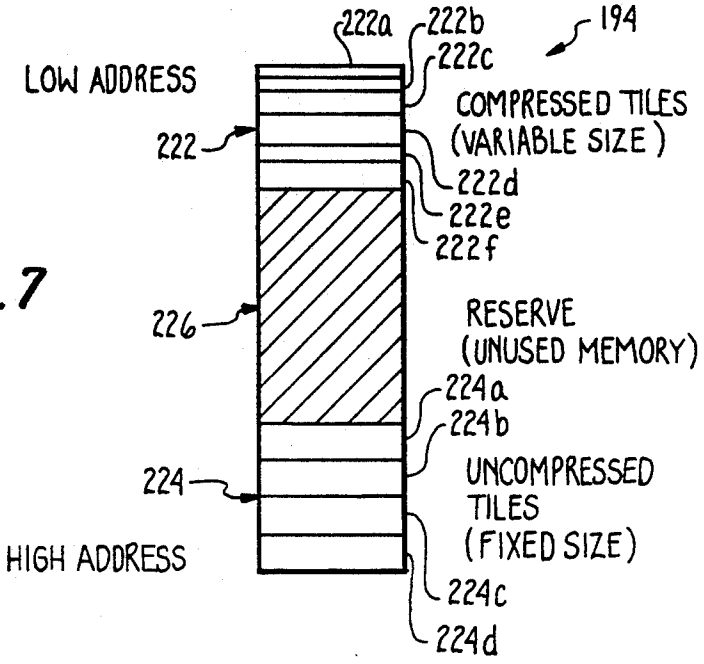

222a
222b
222c
194
LOW ADDRESS
COMPRESSED TILES
(VARIABLE SIZE)
222
222d
222e
222f
226
RESERVE
(UNUSED MEMORY)
224a
224b
224
UNCOMPRESSED
TILES
(FIXED SIZE)
HIGH ADDRESS
224c
224d

Fig. 7

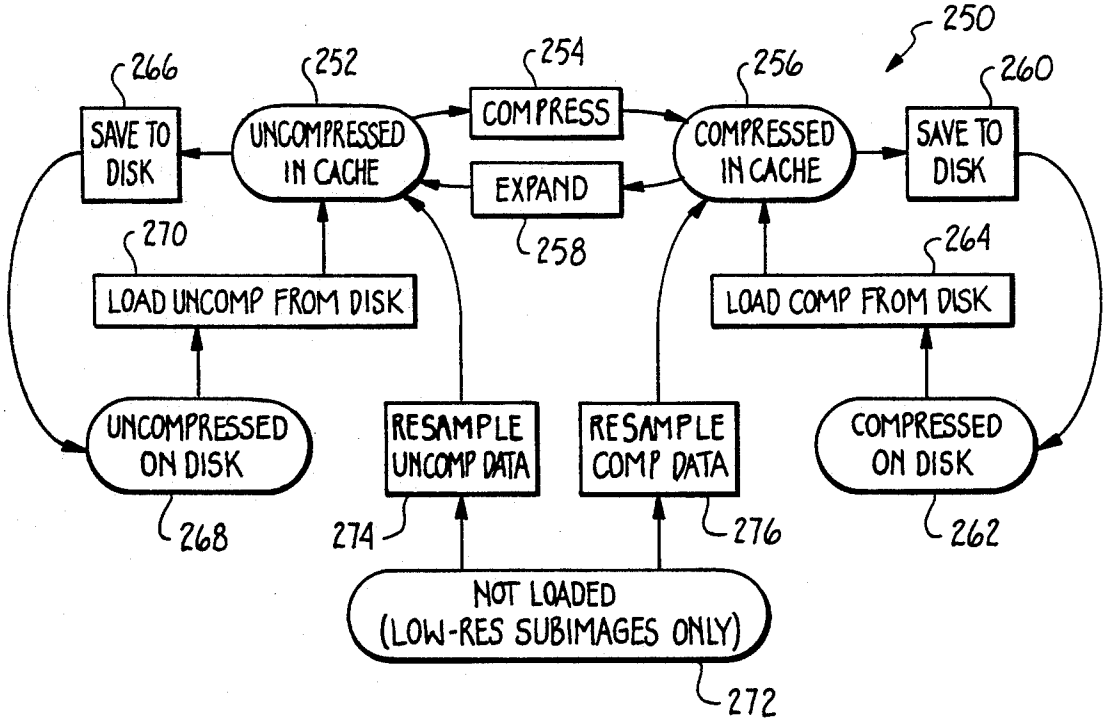

250
266
252
254
256
260
SAVE TO DISK
UNCOMPRESSED IN CACHE
COMPRESS
EXPAND
COMPRESSED IN CACHE
SAVE TO DISK
258
270
264
LOAD UNCOMP FROM DISK
LOAD COMP FROM DISK
UNCOMPRESSED ON DISK
RESAMPLE UNCOMP DATA
RESAMPLE COMP DATA
COMPRESSED ON DISK
268
274
276
262
NOT LOADED
(LOW-RES SUBIMAGES ONLY)
272

Fig. 8

Fig. 9A Document Information Structure

300

SELF-REFERENCE TO DOCUMENT HANDLE 302	"OVERVIEWS INVALID" FLAG 304	CACHE IMAGE COMPRESSION ALGORITHM 306
IMAGE COLOR TYPE 308	BITS PER IMAGE PIXEL 310	TILE SIZE INFORMATION 312
NUMBER OF SUBIMAGES IN DOC 314	INPUT FILE INFO 316	OUTPUT FILE INFO 318
LIST OF SUBIMAGE HEADERS 320		

Fig. 9B

321

POINTER TO TILE HEADERS 322	POINTER TO TILE DIRECTORY 324	SUBIMAGE WIDTH AND HEIGHT 326
NUMBER OF TILE ROWS & COLS IN SUBIMAGE 328	IMAGE STACK INDEX OF THIS SUBIMAGE 330	PIXEL RESOLUTION OF THIS SUBIMAGE 332
⋮		

Fig. 10 Tile Header

350

POINTER TO DOCUMENT CONTAINING THIS TILE 352	INDEX OF SUBIMAGE CONTAINING THIS TILE 354	ROW AND COLUMN INDICES OF TILE 356
STATUS INFORMATION 358	PRESERVE COUNT 360	
LOCATION OF UNCOMPRESSED IMAGE DATA IN CACHE MEMORY 362	LOCATION OF COMPRESSED IMAGE DATA IN CACHE MEMORY 364	
LOCATION OF UNCOMPRESSED IMAGE DATA ON DISK 366	LOCATION OF COMPRESSED IMAGE DATA ON DISK 368	
LINK TO NEXT LESS RECENTLY USED TILE 370	LINK TO NEXT MORE RECENTLY USED TILE 372	
NUMBER OF BYTES OF EXPANDED DATA IN TILE 374	NUMBER OF BYTES OF COMPRESSED DATA IN TILE 376	

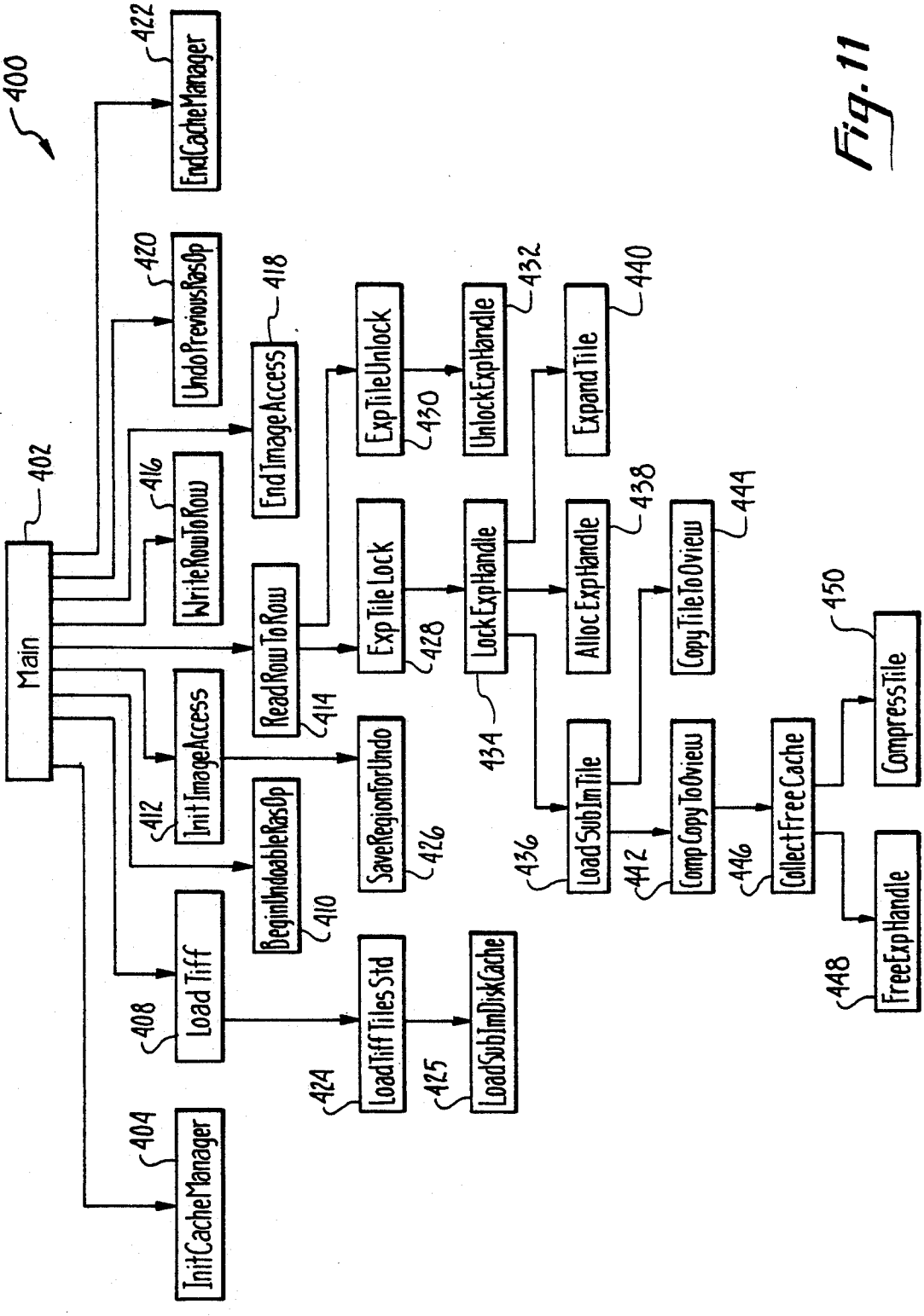

400
Main
402
InitCacheManager
404
LoadTiff
408
InitImageAccess
412
BeginUndoableRasOp
410
LoadTiffTilesStd
424
LoadSubImDiskCache
425
ReadRowToRow
414
SaveRegionForUndo
426
WriteRowToRow
416
UndoPreviousRasOp
420
EndImageAccess
418
EndCacheManager
422
ExpTileLock
428
ExpTileUnlock
430
LockExpHandle
434
UnlockExpHandle
432
AllocExpHandle
438
ExpandTile
440
LoadSubImTile
436
CompCopyToOview
442
CopyTileToOview
444
CollectFreeCache
446
CompressTile
450
FreeExpHandle
448

Fig. 11

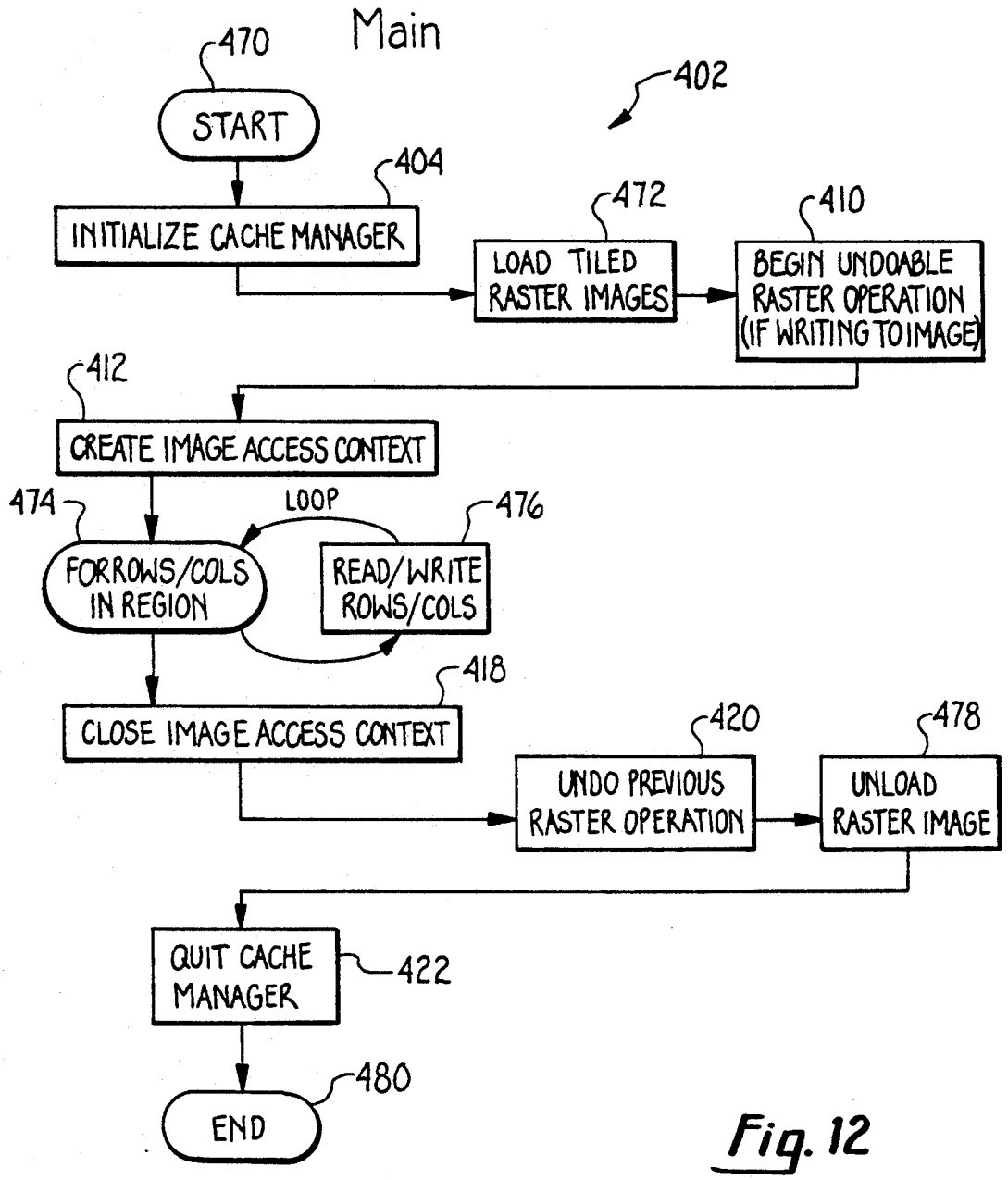
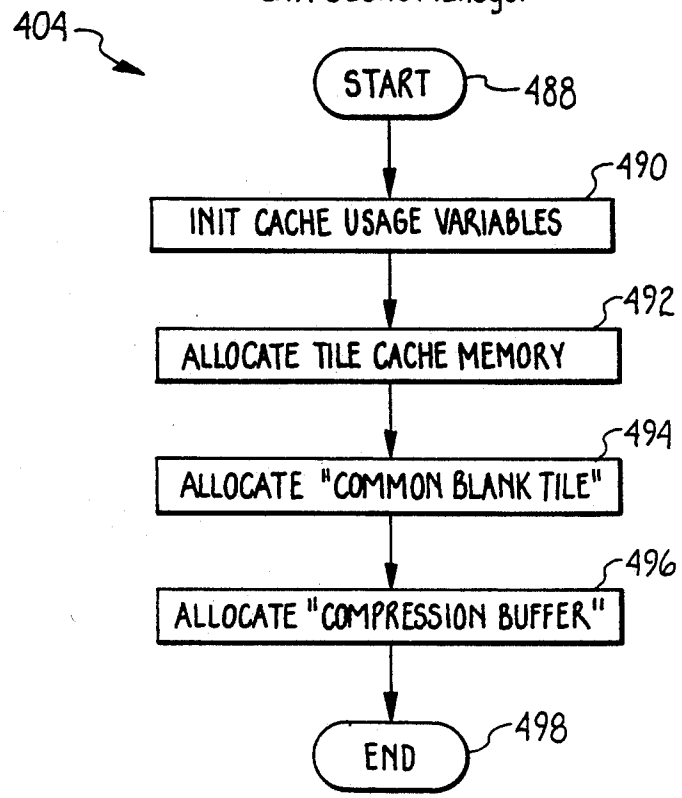

Main
470
START
402
404
INITIALIZE CACHE MANAGER
472
LOAD TILED RASTER IMAGES
410
BEGIN UNDOABLE RASTER OPERATION (IF WRITING TO IMAGE)
412
CREATE IMAGE ACCESS CONTEXT
474
FOR ROWS/COLS IN REGION
LOOP
476
READ/WRITE ROWS/COLS
418
CLOSE IMAGE ACCESS CONTEXT
420
UNDO PREVIOUS RASTER OPERATION
478
UNLOAD RASTER IMAGE
QUIT CACHE MANAGER
422
480
END

Fig. 12

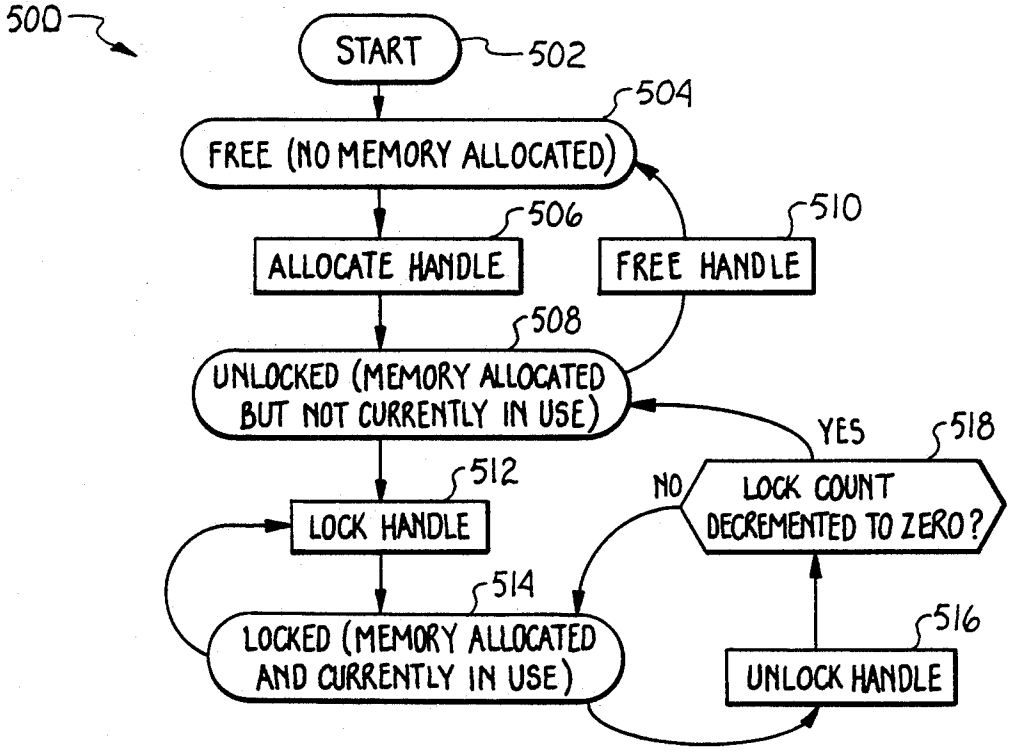
Init Cache Manager

Fig. 13



Memory Block State Diagram

Fig. 14



Init Image Access

Fig. 15A

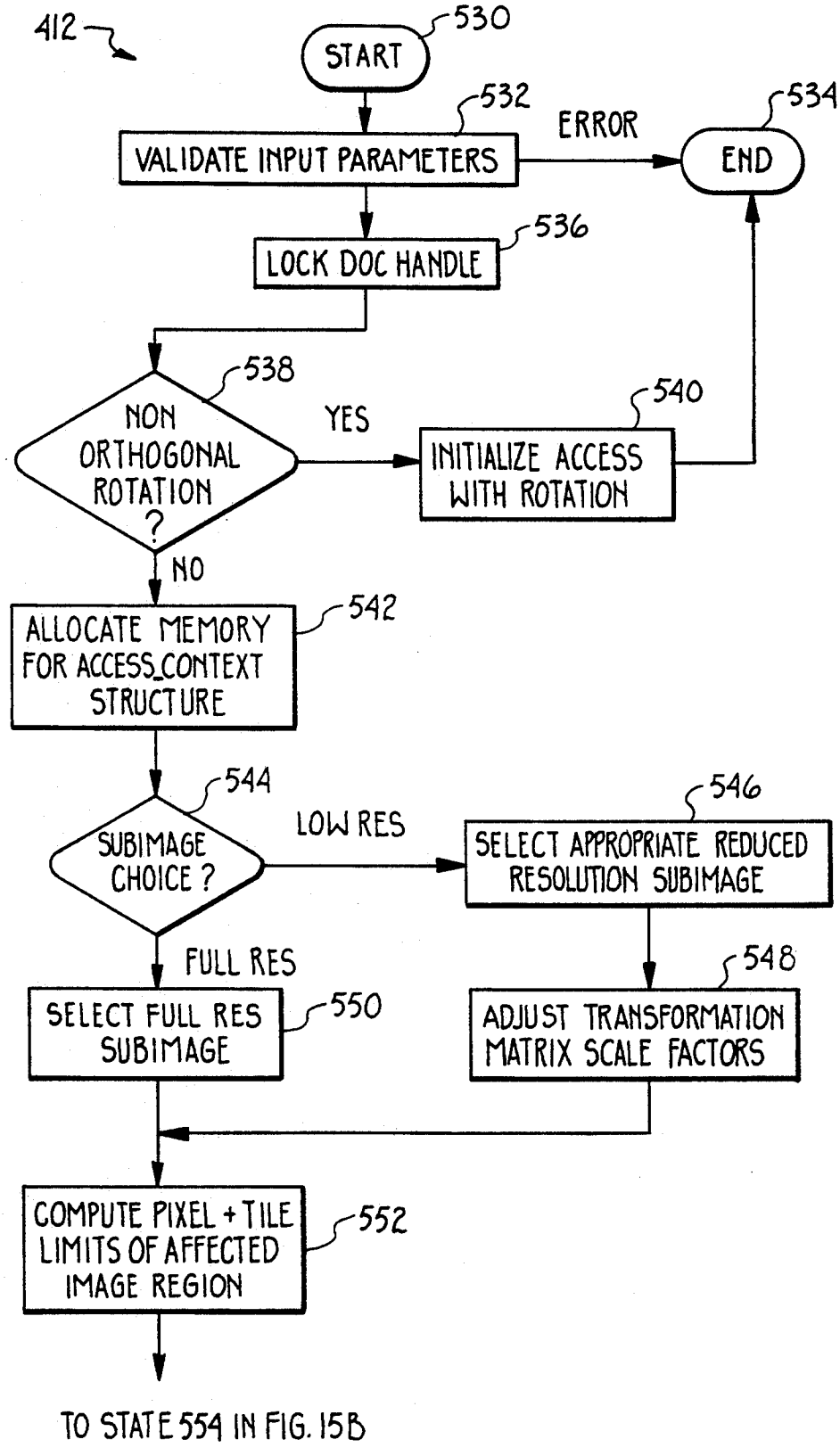
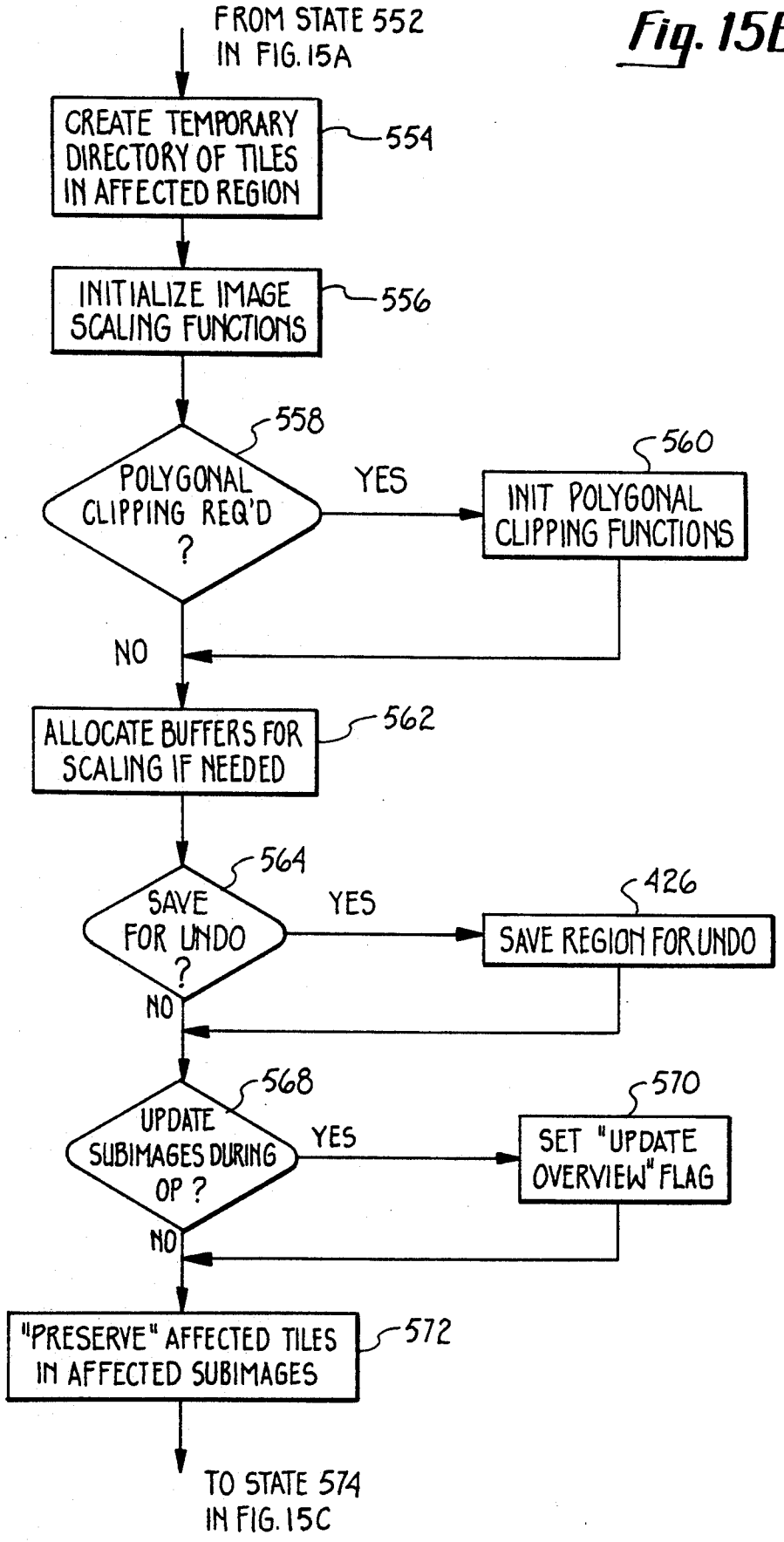


Fig. 15B


FROM STATE 552 IN FIG. 15A
CREATE TEMPORARY DIRECTORY OF TILES IN AFFECTED REGION
554
INITIALIZE IMAGE SCALING FUNCTIONS
556
558
POLYGONAL CLIPPING REQ'D ?
YES
560
INIT POLYGONAL CLIPPING FUNCTIONS
NO
ALLOCATE BUFFERS FOR SCALING IF NEEDED
562
564
SAVE FOR UNDO ?
YES
426
SAVE REGION FOR UNDO
NO
568
UPDATE SUBIMAGES DURING OP ?
YES
570
SET "UPDATE OVERVIEW" FLAG
NO
"PRESERVE" AFFECTED TILES IN AFFECTED SUBIMAGES
572
TO STATE 574 IN FIG. 15C

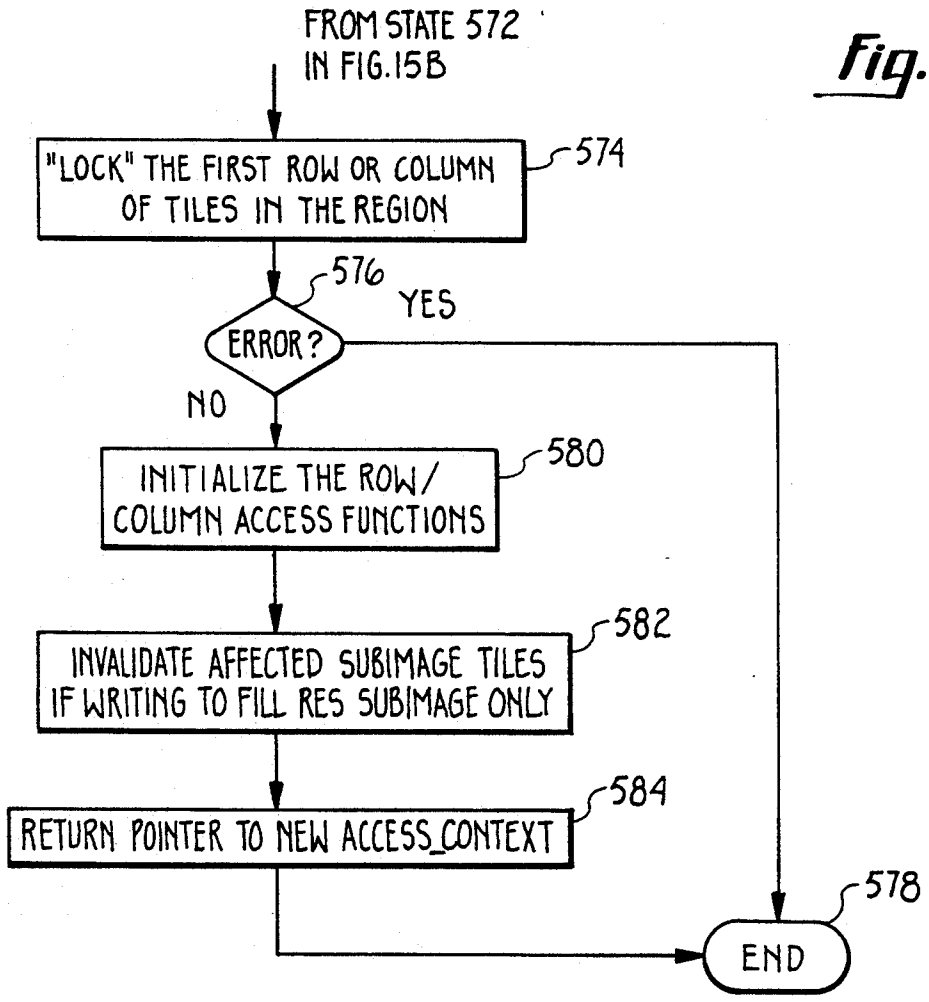

FROM STATE 572
IN FIG.15B
"LOCK" THE FIRST ROW OR COLUMN OF TILES IN THE REGION
574
576
YES
ERROR?
NO
INITIALIZE THE ROW/ COLUMN ACCESS FUNCTIONS
580
INVALIDATE AFFECTED SUBIMAGE TILES IF WRITING TO FILL RES SUBIMAGE ONLY
582
RETURN POINTER TO NEW ACCESS_CONTEXT
584
578
END

Fig. 15C

Fig. 16

Access Context 600

POINTER TO AFFECTED DOC 602	"SUBIMAGE CHOICE" OPTION VALUE 604	INDEX OF AFFECTED SUBIMAGE 606	ACCESS QUANTUM 608	READ/WRITE OPTION 610
BASIC ORTHOGONAL ROTATION VALUE 612	PIXEL COMBINATION OPERATION 614	SCALER TYPE OPERATION 616	"UPDATE OVERVIEWS" FLAG 618	
I/O BUFFER WIDTH & HEIGHT 620	I/O BUFFER PITCH (BYTES/ROW) 622	I/O BUFFER BIT OFFSET TO START OF RUN 624		
ROWS PER STRIP ("AQ_STRIP" ACCESS QUANTUM) 626	NUMBER OF I/O BUFFER ROWS YET TO BE PROCESSED 628			
POINTER TO ACCESS FUNCTION USED IN "SeqBufImageAccess" 630	STEPPING DIRECTIONS FOR IMAGE ROW AND COLUMN INDICIES 632			
POINTER TO POLYGON CLIPPING INFORMATION 634	POINTER TO RASTER SCALING INFORMATION 636			
POINTER TO UNCOMPRESSED DATA IN CURRENTLY LOCKED TILES 638	POINTER TO REGION TILE DIRECTORY 640	NEXT IMAGE ROW & COLUMN TO BE ACCESSED 642		
TERMINAL ROW & COLUMN OF ACCESS REGION 644	UNCLIPPED EXTENT OF ACCESS REGION 646	CLIPPED EXTENT OF ACCESS REGION 648		
CLIPPED IMAGE BUFFER BIT OFFSET AND LENGTH 650	NUMBER OF TILE ROWS & COLS IN ACCESS REGION 652	ROW & COLUMN OF CURRENTLY LOCKED TILES 654		
IMAGE ROW & COL AT ORIGIN OF FIRST TILE IN ACCESS REGION 656	NUMBER OF I/O BUFFER ROWS HELD OVER FOR NEXT STRIP 658			
POINTER TO IMAGE TILING/UNTILING BUFFER 660	NUMBER OF BYTES IN TILING/UNTILING BUFFER 662	BIT OFFSET FOR TILING/UNTILING BUFFER 664		
ACCESS TRANSFORMATION MATRIX 666				

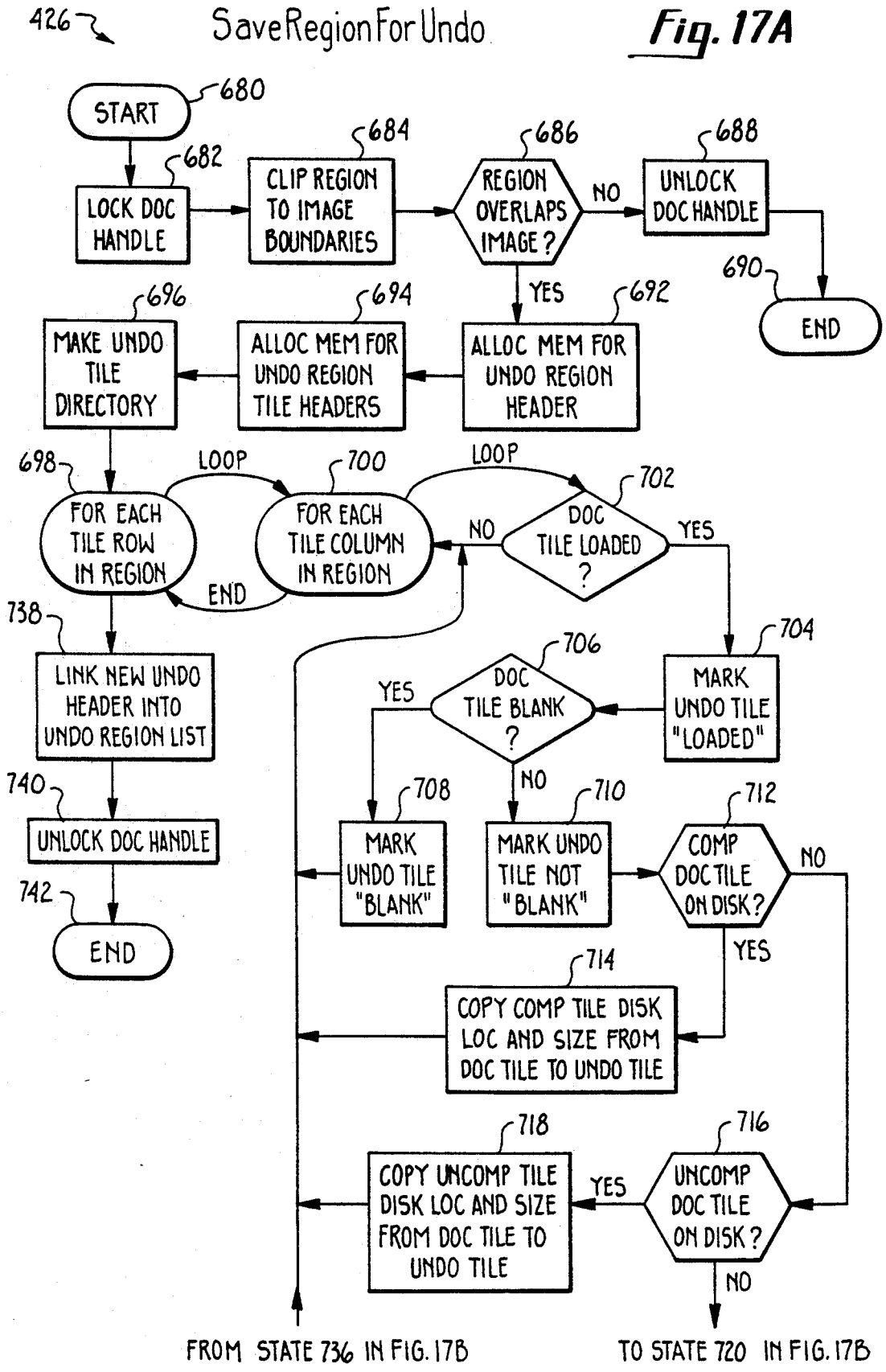

426
SaveRegionForUndo
START
680
LOCK DOC HANDLE
682
CLIP REGION TO IMAGE BOUNDARIES
684
REGION OVERLAPS IMAGE?
686
NO
UNLOCK DOC HANDLE
688
690
END
YES
ALLOC MEM FOR UNDO REGION HEADER
692
ALLOC MEM FOR UNDO REGION TILE HEADERS
694
MAKE UNDO TILE DIRECTORY
696
698
FOR EACH TILE ROW IN REGION
LOOP
700
FOR EACH TILE COLUMN IN REGION
LOOP
END
NO
DOC TILE LOADED ?
702
YES
738
LINK NEW UNDO HEADER INTO UNDO REGION LIST
740
UNLOCK DOC HANDLE
742
END
706
DOC TILE BLANK ?
YES
NO
704
MARK UNDO TILE "LOADED"
708
MARK UNDO TILE "BLANK"
710
MARK UNDO TILE NOT "BLANK"
712
COMP DOC TILE ON DISK?
NO
YES
714
COPY COMP TILE DISK LOC AND SIZE FROM DOC TILE TO UNDO TILE
718
COPY UNCOMP TILE DISK LOC AND SIZE FROM DOC TILE TO UNDO TILE
YES
716
UNCOMP DOC TILE ON DISK?
NO
FROM STATE 736 IN FIG. 17B
TO STATE 720 IN FIG. 17B

Fig. 17A

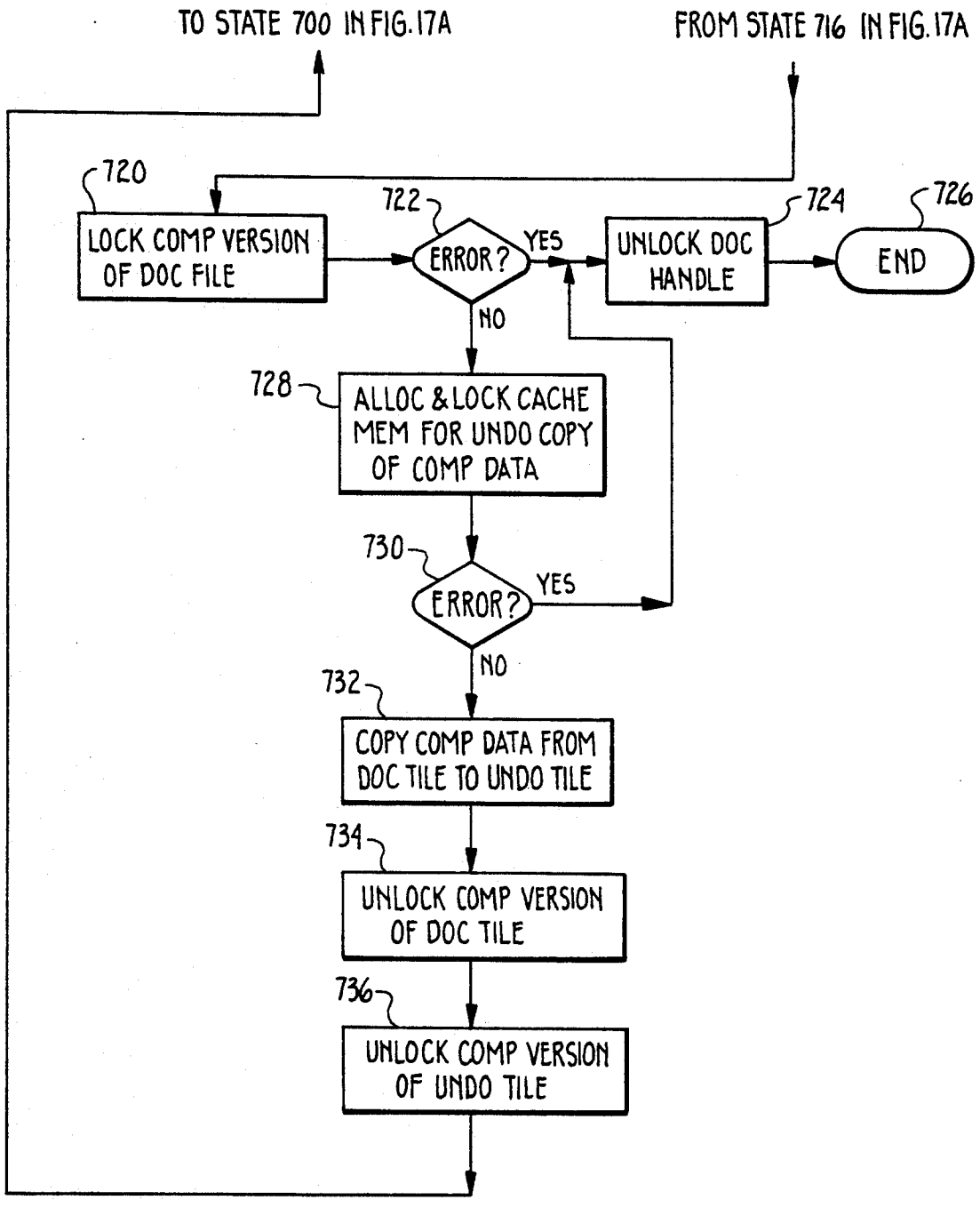
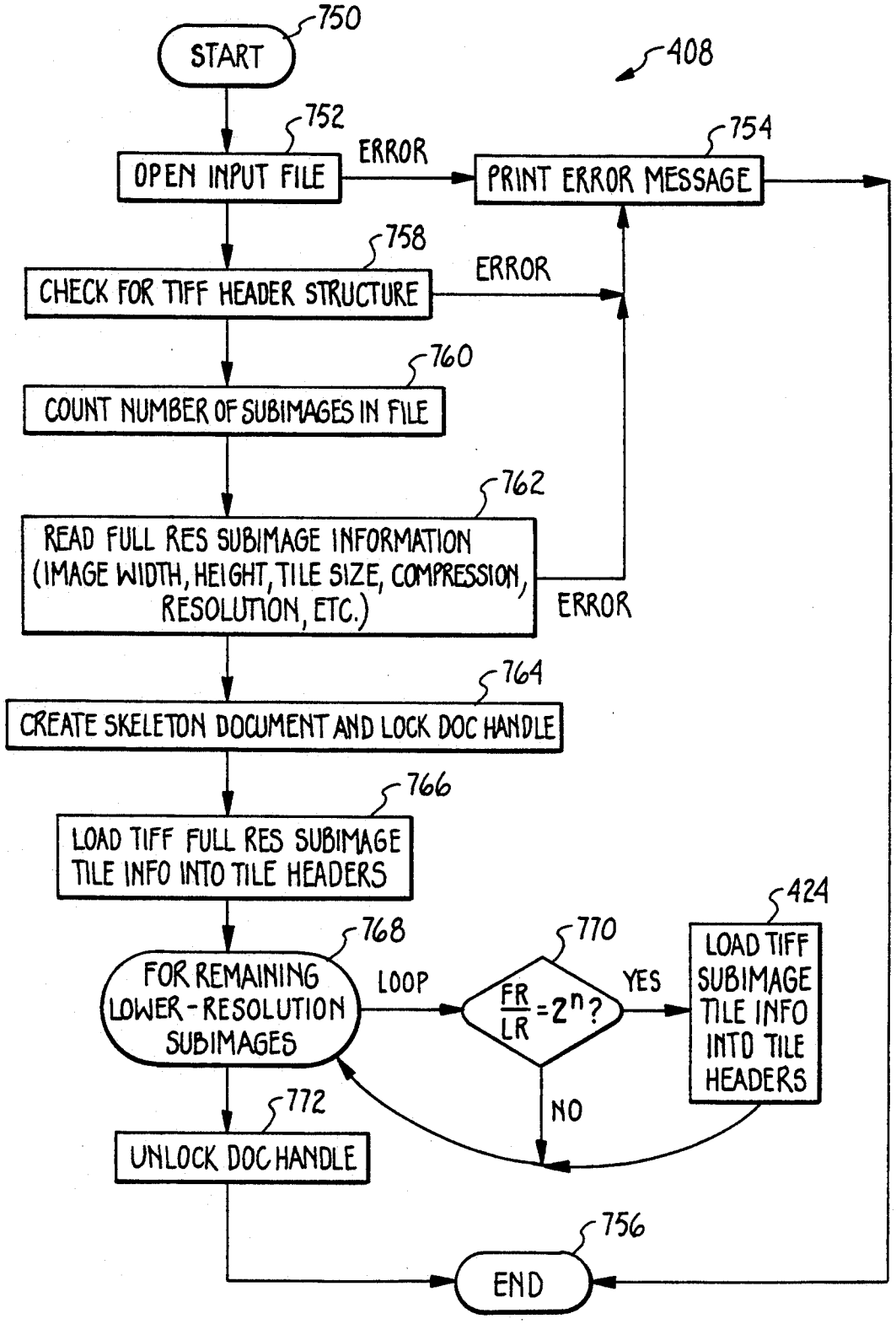

TO STATE 700 IN FIG. 17A
FROM STATE 716 IN FIG. 17A
720
LOCK COMP VERSION OF DOC FILE
722
ERROR?
YES
NO
724
UNLOCK DOC HANDLE
726
END
728
ALLOC & LOCK CACHE MEM FOR UNDO COPY OF COMP DATA
730
ERROR?
YES
NO
732
COPY COMP DATA FROM DOC TILE TO UNDO TILE
734
UNLOCK COMP VERSION OF DOC TILE
736
UNLOCK COMP VERSION OF UNDO TILE

Fig. 17B

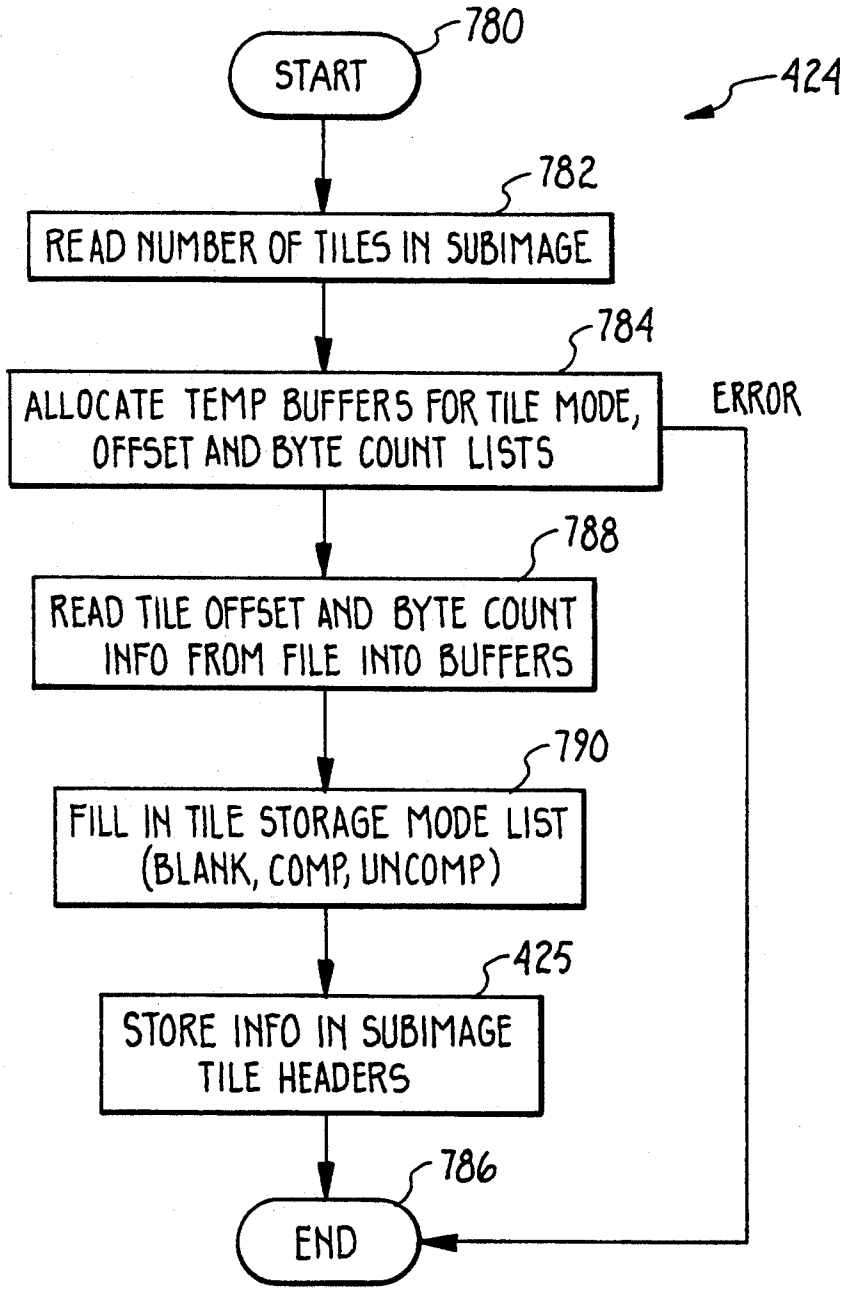
LoadTiff

Fig. 18



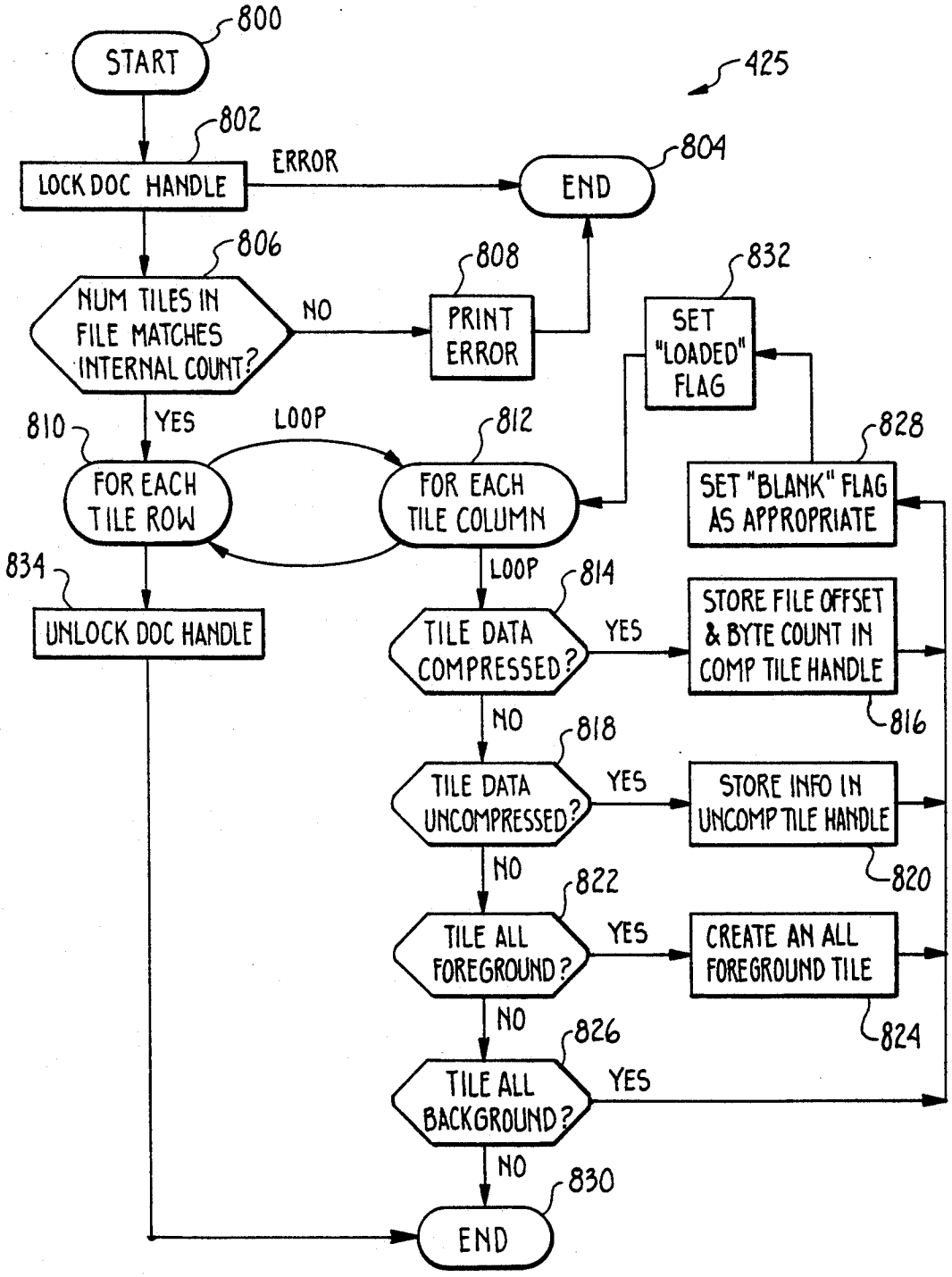
Load Tiff Tiles Std

Fig. 19



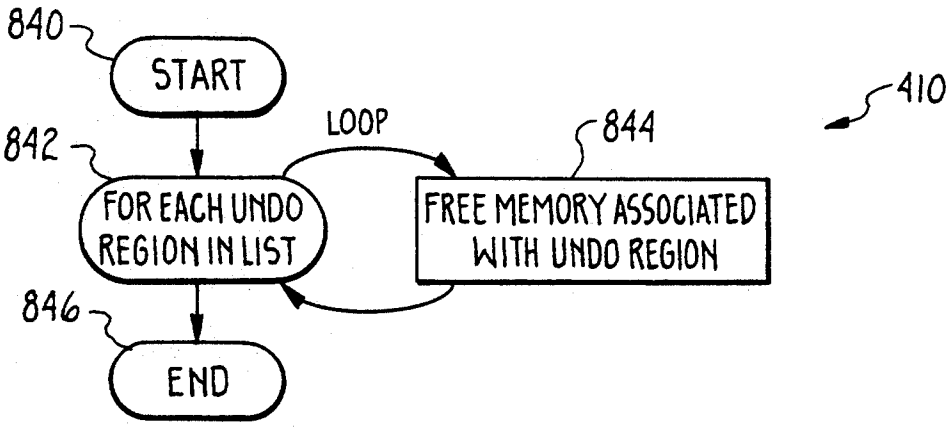
LoadSubImDiskCache

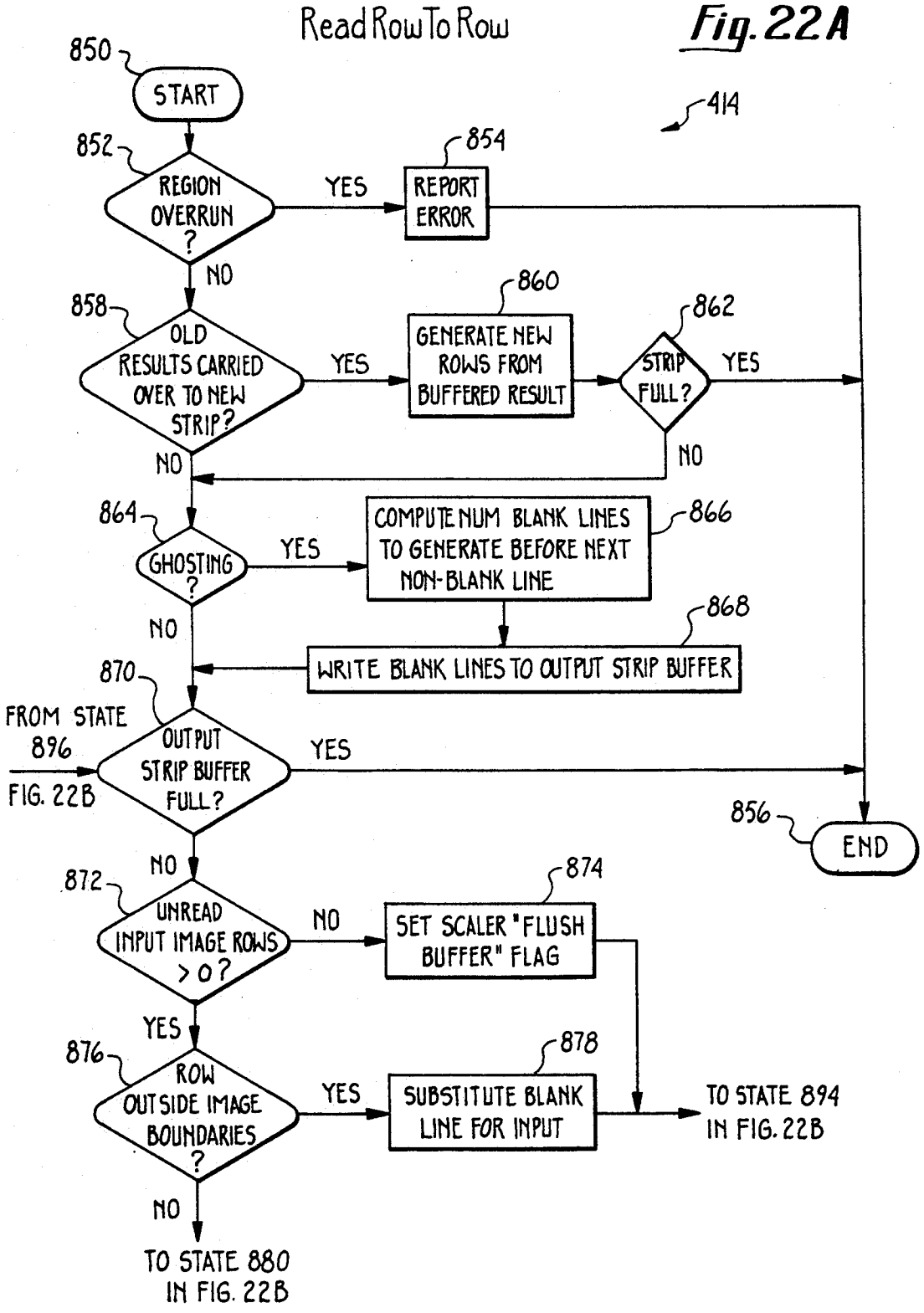
Fig. 20



BeginUndoableRasOp

Fig. 21




Read Row To Row
Fig. 22A
414
850
START
852
REGION OVERRUN ?
YES
854
REPORT ERROR
NO
858
OLD RESULTS CARRIED OVER TO NEW STRIP?
YES
860
GENERATE NEW ROWS FROM BUFFERED RESULT
862
STRIP FULL?
YES
NO
NO
864
GHOSTING ?
YES
COMPUTE NUM BLANK LINES TO GENERATE BEFORE NEXT NON-BLANK LINE
866
868
WRITE BLANK LINES TO OUTPUT STRIP BUFFER
NO
870
FROM STATE 896
FIG. 22B
OUTPUT STRIP BUFFER FULL?
YES
856
END
NO
872
UNREAD INPUT IMAGE ROWS > 0?
NO
874
SET SCALER "FLUSH BUFFER" FLAG
YES
876
ROW OUTSIDE IMAGE BOUNDARIES ?
YES
878
SUBSTITUTE BLANK LINE FOR INPUT
TO STATE 894 IN FIG. 22B
NO
TO STATE 880 IN FIG. 22B

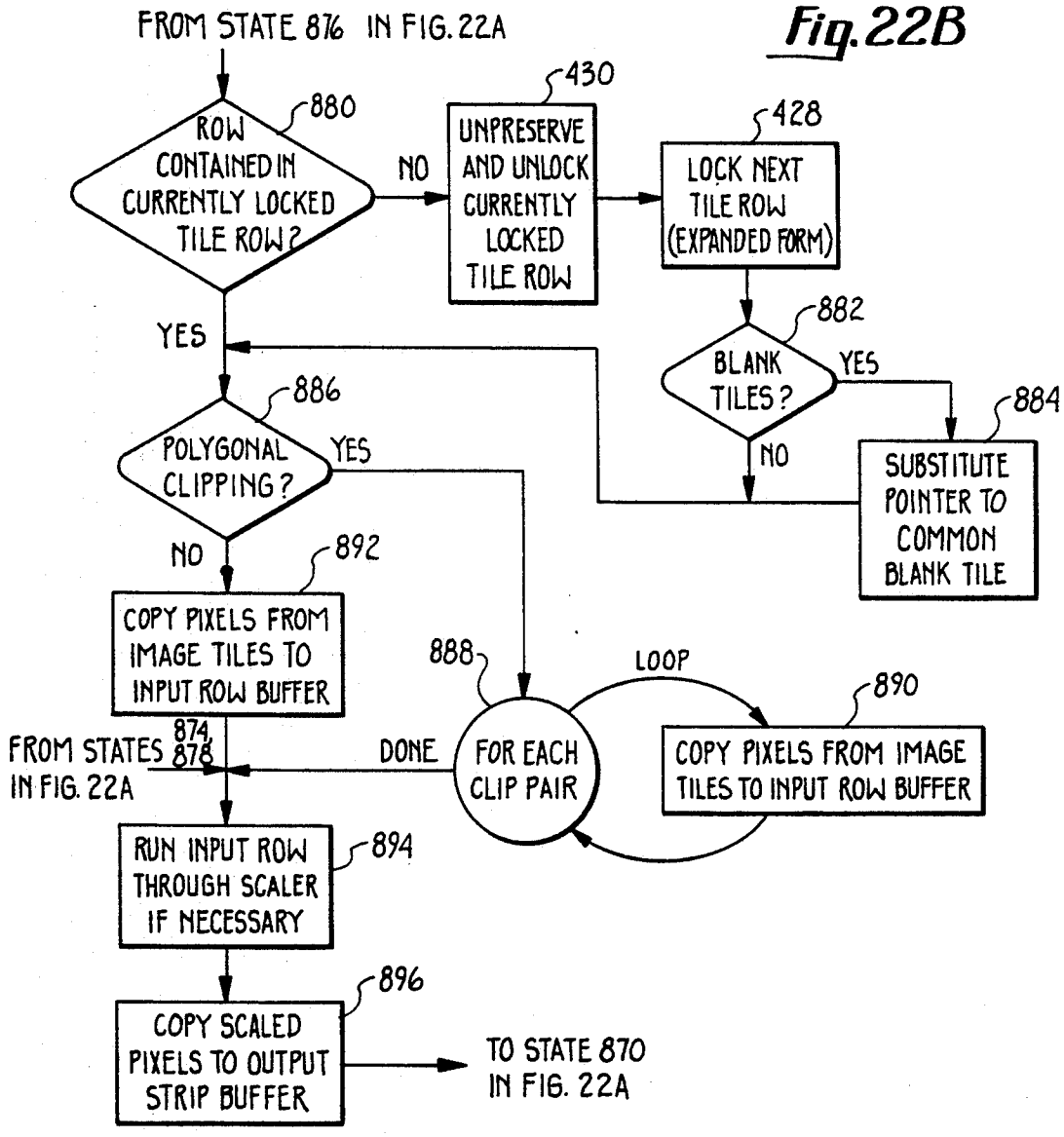
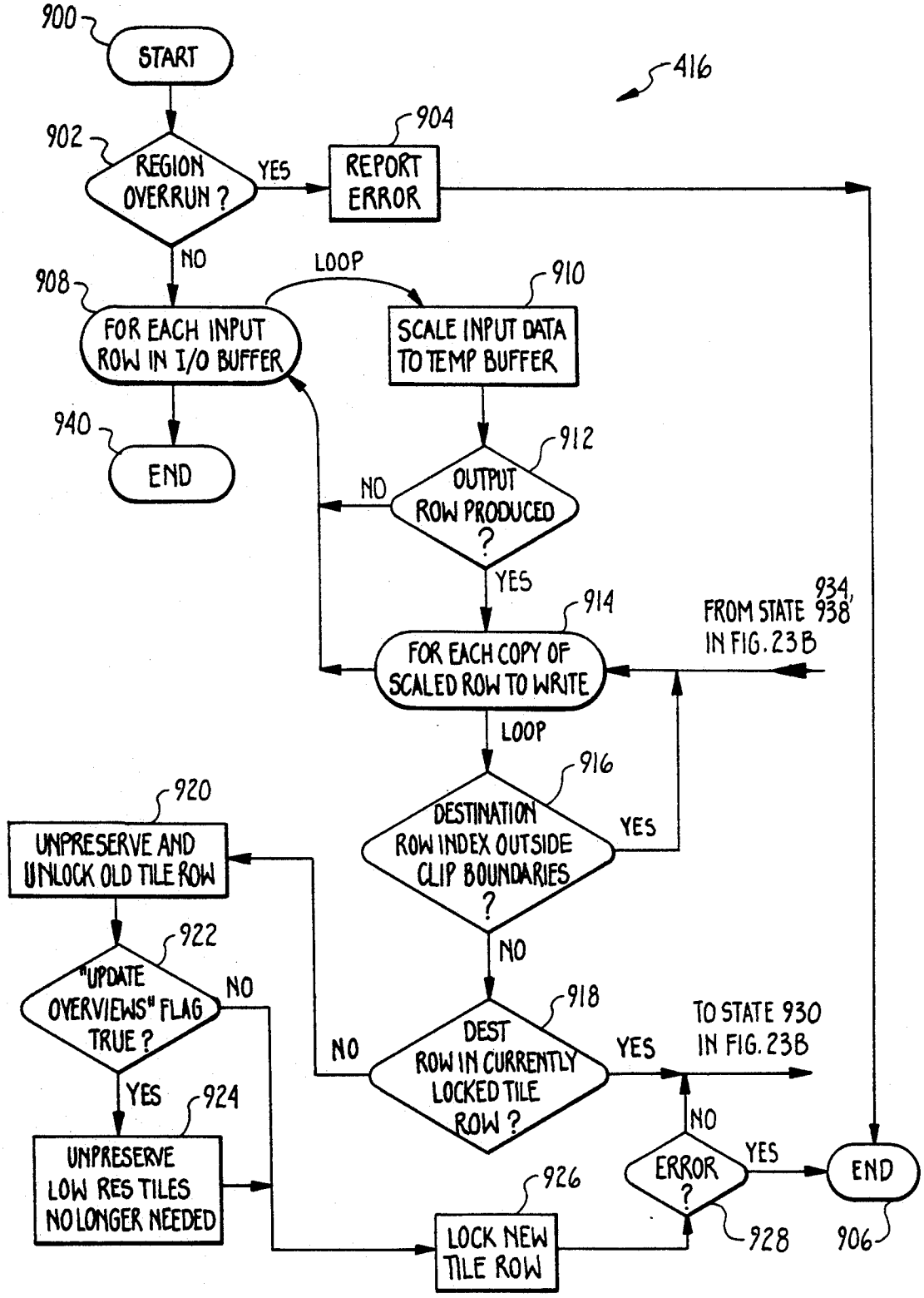

FROM STATE 876 IN FIG. 22A
880
ROW CONTAINED IN CURRENTLY LOCKED TILE ROW?
NO
430
UNPRESERVE AND UNLOCK CURRENTLY LOCKED TILE ROW
428
LOCK NEXT TILE ROW (EXPANDED FORM)
882
BLANK TILES?
YES
884
SUBSTITUTE POINTER TO COMMON BLANK TILE
NO
YES
886
POLYGONAL CLIPPING?
YES
NO
892
COPY PIXELS FROM IMAGE TILES TO INPUT ROW BUFFER
888
LOOP
890
FOR EACH CLIP PAIR
COPY PIXELS FROM IMAGE TILES TO INPUT ROW BUFFER
DONE
874
878
FROM STATES IN FIG. 22A
RUN INPUT ROW THROUGH SCALER IF NECESSARY
894
COPY SCALED PIXELS TO OUTPUT STRIP BUFFER
896
TO STATE 870 IN FIG. 22A

Fig. 22B

WriteRowToRow

Fig. 23A



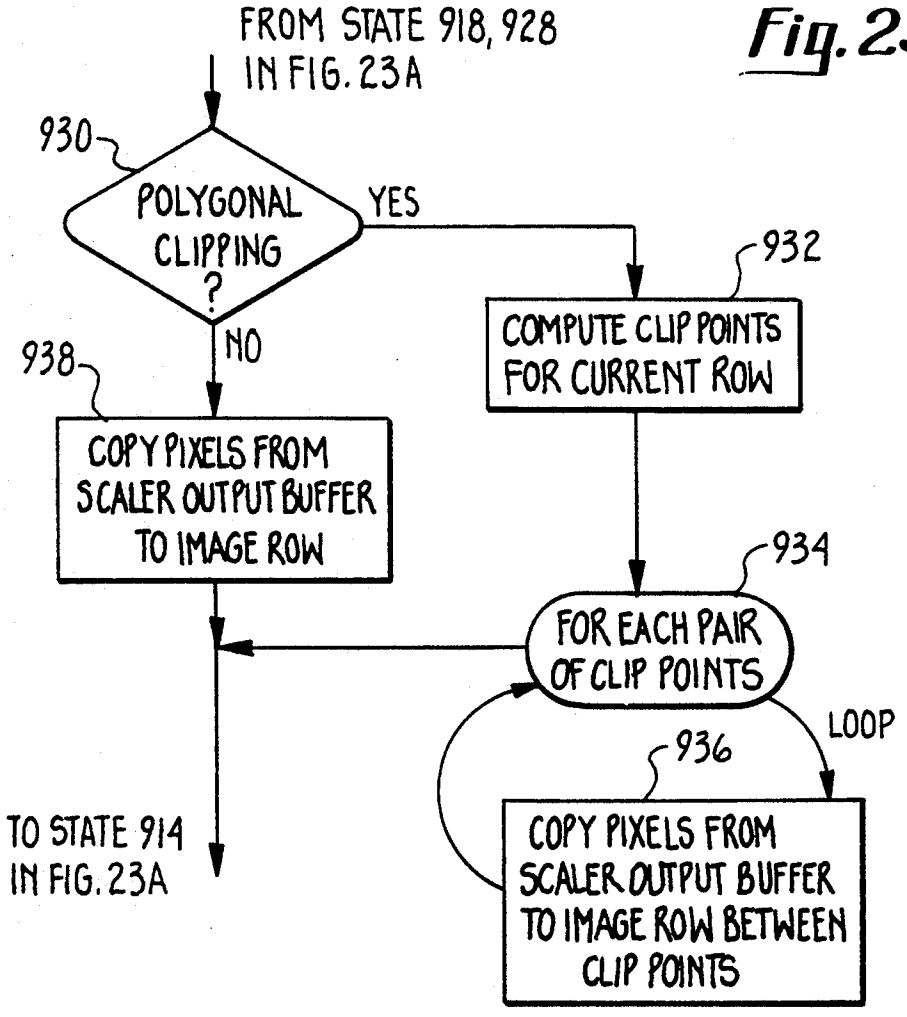
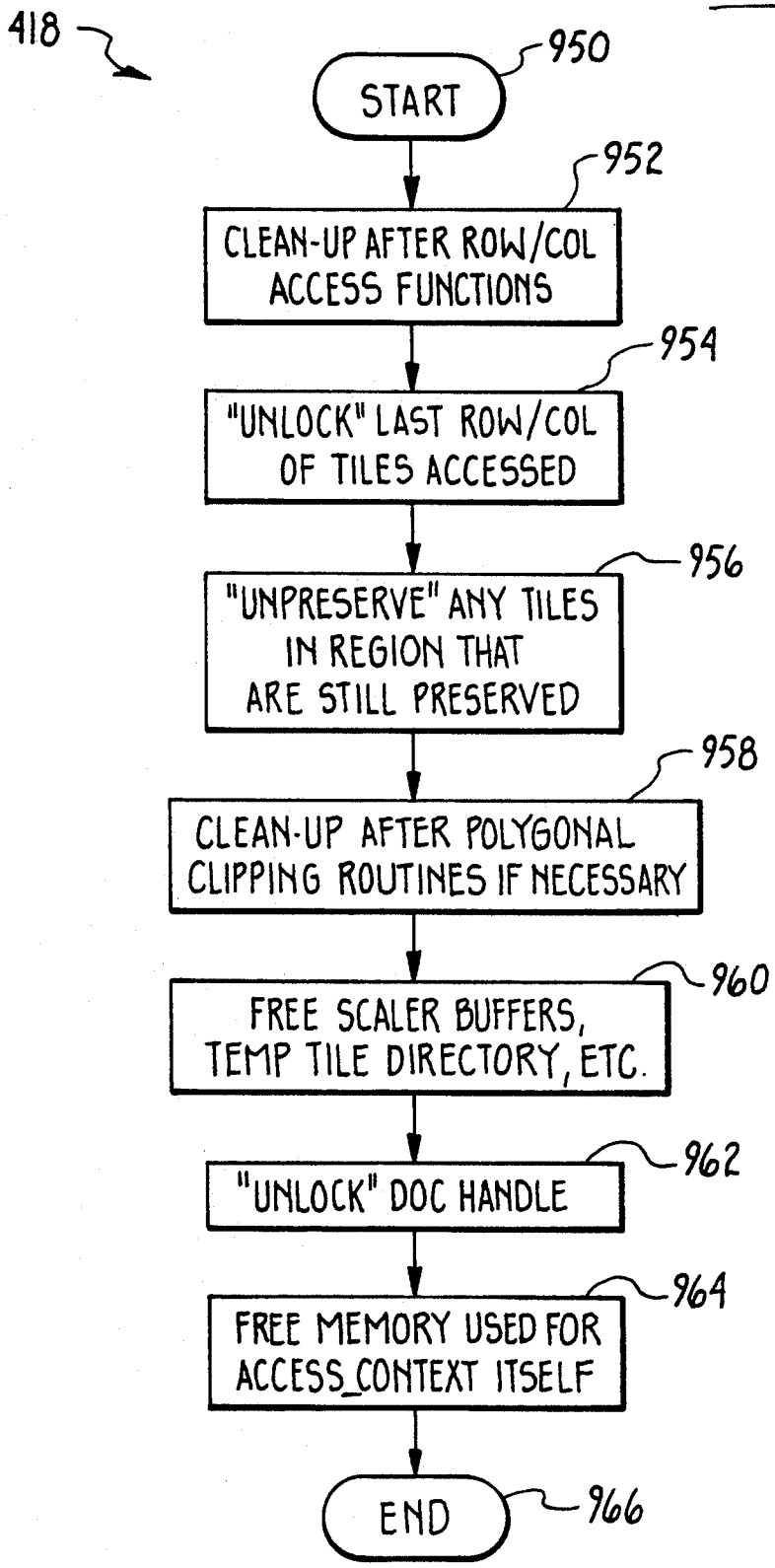

FROM STATE 918, 928 IN FIG. 23A
930
POLYGONAL CLIPPING ?
YES
NO
932
COMPUTE CLIP POINTS FOR CURRENT ROW
938
COPY PIXELS FROM SCALER OUTPUT BUFFER TO IMAGE ROW
934
FOR EACH PAIR OF CLIP POINTS
LOOP
936
COPY PIXELS FROM SCALER OUTPUT BUFFER TO IMAGE ROW BETWEEN CLIP POINTS
TO STATE 914 IN FIG. 23A

Fig. 23B

End Image Access

Fig. 24



UndoPreviousRasOp

Fig. 25A

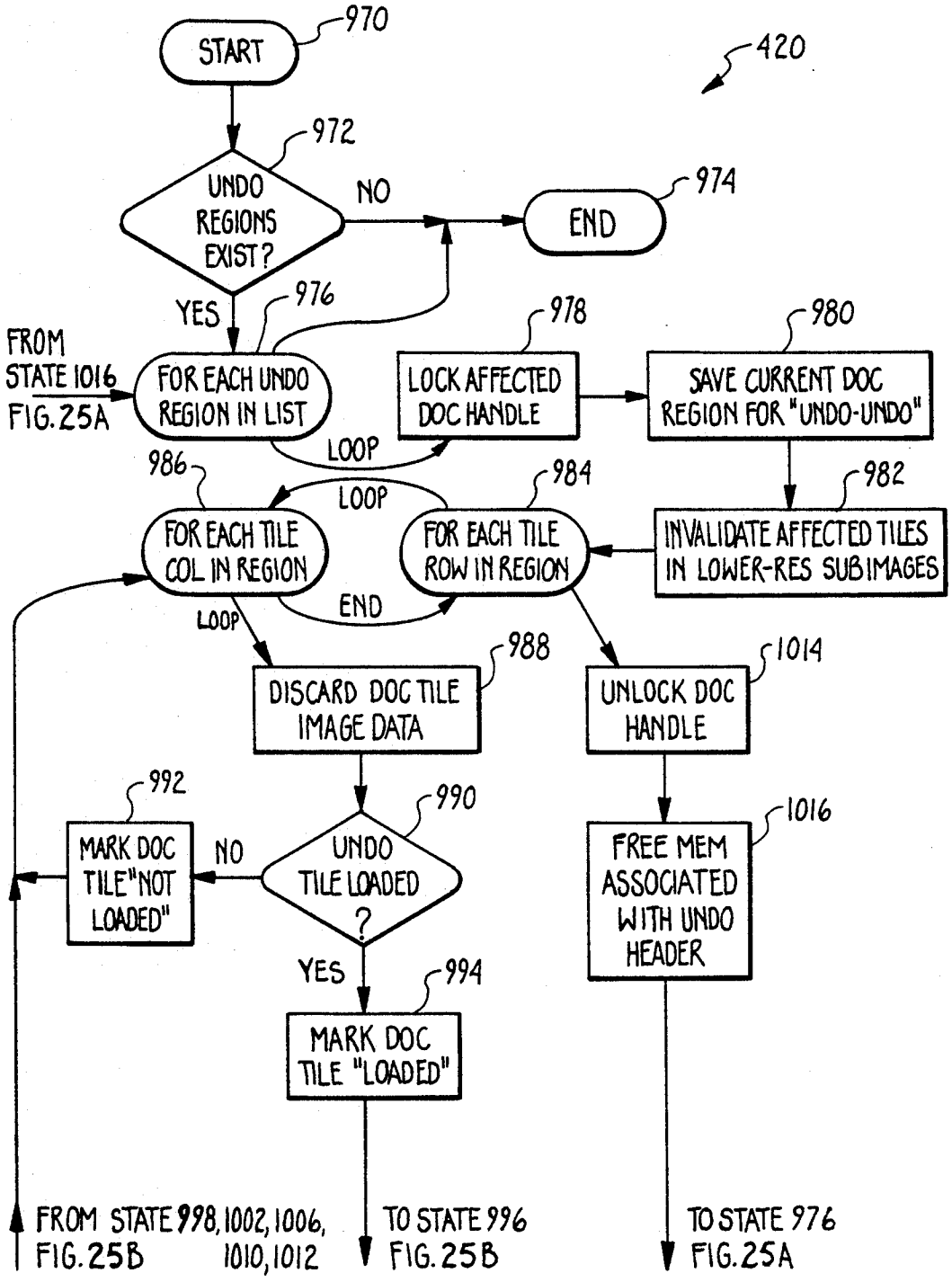
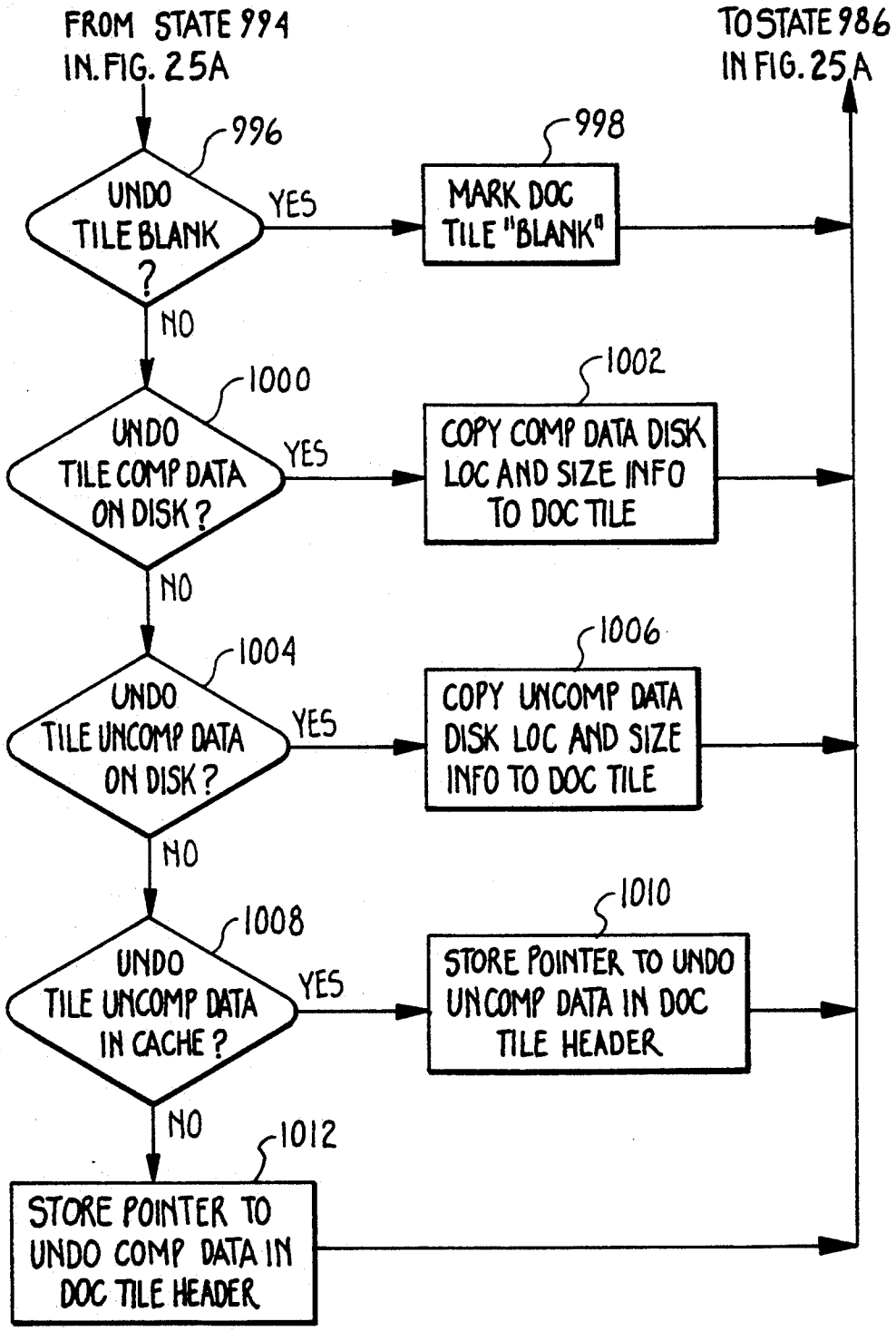
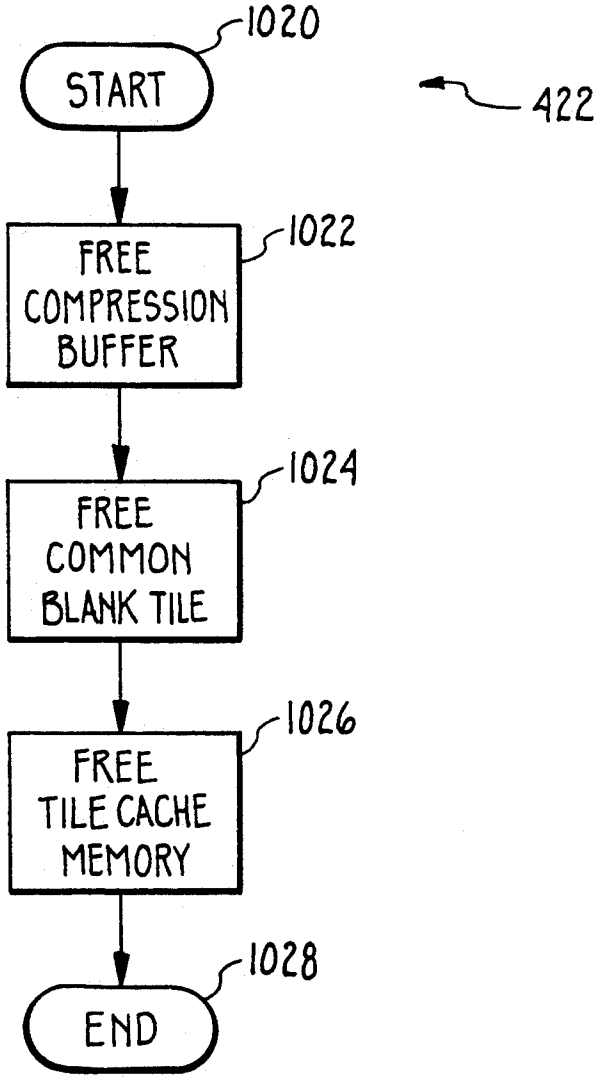


Fig. 25B


FROM STATE 994 IN. FIG. 25A
TO STATE 986 IN FIG. 25A
996
UNDO TILE BLANK ?
YES
998
MARK DOC TILE "BLANK"
NO
1000
UNDO TILE COMP DATA ON DISK ?
YES
1002
COPY COMP DATA DISK LOC AND SIZE INFO TO DOC TILE
NO
1004
UNDO TILE UNCOMP DATA ON DISK ?
YES
1006
COPY UNCOMP DATA DISK LOC AND SIZE INFO TO DOC TILE
NO
1008
UNDO TILE UNCOMP DATA IN CACHE ?
YES
1010
STORE POINTER TO UNDO UNCOMP DATA IN DOC TILE HEADER
NO
1012
STORE POINTER TO UNDO COMP DATA IN DOC TILE HEADER

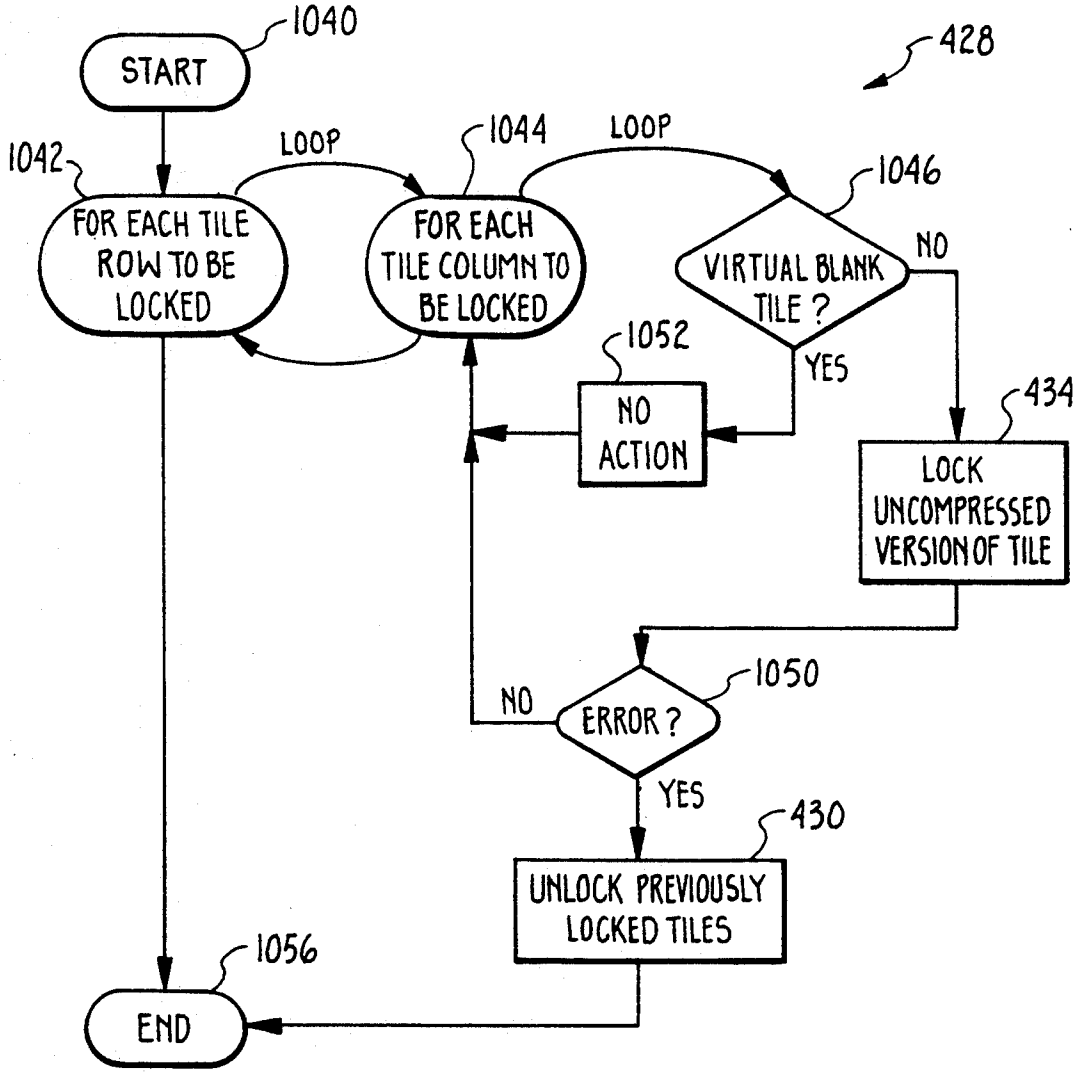
End Cache Manager

Fig. 26



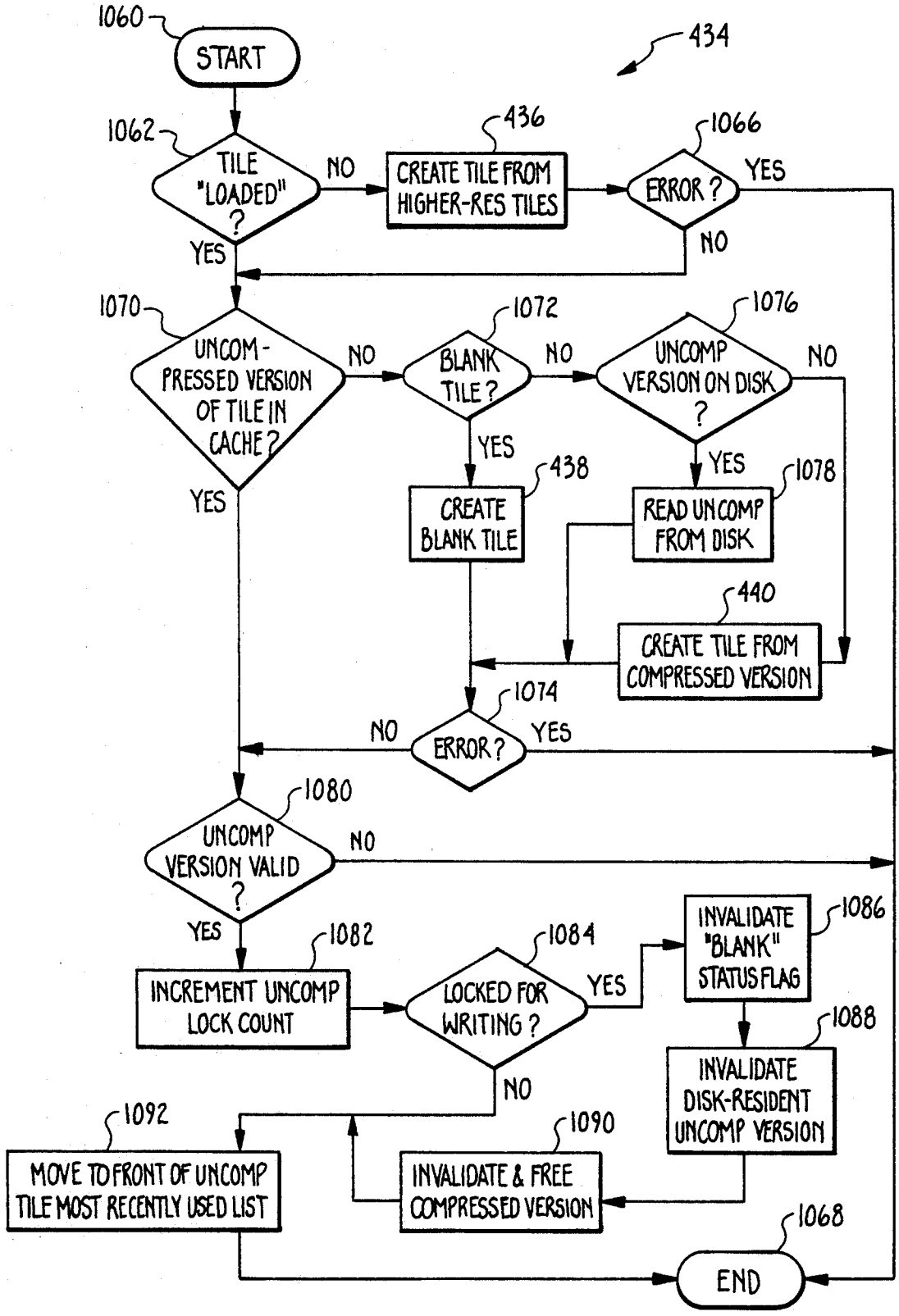
ExpTileLock

Fig. 27



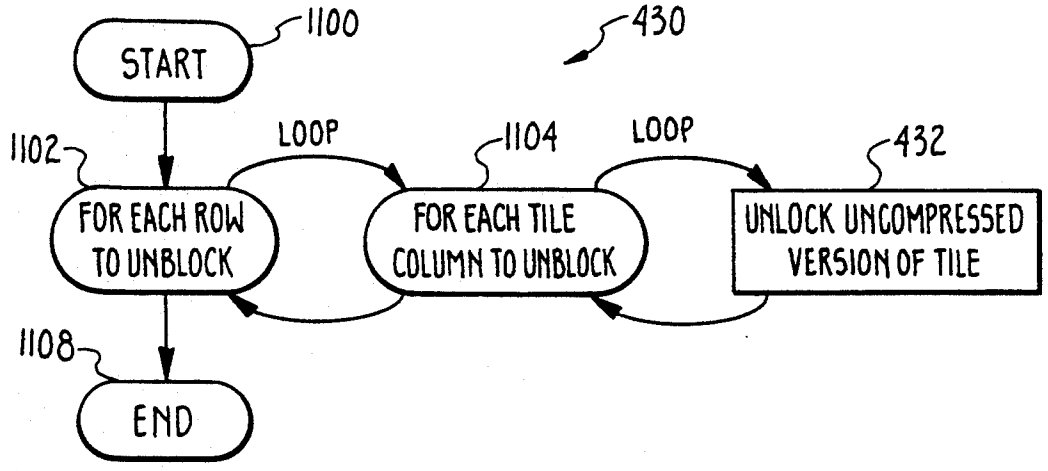
LockExpHandle

Fig. 28



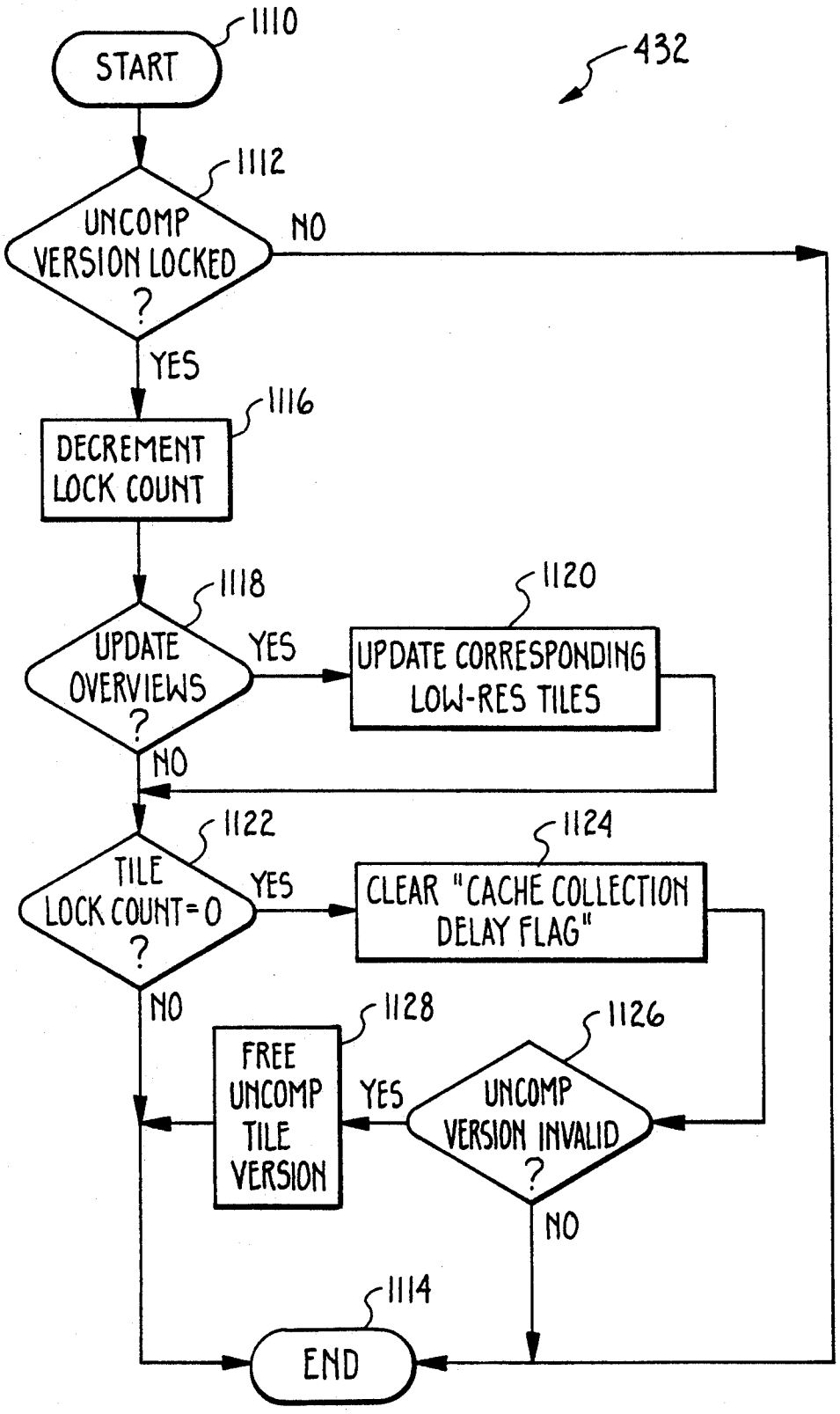
ExpTileUnlock

Fig. 29



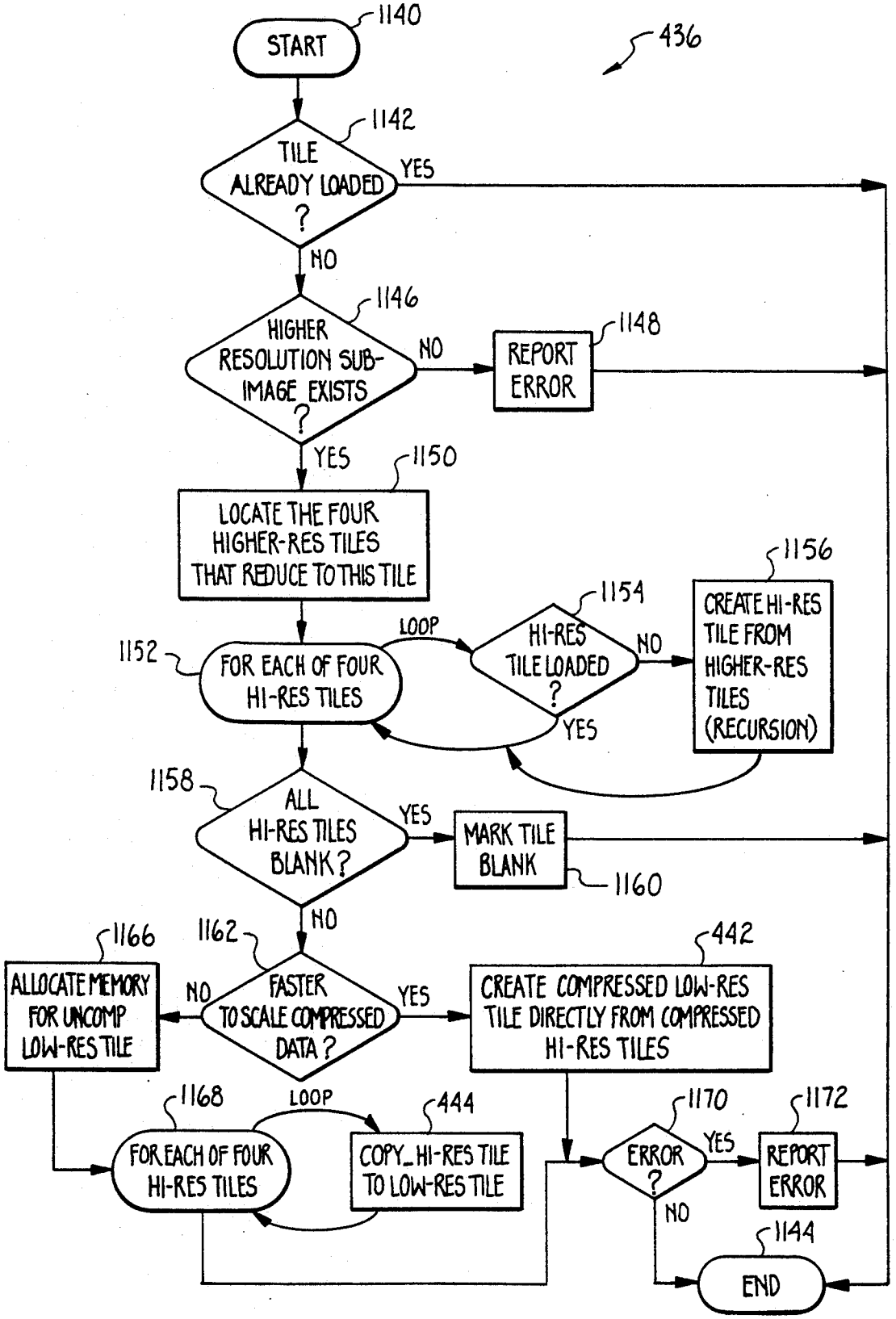
Unlock Exp Handle

Fig. 30



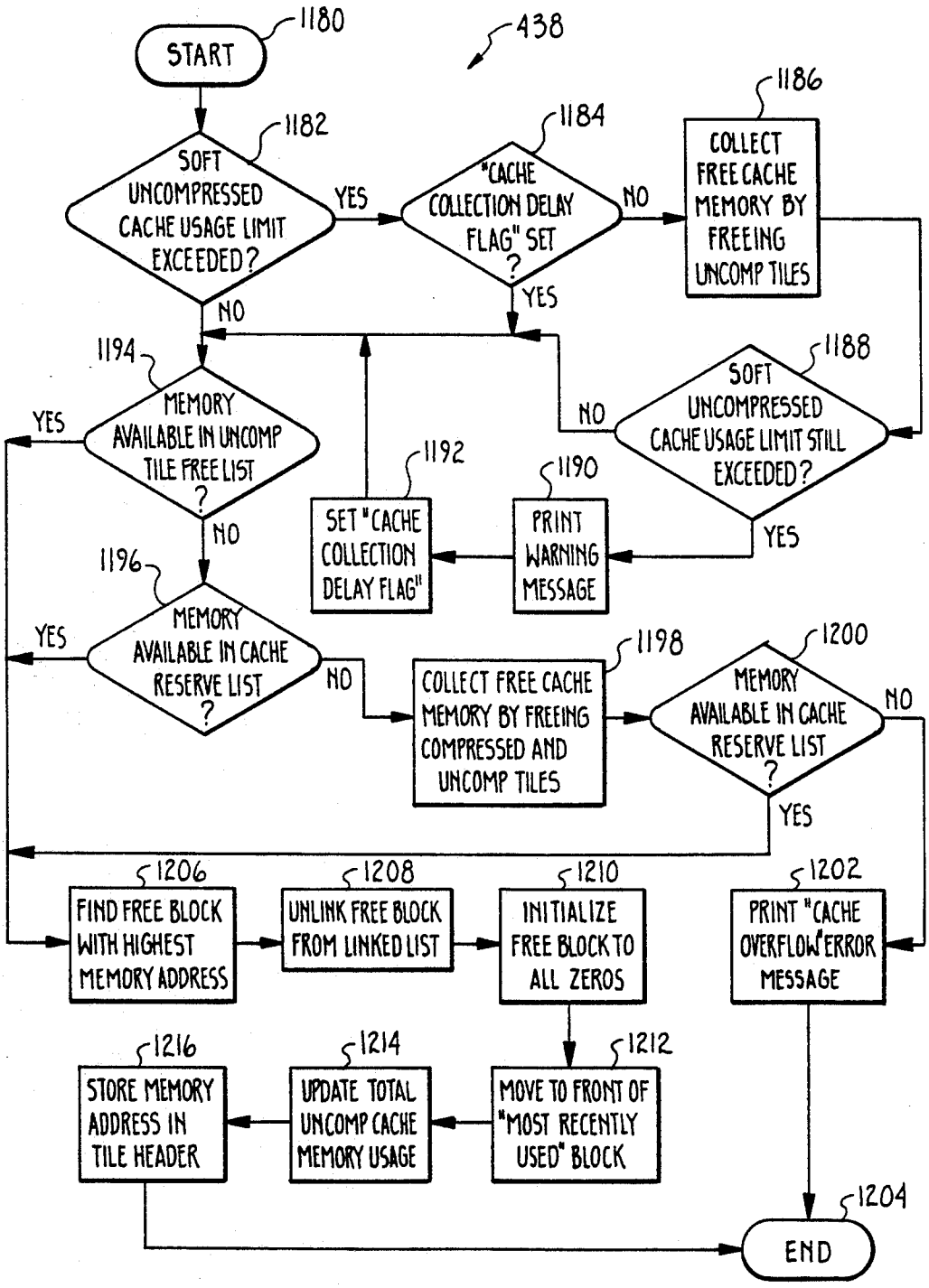
LoadSubImTile

Fig.31



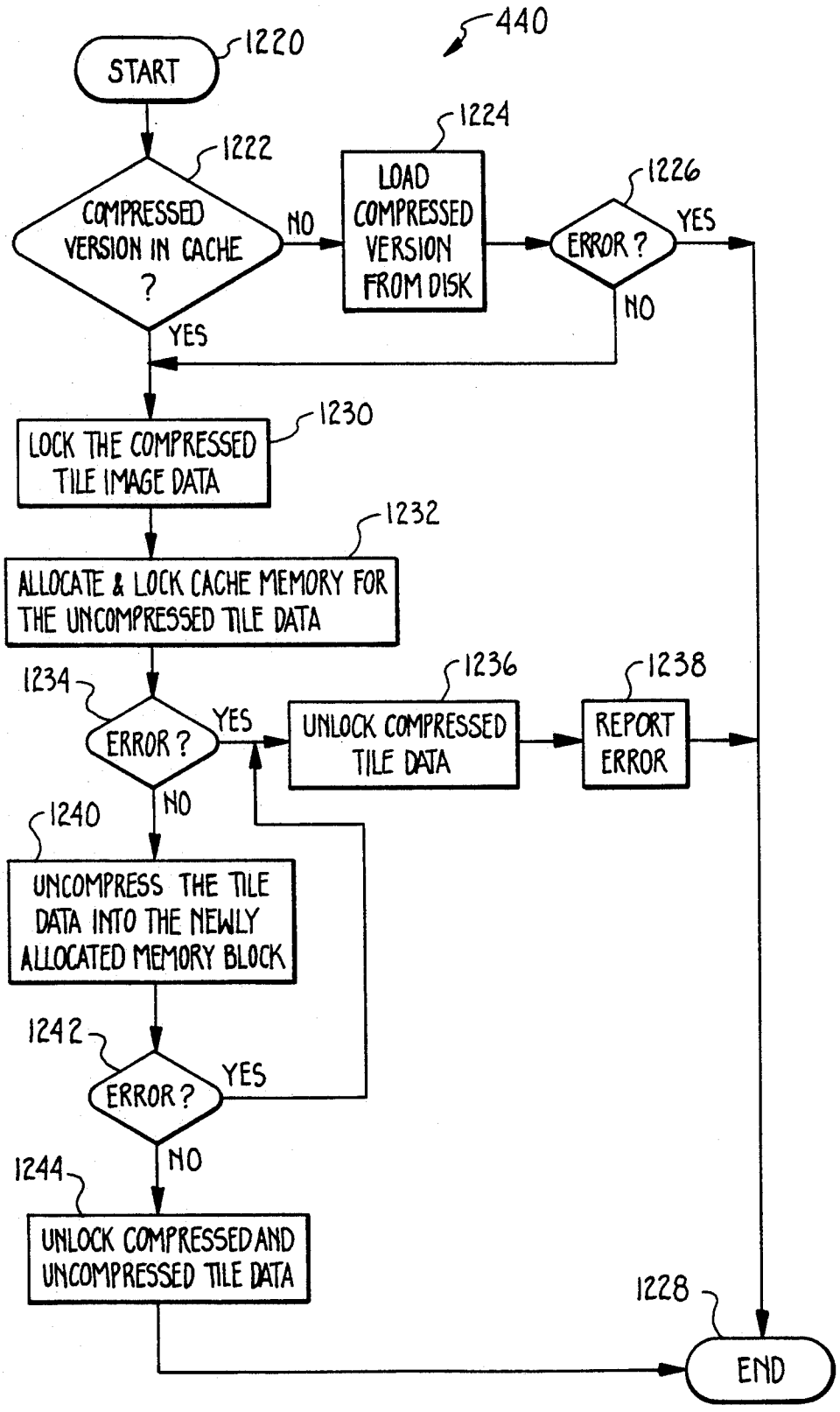
AllocExpHandle

Fig. 32



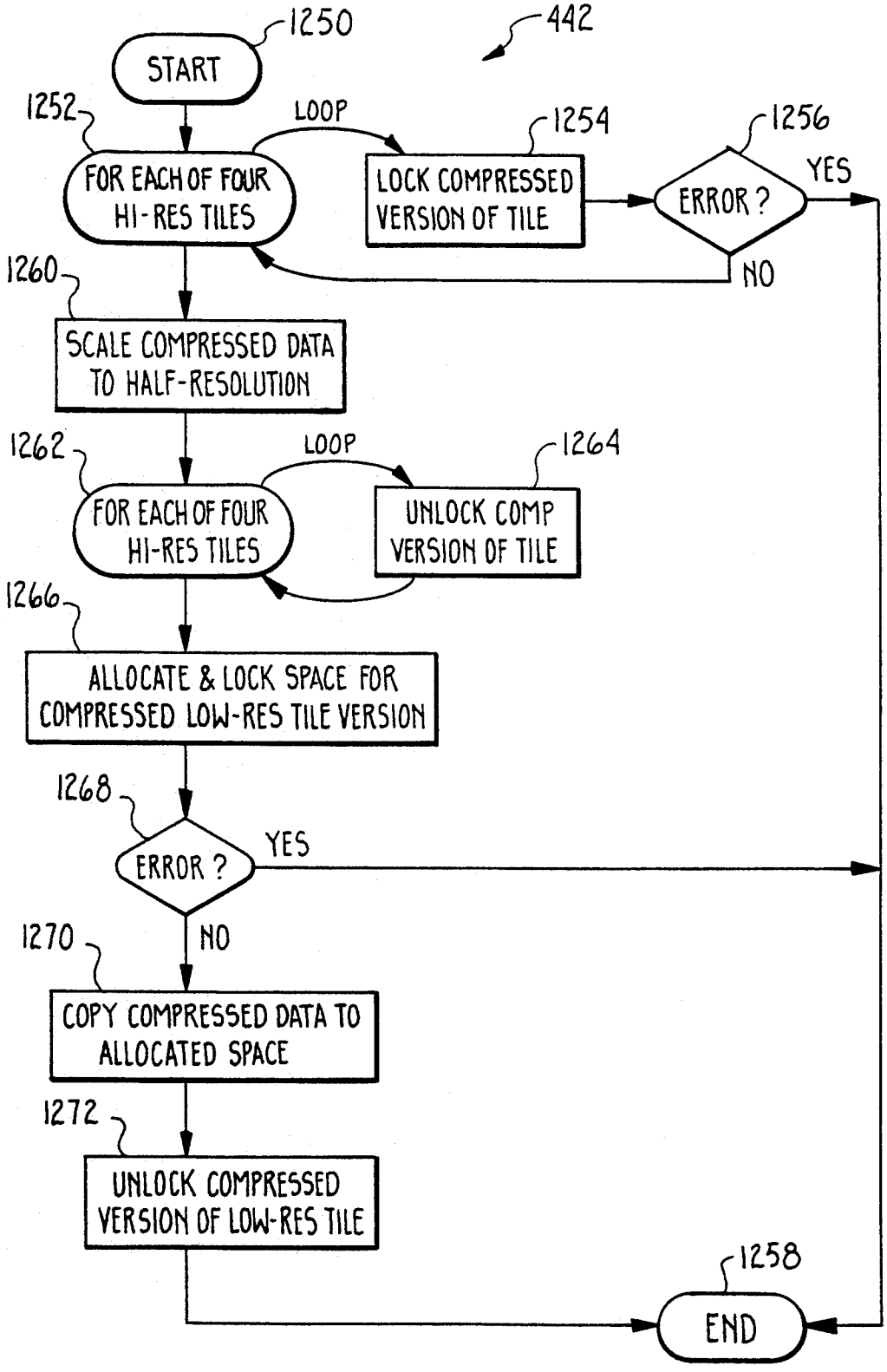
ExpandTile

Fig. 33



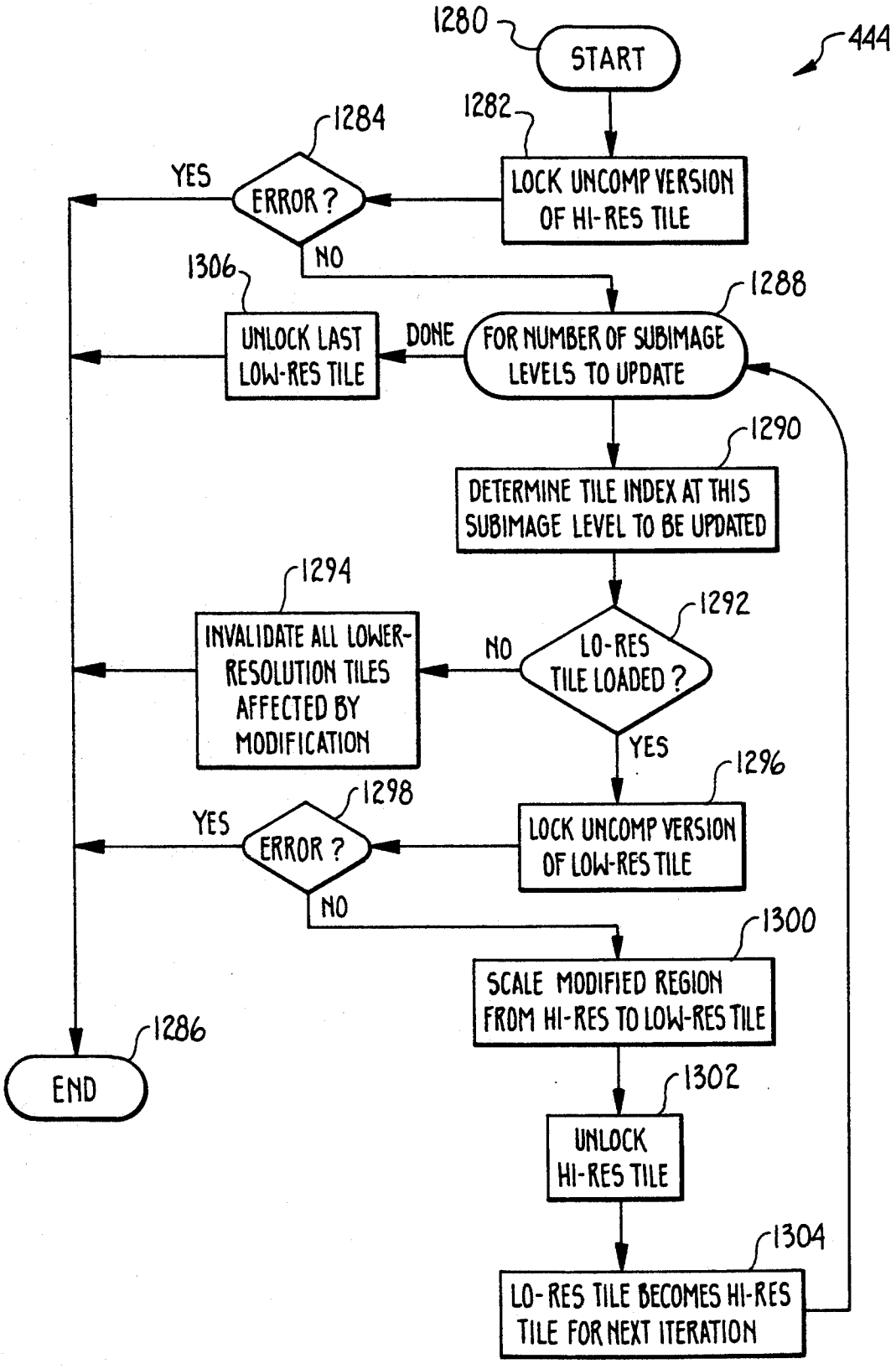
Comp Copy To Oview

Fig. 34



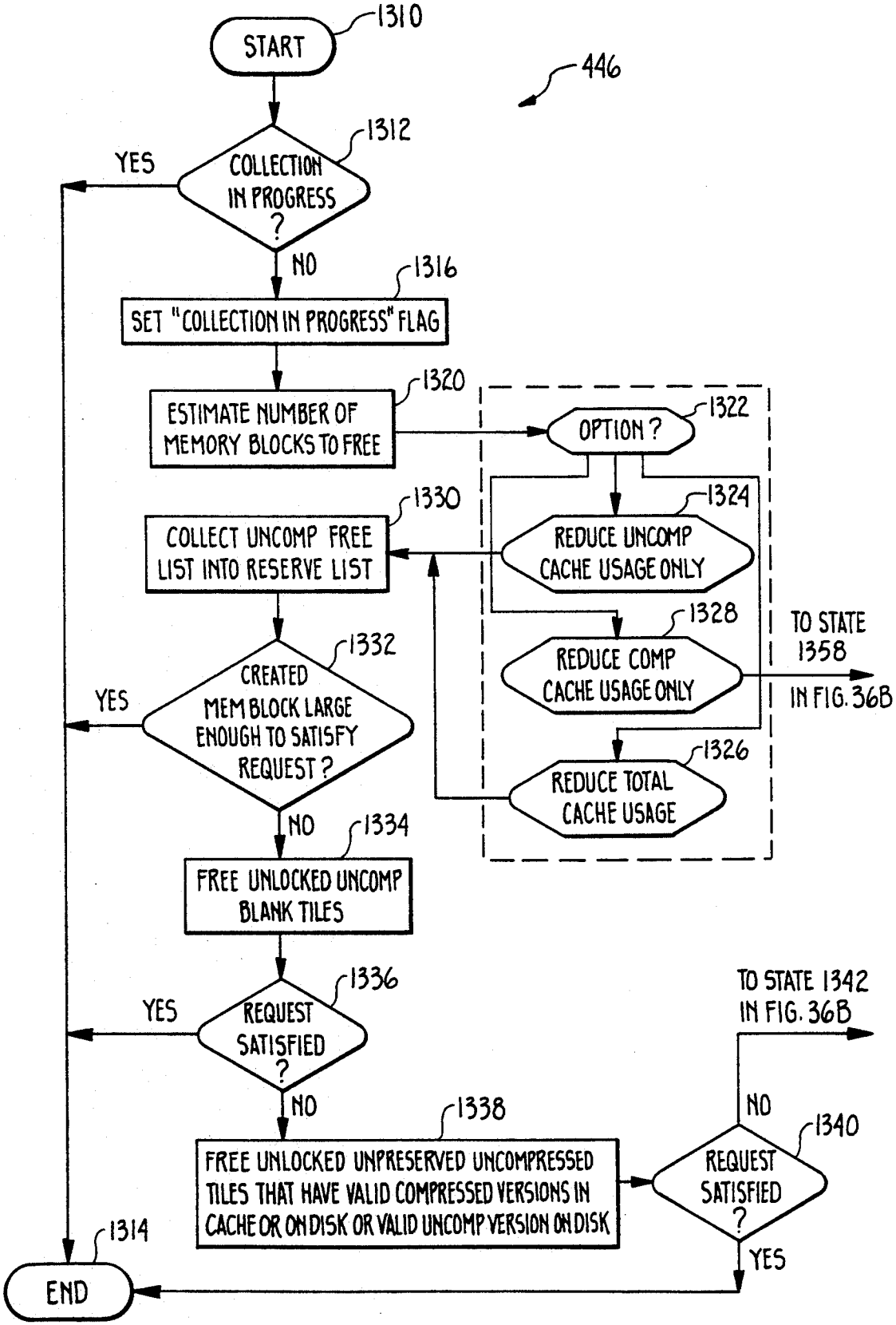
CopyTileToOview

Fig. 35



Collect Free Cache

Fig. 36A



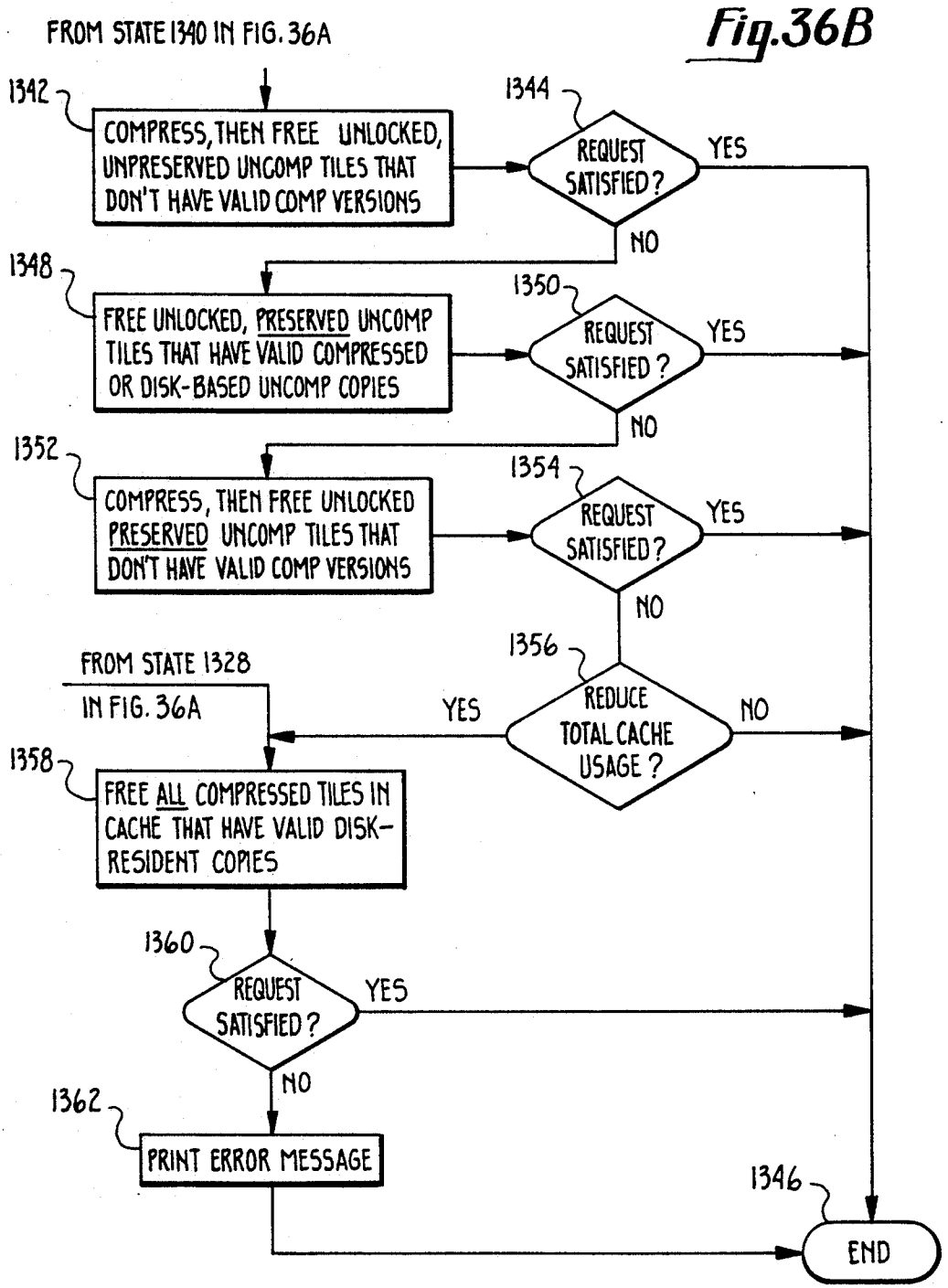
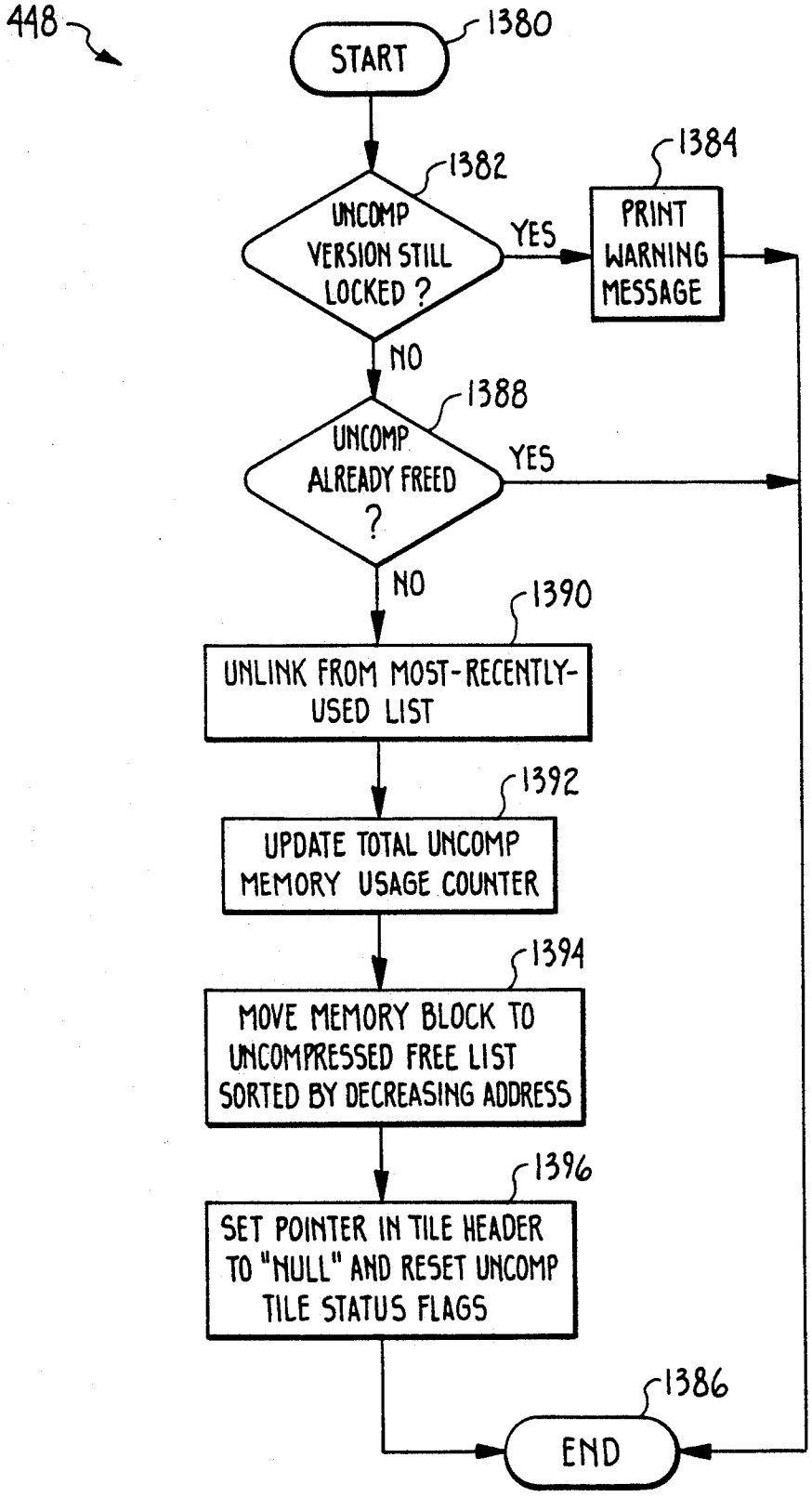

FROM STATE 1340 IN FIG. 36A
1342
COMPRESS, THEN FREE UNLOCKED, UNPRESERVED UNCOMP TILES THAT DON'T HAVE VALID COMP VERSIONS
1344
REQUEST SATISFIED?
YES
NO
1348
FREE UNLOCKED, PRESERVED UNCOMP TILES THAT HAVE VALID COMPRESSED OR DISK-BASED UNCOMP COPIES
1350
REQUEST SATISFIED?
YES
NO
1352
COMPRESS, THEN FREE UNLOCKED PRESERVED UNCOMP TILES THAT DON'T HAVE VALID COMP VERSIONS
1354
REQUEST SATISFIED?
YES
NO
FROM STATE 1328 IN FIG. 36A
1356
REDUCE TOTAL CACHE USAGE?
YES
NO
1358
FREE ALL COMPRESSED TILES IN CACHE THAT HAVE VALID DISK-RESIDENT COPIES
1360
REQUEST SATISFIED?
YES
NO
1362
PRINT ERROR MESSAGE
1346
END

Fig. 36B

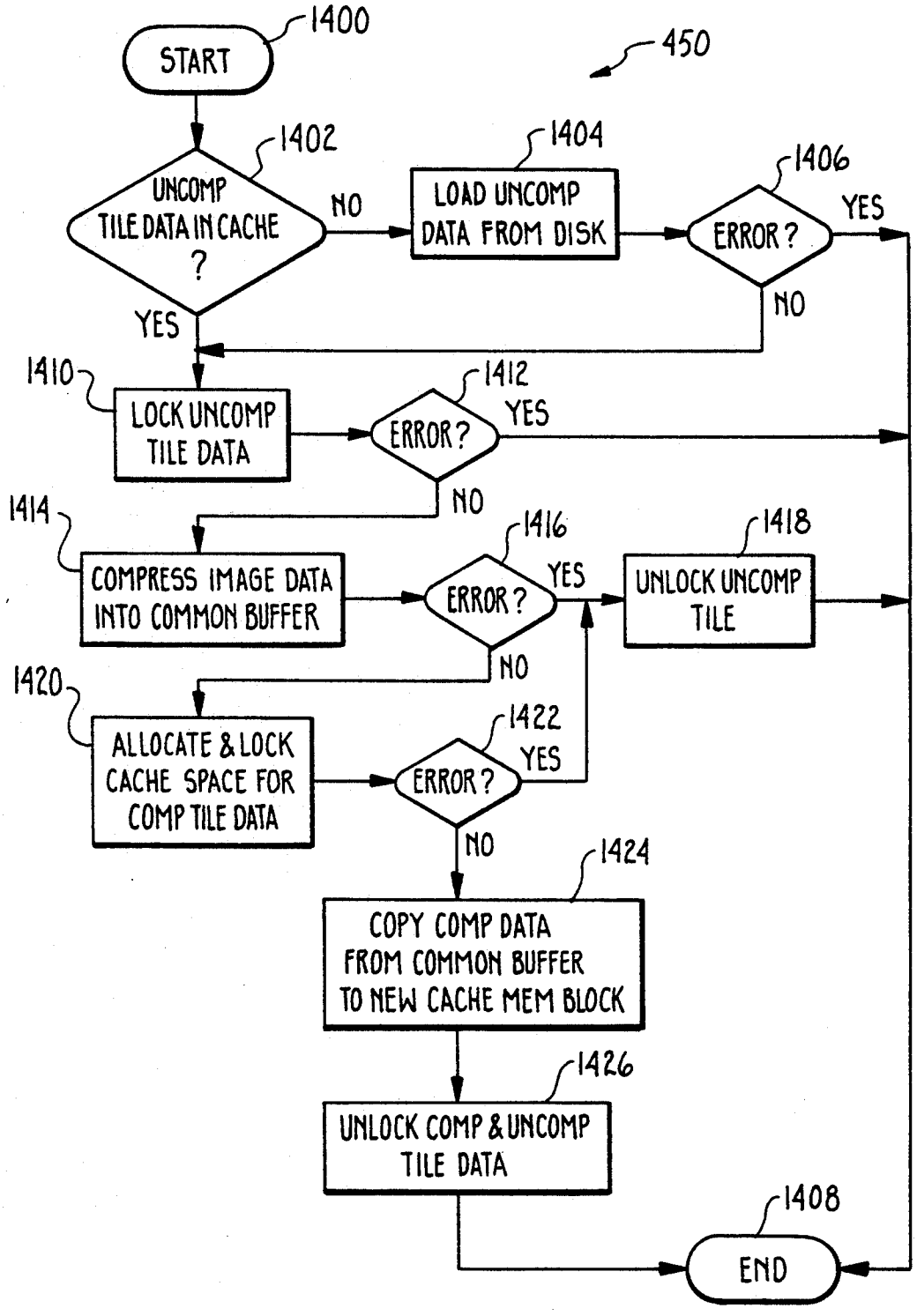
FreeExpHandle

Fig. 37



CompressTile

Fig. 38



SYSTEM FOR MANAGING TILED IMAGES USING MULTIPLE RESOLUTIONS

MICROFICHE APPENDIX

A microfiche appendix containing computer source code is attached. The microfiche appendix comprises one (1) sheet of microfiche having 74 frames.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to memory management systems and, more particularly, to the memory management of large digital images.

2. Description of the Prior Art

The present invention comprises a memory management system for large digital images. These digital, or raster, images are made up of a matrix of individually addressable pixels, which are ultimately represented inside of a computer as bit-maps. Large digital images, such as those associated with engineering drawings, topographic maps, satellite images, and the like, are often manipulated by a computer for the purpose of viewing or editing by a user. The size of, such images are often on the order of tens and even hundreds of Megabytes. Given the current cost of semiconductor memory it is economically impracticable to dedicate a random access memory (RAM) to storing even a single large digital image (hereinafter just referred to as a "digital image"). Thus, the image is usually stored on a slower, secondary storage medium such as a magnetic disk, and only the sections being used are copied into main memory (also called RAM memory).

However, as is well known by users of computer aided design ("CAD") systems, a simplistic memory transfer scheme will cause degraded performance during many typical operations, including zooming or panning. Essentially, during such operations, the computer cannot transfer data between disk and main memory fast enough so that the user must wait for a video display to be refreshed. Clearly, these periods of waiting on memory transfers are wasteful of engineering time.

Presently, to enhance main memory storage of only relevant sections of a digital image, the image is logically segmented into rectangular regions called "tiles". Two currently preferred standards for segmenting an image into tiles are promulgated by the Computer Aided Logistics Support (CALS) organization of the United States government (termed the "CALS standard" herein) and by Aldus Corporation of Seattle, Washington, as defined in the Tagged Image Format File (TIFF) definition (e.g., "TIFF Specification, Revision 5.0, Appendix L). Among other tile sizes, both standards define a square tile having dimensions of 512×512 pixels. Thus, if each pixel requires one byte of storage, the storage of one such tile would require a minimum of 256 kilobytes of memory.

Others, such as Thayer, et al. (U.S. Pat. No. 4,965,751) and Sawada, et al. (U.S. Pat. No. 4,920,504) have discussed tiling or blocking a memory. However, such computer hardware is generally associated with a graphics board for improving the speed of pixel transfers between a frame buffer and a video display by addressing a group of pixels simultaneously. These systems have no relationship to tiling of the image itself and thus do not require knowledge of image size. Tiling has also been used to refer to polygon filling as in Dalrymple, et al. (U.S. Pat. No. 4,951,230), which is unrelated to the notion of tiling discussed herein.

The patent to Ewart (U.S. Pat. No. 4,878,183) discusses interlaced cells, each cell containing one or more pixels, for storing continuous tone images such as photographs. The variable size cells are used to vary the resolution of an image according to a distance which is to be perceived by a user. However, the Ewart disclosure does not discuss rasterized binary images containing line drawings, nor does Ewart discuss virtual memory management for modifying or editing images, as will be more fully discussed below.

Even when stored in a mass storage system, an image library, containing a number of digital images, will consume disk space very quickly. Furthermore, "raw" digital images are generally too large to transfer from mass storage to portable floppy disks, or between computer systems (by telephone, for example), in a timely and inexpensive manner unless some means is used to reduce the size of the image. Hence, users of binary images employ image compression techniques to improve storage and transfer efficiencies. One existing compression standard applicable to facsimile transmission, CCITT Group IV, or T.6 compression, is now being used for digital images. Like many other compression techniques, however, the CCITT standard uses statistical techniques to compress data and, hence, it does not always produce a compressed image that is smaller than the original, uncompressed image. That means that image libraries will often contain a mix of compressed and uncompressed binary images. Similar compression standards exist for color and gray-scale images such as those promulgated by the JPEG (Joint Photog. Exp. Group) Standards Committee of the CCITT as SGV III Draft Standard.

At the present time, digital images are typically viewed and modified with an image editor using an off-the-shelf computer workstation. These workstations usually come with a sophisticated operating system, such as UNIX, that employs a virtual memory to effectively manage memory accesses in secondary and main memories. In an operating system having virtual memory, the data that represents the executable instructions for a program or the variables used by that program do not need to reside entirely in main memory. Instead, the operating system brings portions of the program into main memory only as needed. (The data that is not stored in main memory being stored on magnetic disk or other like nonvolatile memory.) The address space that is available to any one application program is generally managed in blocks of convenient sizes called "pages" or "segments".

In general, a virtual memory system allows application programs to be written and executed without concern for the management of virtual memory carried out by the operating system. Thus, independence of the size of main memory is achieved by creating a "virtual" address space for the program. The operating system translates virtual addresses into physical addresses (in a main or cache memory) with the aid of an "address translation table". This table contains one entry per virtual memory segment of status information. For instance, segment status will commonly include information about whether a segment is currently in main memory, when a segment was last used, a disk address at which the disk copy of the segment resides, and a RAM address at which the segment resides (only valid if the segment is currently loaded in main memory).

When the program attempts to access data in a segment that is not currently resident in main memory, the operating system reads the segment from disk into main memory. The operating system may need to discard another segment to make room for the new one (by overwriting the area of main memory occupied by the old segment), so some method of determining which segment to discard is required. Usually the method is to discard the least recently used segment. If the discarded segment was modified then it must be written back to disk. The operating system completes the "swap" operation by updating the address translation table entries of the new and discarded segments.

In summary, the conventional memory management schemes consider data to be in one of two states: resident or not resident in main memory. Which segments are stored in main memory at any given time is generally determined only by past usage, with no way of predicting future memory demands. For instance, just because a segment is the least recently used does not mean that it will not be used at the very next memory access.

However, the management of virtual memory for images departs significantly from conventional virtual memory schemes because images and computer programs are accessed in very different ways. Computer programs tend to access one small neighborhood of virtual address, and then jump to some distant, essentially random, location. However, during normal image processing operations an image is accessed in one of a finite set of predictable patterns. It is not surprising then that conventional memory management systems can significantly degrade performance when used in image processing applications by applying inappropriate memory management rules. Rules which should be abided by a memory management system for large digital images are the following:

1. Image memory must be managed as rectangular image regions (called "tiles"), not as linear memory address ranges.

2. An image tile can exist in five forms: uncompressed memory-resident, compressed memory-resident, uncompressed disk-resident, compressed disk-resident and "can be derived from other available image tiles", in contrast to the two basic forms of memory-resident and disk-resident available in conventional virtual memory schemes.

3. The image region that will be affected by a particular image processing operation is known before the operation begins, and that information can be conveyed to the memory manager.

4. An image memory manager must be tunable to different system capabilities and image types. For example, many computers can decompress a tile of binary data much faster that they can retrieve the uncompressed version of the same tile from disk. On the other hand, some images cannot be compressed at all.

5. An image memory management system should support the capability to "undo" editing operations which is built into the memory manager for optimal performance and ease of use. Thus, the memory manager could easily save copies of the compressed tiles in the affected region, and quickly restore the image to the original state by simply modifying the tile directory entries to point to the old version.

Reader, et al., ("Address Generation and Memory Management for Memory Centered Image Processing Systems", SPIE, Vol. 757, Methods for Handling and Processing Imagery, 1987) discuss a primitive memory management system for images. However, in that system, image tiles are only stored in memory and not on disk. Furthermore, in the Reader, et al., system, there is no capability to handle images in compressed form, nor is there any discussion of "undoing" editing operations.

Consequently, a need exists for an image memory management system that provides: linkages with a raster image editor which includes modify and undo operations, true virtual memory for large images specifying locations on disk and in memory, simultaneous handling of compressed and uncompressed images, and a method for rapidly constructing reduced resolution views of the image for display. The latter need is particularly important when viewing a large image reduced to fit on a video display.

SUMMARY OF THE INVENTION

The above-mentioned needs are satisfied by the present invention which includes a memory management system for tiled images. The memory management system includes a tile manager for maintaining a virtual memory comprising a main memory and a secondary memory such as a disk. The tiled images may include tiles in compressed or uncompressed form.

The tile manager selects the form of image tile that most appropriately matches a request. Each tile of an image may exist in one or more of five different forms, or states, as follows: uncompressed and resident in the image data cache, compressed and resident in the image data cache, uncompressed and resident on disk, compressed and resident on disk and not loaded but re-creatable using data from higher-resolution image tiles.

An image stack having successively lower-resolution subimages is constructed from a full resolution source image. The lower-resolution images in the image stack may be used to enhance such standard image accesses as zooming and panning where high speed image reduction is advantageous.

The image memory management system provides linkages with image processing applications that facilitate image modifications. The tile manager need only store compressed tiles that relate to so-called undoable operations.

These and other objects and features of the present invention will become more fully apparent from the following description and appended claims taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a perspective view of an image stack comprising full, half, quarter and eighth resolution tiled images;

FIG. **2** is a full resolution image of a mechanical part;

FIG. **3** is a half resolution image of the mechanical part shown in FIG. **2**;

FIG. **4** is a quarter resolution image of the mechanical part shown in FIG. **2**;

FIG. **5** is an eighth resolution image of the mechanical part shown in FIG. **2**;

FIG. **6** is a block diagram showing one preferred embodiment of a computer system that includes the present invention;

FIG. **7** is a memory map showing the general arrangement of cache memory according to the present invention;

FIG. 8 is a state diagram defining the flow of tile data between different storage states according to the present invention;

FIGS. 9A and B are a diagram of one preferred data structure defining document information according to the present invention;

FIG. 10 is a diagram of one preferred data structure defining a tile header for maintaining the status of compressed or uncompressed tiles;

FIG. 11 is a diagram of a partial calling hierarchy for the various functions of the presently preferred embodiment of the tile manager of the present invention;

FIG. 12 is a flow diagram of one preferred embodiment of the tile manager;

FIG. 13 is a flow diagram defining the "initialize cache manager" function referred to in the flow diagram of FIG. 12;

FIG. 14 is a state diagram of the locking and unlocking of a memory, state, according to the present invention;

FIGS. 15A, 15B, and 15C are a flow diagram defining the "create image access context" function referred to in FIG. 12;

FIG. 16 is a diagram, of a data structure defining the access context referred to in FIGS. 15A,B;

FIGS. 17A and 17B are a flow diagram defining the "save region for undo" function referred to in FIG. 15B;

FIG. 18 is a flow diagram defining the "load tiled raster image" function referred to in FIG. 12;

FIG. 19 is a flow diagram defining the "load TIFF subimage tile information into tile headers" function referred to in FIG. 18;

FIG. 20 is a flow diagram defining a "store tile info in tile headers" function referred to in FIG. 12;

FIG. 21 is a flow diagram defining the "begin undoable raster operation" function referred to in FIG. 12;

FIGS. 22A and 22B are a flow diagram defining the "read rows from region" function referred to in FIG. 12;

FIGS. 23A and 23B are a flow diagram defining the "write rows to region" function referred to in FIG. 12;

FIG. 24 is a flow diagram defining the "close image access context" function referred to in FIG. 12;

FIGS. 25A and 25B are a flow diagram defining the "undo previous raster operations" function referred to in FIG. 12;

FIG. 26 is a flow diagram defining the "quit cache manager" function referred to in FIG. 12;

FIG. 27 is a flow diagram defining the "lock expanded image tile group" function referred to in FIG. 22A;

FIG. 28 is a flow diagram defining the "lock expanded tile" function referred to in FIG. 27;

FIG. 29 is a flow diagram defining the "unlock expanded image tile group" function referred to in FIG. 27;

FIG. 30 is a flow diagram defining the "unlock expanded tile" function referred to in FIG. 29;

FIG. 31 is a flow diagram defining the "create tile from higher-resolution tiles" function referred to in FIG. 28;

FIG. 32 is a flow diagram defining the "allocate space for uncompressed version of tile" function referred to in FIG. 28;

FIG. 33 is a flow diagram defining the "create uncompressed version of tile from compressed version" function referred to in FIG. 28;

FIG. 34 is a flow diagram defining the "create compressed low resolution tile from compressed higher-resolution tiles" function referred to in FIG. 31;

FIG. 35 is a flow diagram defining the "copy uncompressed high resolution tile to uncompressed low resolution tiles" function referred to in FIG. 31;

FIGS. 36A and 36B are a flow diagram defining the "collect freeable cache memory" function referred to in FIG. 32;.

FIG. 37 is a flow diagram defining the "free uncompressed version of tile" function referred to in FIGS. 36A,B; and

FIG. 38 is a flow diagram defining the "create compressed version of tile from uncompressed version" function referred to in FIG. 17B.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference is now made to the drawings wherein like parts are designated with like numerals throughout.

FIG. 1 illustrates an image stack, generally indicated at 100. The design of the image stack 100 is based on the idea that image memory can be managed as small square regions, called tiles, that are mostly independent of one another. In general, a tile may be either uncompressed (also termed expanded) or compressed. While the basic uncompressed tile size could be a variable, it is presently preferred to be fixed at 32 kilobytes, or 512 pixels by 512 pixels to conform with the Computer Aided Logistics Support (CALS) raster file format standard for binary images. (Note that the present invention allows binary and color images to coexist in a common image memory management system.)

In order to compensate for lower performance expected with a virtual memory management system for images, particularly when reducing large portions (by combining pixels) of the image for display, the present invention automatically maintains a series of reduced resolution copies, called subimages, of the full resolution image. Preferably, the resolution (i.e., pixels per inch) of each subimage is reduced by exactly half relative to the next higher-resolution subimage. Thus, the image stack 100 can be visualizing as an inverted pyramid, wherein the images can be stacked beginning with a full resolution subimage (or image) 102 at the top, followed by a half resolution subimage 104, then a quarter resolution subimage 106, and an eighth resolution subimage 108. (In FIG. 1, the subimages 102–108 are outlined by bolded lines.)

The subimages 102, 104, 106, 108 are superimposed on a set of tiled subimages 110*a*, 110*b*, 110*c*, 110*d*, respectively, defining sets of tiles. The extent of the image stack 100 ends at the resolution that allows the entire subimage to be stored within a single tile 108 (preferably 512×512 pixels square). Each lower-resolution subimage 104–108 is a faithful representation of the full resolution subimage 102 at all times, with the exception of certain times during operations that modify the appearance of the full resolution subimage 102.

FIG. 2 illustrates an 8½″×11″, A-size mechanical drawing (to scale) as the full resolution subimage 102 showing a mechanical part 120*a*. Of course, other larger drawings such as, for example, D-size and E-size may be used by the present invention. Also, other image processing applications besides mechanical drawings may be used with the present invention including electrical schematics, topographical maps, satellite images, hea-

ting/ventilating/air conditioning (HVAC) drawings, and the like.

FIG. 3 illustrates the corresponding half resolution subimage **104** showing the half resolution part **120***b*. FIG. 4 illustrates the corresponding quarter resolution subimage **106** showing the quarter resolution part **120***c*. Lastly, FIG. 5 illustrates an eighth resolution subimage **108** showing the eighth resolution part **120***d*. In the preferred embodiment, reduced resolution subimages can be used any time that a reduction factor of 2:1 or higher would be used to scale a region of interest in the full resolution subimage **102** for display, plotting or copying.

The subimages **102–108** can be loaded from a source image file, if they exist, or they can be created on demand by the image memory management system of the present invention. The present invention includes editing capabilities that allow a user to trade off between "quick flash" pan/zoom performance and file size as measured by the number of reduced resolution subimages stored with each image. Depending on the application, the user will normally opt to store one or more reduced resolution subimages with each source image file.

The lower-resolution subimages, for example, subimages **104–108**, are utilized by the image memory management system to produce the illusion of instant access to any region of the image at any scale factor (not just the scale factor of the overview subimage). Increasing the number of lower-resolution subimages gives a higher quality "first flash" image during panning and zooming and reduces the time to get the final version of the image to the screen.

FIG. 6 illustrates a computer workstation generally indicated at **150** which is representative of the type of computer that is used with the present invention. The workstation **150** comprises a computer **152**, a color monitor **154**, a mouse **156**, a keyboard **158**, a floppy disk drive **160**, a hard disk drive **162** and an Ethernet communications port **164**. The computer **152** includes a motherboard bus **166** and an I/O bus **168**. The I/O bus **168**, in one preferred embodiment, is an IBM PC/AT ® bus, also known as an Industry Standard Architecture (ISA) bus. The two buses **166**, **168** are electrically connected by an I/O bus interface and controller **170**.

The I/O bus **168** provides an electromechanical communication path for a number of I/O circuits. For example, a graphics display controller **172** connects the monitor **154** to the I/O bus **168**. In the presently preferred embodiment, the monitor **154** is a 19-inch color monitor having a 1,024×768 pixel resolution. A serial communications controller **174** connects the mouse **156** to the I/O bus **168**. The mouse **156** is used to "pick" an image entity displayed on the monitor **154**.

The I/O bus **168** also supports the hard disk drive **162**, and the Ethernet communications port **164**. A hard disk controller **176** connects the hard disk drive **162** to the I/O bus **168**. The hard disk drive **162**, in one possible configuration of the workstation generally indicated at **150**, stores 60 megabytes of data. An Ethernet communications controller **178** connects an Ethernet communications port **164** with the I/O bus **168**. The Ethernet communications controller **178** supports the industry standard communications protocol TCP/IP which includes FTP and Telnet functions. The Ethernet communications port **164** of the preferred embodiment allows the Workstation **150** to be connected to a network which may include, among other things, a document scanner (not shown) and a print server (not shown).

The motherboard bus **166** also supports certain basic I/O peripherals. For example, the motherboard bus **166** is connected to a keyboard and floppy disk controller **180** which supports the keyboard **158** and the floppy disk drive **160**. The floppy disk drive **160**, in one present configuration, can access floppy disks which store up to 1.2 megabytes of data.

The fundamental processing components of the computer **152** are a microprocessor **182** such as, for example, an 80386 microprocessor manufactured by Intel, a math coprocessor **184** such as, for example, a 80387 math coprocessor also manufactured by Intel and a main memory generally indicated at **186** comprising, for example, 4 megabytes of random access memory (RAM). The main memory **186** is used to store certain computer software including a Unix compatible operating system **188** such as, for example, SCO Xenix licensed by Santa Cruz Operation of Santa Cruz, California, a subsidiary of Microsoft Corporation, an image processing application **190**, a tile manager **192**, and an image data cache **194**. The image processing application **190** includes editing functions such as zoom and pan.

Another presently preferred computer workstation **150** having somewhat different processing components from those just described is available from Sun Microsystems, Inc. of Mountain View, California, under the tradename "SPARCstation 1". In such an embodiment, the UNIX compatible operating system would be licensed directly from Sun.

Although a representative workstation has been shown and described, one skilled in the applicable technology will understand that many other computer and workstation configurations are available to support the present invention.

FIG. 7 illustrates a representative configuration of the image data cache **194** some time after the tile manager **192** (FIG. 6) begins operation. A set of compressed tiles **222** are kept at the low addresses of the image data cache **194**, and a set of uncompressed (or expanded) tiles **224** at the high addresses of the image data cache **194**. The terms expanded or uncompressed are used interchangeably. In between the two sets of tiles **222**, **224** is a reserved area **226** (free cache memory). As the operation of the tile manager **192** continues, the image data cache **194** becomes more unordered. As the cache requirement for compressed or uncompressed tiles increases, each set of tiles **222**, **224** approach the reserve area **226** from each end. In fact, the reserve area **226** can become completely exhausted.

Since the memory management schemes that apply to compressed data allocation are very different from that of uncompressed data, it is desirable to keep the two sets of tiles **222**, **224** separate. Compressed tiles are variable sized tiles (blocks of memory) **222***a,b,c,d,e,f* whereas the uncompressed tiles are all fixed sized tiles **224***a,b,c,d* and therefore the locations of the fixed sized tiles **224** are interchangeable. Linked lists of allocated memory are kept sorted according to size and address for compressed tiles. The number of linked lists is a variable number but presently there are about 64 different size categories for compressed tiles and only one size category for uncompressed tiles (for binary images).

To use the image data cache **194**, the memory management functions begin by determining how much fast memory (RAM) and slow memory (disk or host memory) is available for image memory uses. When an image

is loaded, the system allocates memory for image information and related tile directory structures. Cache management parameters are modified as necessary to balance the requirements for expanded tile and compressed tile cache memory. The expanded tile cache memory pool and the compressed tile cache memory pool allow tiles from different images to intermingle. Expanded and compressed tiles are kept in separate areas as much as possible so that memory allocation can be optimized for each of two different situations (i.e., fixed allocation block size versus variable size). However, the storage ranges of compressed and expanded tiles are allowed to mingle so as to maximize the flexibility of the cache usage.

FIG. **8** is a state diagram illustrating the flow of image data or tiles between different storage states **250**. A tile can contain data in one or more of five states or forms as illustrated by ovals in FIG. **8**. The possible forms are: uncompressed and resident in cache memory (state **252**); compressed and resident in cache memory (state **256**); uncompressed and resident on disk (state **268**); compressed and resident on disk (state **262**); "not loaded" but re-creatable using information from higher-resolution image tiles (state **272**).

For most image access operations, the image data must be uncompressed and resident in cache memory **252**. However, that form consumes the most cache memory of any of the five forms. Therefore, a primary function of the tile manager **192** is to transform image tile data between state **252** and the other states which consume less (in the case of state **256**) or no cache memory whatsoever (in the cases of states **268**, **262** and **272**).

The eight transformation operations, shown in square boxes in FIG. **8**, constitute the main computational operations associated with managing image memory. The operation "load compressed tile image data from disk into cache memory" **264** is typically the first operation performed on a tile because most pre-scanned images are stored in compressed form in disk files. (A discussion of this "virtual loading" is provided hereinbelow.) The load operation **264** is performed by the Load CompFromDisk function which simply copies data from the disk into cache memory. The disk location and number of bytes to read is stored in the tile header fields **368** and **376** shown in FIG. **10**.

The function LoadCompFromDisk is normally used by the function LockCompHandle when the tile manager **192** needs to access the compressed form of data associated with a tile. LockCompHandle is analogous to LockExpHandle, described in FIG. **28**. The LockCompHandle function is also included in source code form in the Microfiche Appendix, in the file tilealloc.c.

Compressed data in cache **256** can be written back to the disk by the operation **260**. This is the reverse of the LoadCompFromDisk function. The present embodiment is capable of writing to disk in a wide variety of file formats. One skilled in the art can easily create a function to perform this task.

Compressed data in cache can be uncompressed (also termed "expanded") into another region of cache memory by the expand operation **258**. The expand operation **258** is controlled by the "Expand Tile" function **440** which is described with respect to FIG. **33**. The method of image compression varies according to image type (e.g. binary, 8-bit color, 24-bit color). Commonly used compression techniques include CCITT T.6 for binary images and CCITT SGVIII (draft standard) for color and gray-scale images. The ExpandTile function **440** selects the appropriate compression algorithm by referring to field **306** of the Document Information Structure shown in FIG. **9**.

Uncompressed data in cache **252** can be compressed and written to a separate region of cache memory by the compress operation **254**. The compress operation **254** is controlled by the CompressTile function **450** described with respect to FIG. **38**. Like ExpandTile, the CompressTile function **450** uses an image compression algorithm appropriate to the image type.

Uncompressed data on disk **268** can also be read directly into cache memory by the load operation **270**. The load operation **270** is performed by the LoadExpFromDisk function, which appears in source code form in the Microfiche Appendix, in file diskcach.c. The LoadExpFromDisk function is analogous to LoadCompFromDisk. The LoadExpFromDisk function refers to the fields **362** and **374** of the tile header **350** shown in FIG. **10**, for the location and number of bytes of the expanded file data on the disk.

Uncompressed data in cache **252** can be written back to the disk by the save to disk operation **266**. This operation is analogous to the save to disk operation **260** which operates on compressed data. The present embodiment can write compressed or uncompressed tile data to disk in a variety of formats. One skilled in the art can easily implement an equivalent function.

Image data for tiles in the "not loaded" state **272** must be constructed by resampling higher-resolution tiles. (During normal operation, only lower-resolution tiles can exist in this state—the full resolution subimage tiles are always "loaded".) The present embodiment provides two operations from the "not loaded" state **272** to the "loaded" state **252**, **256**. Uncompressed higher-resolution tile data is resampled to create uncompressed data in cache **252** by the resample operation **274**. Similarly, in the resample operation **276**, compressed data in cache **256** can be created from compressed higher-resolution tile data.

In both resampling operations, extensive advantage is taken of the fact that the resolutions of adjacent subimages in the subimage stack are related by a power of **2**. This greatly simplifies and speeds the resampling operation. Basic resampling techniques are well-known (See, for example, A. Rosenfeld and A. C. Kab, *Digital Picture Processing*, Academic Press, 1976). The resampling operation **274** and **276** are controlled by the function LoadSubImTile **436** described with respect to FIG. **31**.

In summary, FIG. **8** shows that a great part of the tile manager's utility derives from its ability to coordinate a variety of forms of image data in the course of complex image processing operations.

Generally, the way data starts out on the disk **162** is by loading a tiled image file into an application **190** via the tile manager **192**. An image file, like a Tagged Image File Format (TIFF) or CALS tiled image file, for example, can be loaded instantaneously, in a virtual sense. In the tiled formats, there are tiled image data that is stored in the image file and at the beginning of the file there is a directory with entries that locate the tiles (for example, the disk file version of tile **0** in subimage **0**, (0,0), is located at one address in the file and the disk file version of tile **1**, subimage **0** (0,1) is located at another address in the file). When an image file is loaded, the tile manager **192** gets the tile offsets and stores them in the tile directory and does nothing else. Hence, the image file is basically loaded without copying any data from the disk **162** into the image data cache

194, and a directory is created that maps the tiles in the virtual image memory space onto the disk 162.

FIG. 9A illustrates a document information structure 300. Each image, or document, in the system is associated with (and described by) a document information structure (called "docinfo", defined in FIG. 9). The docinfo structure contains information about the image as a whole, such as color and pixel organization, etc. It also contains a list of subimages contained in the image. Each subimage entry in the docinfo structure contains information about that subimage, such as width and height, etc. The intention is to make this data visible only to cache management functions and low-level access functions. The overall docinfo data structure 300 contains the following information:

302 Self-reference to document handle. Handle value assigned to this document by the host procedure which created the document. This value is unique over the entire system.

304 "Overviews Invalid" flag. This flag is true if the document is in the middle of a write operation.

306 Cache image compression algorithm. Compression algorithm used by the memory manager for this image.

308 Image color type. How the image is displayed.

310 Bits per image pixel. Number of bits per image pixel.

312 Tile size information. Size of expanded tile in pixels. The tiles are assumed to be square.

314 Number of subimages in doc. Number of subimages maintained in this document. The minimum value is one (the full resolution subimage).

316 Input file info. Input raster file information.

318 Output file info. Output raster file information.

320 List of subimage headers. Array of pointers to subimage header structures 321. The first entry in the array is always the full resolution image. Each position thereafter corresponds to a 2× resolution reduction from the previous subimage.

The subimage header structure 321 is illustrated in FIG. 9B. Each subimage has its own entry with each field as follows:

312 Pointer to tile headers.

314 Pointer to tile directory. Pointer to array of pointers to tile header records. This two-dimensional table provides an easy way to access individual tile headers on a (row,col) basis.

326 Subimage width and height. The width (x extent) and height (y extent) of the document measured in pixels.

328 Number of tile rows & cols in subimage. Number of tile rows in the image and the number of tile columns (i.e., the number of tiles needed to span the height and width of the image).

330 Image stack index of this subimage. This is the position of the subimage in the docinfo structure subimage list. It can also be used to determine the factor by which the subimage resolution is reduced relative to the full resolution subimage.

332 Pixel resolution of this subimage. Scan resolution in pixels per millimeter.

FIG. 10 illustrates the tile header 350. The tile manager's analog to the conventional address translation table is the tile directory. The tile directory is a two-dimensional array of entries corresponding to the two-dimensional array of tiles that form the image. Each full and reduced resolution image has its own tile directory. The tile directory record contains a list of pointers to lists of individual tile headers. The list in the tile directory record has one entry for each row of tiles. Each of those entries points to a tile header record list with as many elements as tile columns. Thus, there is one tile directory record per subimage and one tile header record per tile. The tile header record defines the current state of the tile and contains information used by the cache management functions. The tile header contains the following information:

352 Pointer to document containing this tile. Pointer to the document to which this tile belongs.

354 Index of subimage containing this tile. Index of the subimage (i.e., image stack layer) that contains this tile.

356 Row and column indices of tile. Tile row and column position of this tile within the subimage.

358 Status information. Defines the current state of the tile. This includes lock counts for expanded and compressed tiles.

360 Preserve count. Value greater than zero means the tile is desired for future operation, so the tile should be preserved in cache if possible.

362 Location of uncompressed image data in cache memory. Location of uncompressed (expanded) image data for this tile (if it exists). Status flag "ExpCached" will be true to indicate that the data is currently in expanded tile cache memory.

364 Location of compressed image data in cache memory. Location of compressed image data for this tile (if it exists). Status flag "CompCached" will be true to indicate that the data is currently in compressed tile cache memory.

366 Location of uncompressed image data on disk. Location of uncompressed (expanded) image data for this tile (if it exists). Status flag "ExpOnDisk" will be true to indicate that the data is currently on disk.

368 Location of compressed image data on disk. Location of compressed image data for this tile (if it exists). Status flag "CompOnDisk" will be true to indicate that the data is currently on disk.

370 Link to next less recently used tile. Pointer to next older (less recently used) tile, not necessarily a tile in this image.

372 Link to next more recently used tile. Pointer to next newer (more recently used) tile, not necessarily a tile in this image.

374 Number of bytes of expanded data in tile.

376 Number of bytes of compressed data in tile.

FIG. 11 illustrates a calling hierarchy 400 for the constituent functions. Further discussions relating to flow diagrams, herein, will include names which correspond to source code modules written in the "C" programming language. The object code is presently generated from the source code using a "C" compiler licensed by Sun Microsystems, Inc. However, one skilled in the technology will recognize that the steps of the accompanying flow diagrams can be implemented by using a number of different compilers and/or programming languages.

The top level in the program hierarchy is Main 402. Main initiates the functions calls to the lower level functions. Main embodies the top level control flow of the present invention.

The first function called by Main is Initialize Cache Manager 404 (InitCacheManager). InitCacheManager allocates the RAM and disk swap space needed for a

particular raster image. It must be called before attempting to load any image tiles into memory.

The next function Main may call is Load Tiled Raster Image **408** (LoadTIFF). LoadTIFF manages the loading of tiled images. This is the process where an existing image file on disk is mapped into memory.

Main will then call the function Begin Undoable Raster Operation **410** (BeginUndoableRasOp). BeginUndoableRasOp marks the beginning of a distinct, "undoable" raster image operation. This function does not save any region of image memory but only creates a new entry on the undo stack. The current version of the tiles in the affected region are saved by InitImageAccess.

The following function called by Main is Create Image Access Context **412** (InitImageAccess). InitImageAccess prepares the tile cache manager for upcoming accesses to a particular region of the specified image. This function creates a data structure called an "access context" (defined in FIG. **16**) that is used by the sequential access functions.

Main optionally calls the function Read Rows From Region **414** (ReadRowToRow) next according to the operation performed by the user. ReadRowToRow causes one input/output buffer row or strip to be read and transformed from tiled image memory as specified in the associated InitImageAccess call and the resulting access context.

The next optional function called by Main is Write Rows To Region **416** (WriteRowToRow), again according to the operation performed by the user. WriteRowToRow causes one input/output buffer row or strip to be transformed and written to tiled image memory as specified in the associated InitImageAccess call and the resulting access context.

It should be understood that other access functions, such as random pixel accesses, may optionally be called by Main.

Main then calls the function Close Image Access Context **418** (EndImageAccess). EndImageAccess terminates and discards an image access context. The memory allocated for the access context structure is freed. The tile manager is informed that the specified region of image memory is no longer needed by this operator.

The next function, Undo Previous Raster Operations **420** (UndoPreviousRasOp), is optionally called by Main. UndoPreviousRasOp restores the specified region to its original state using information from the undo stack.

The last function Main calls is Quit Cache Manager **422** (EndCacheManager). EndCacheManager frees the RAM and disk swap space. This function basically reverses what InitCacheManager does.

The second level of functions on the calling hierarchy **400** is shown starting with Load TIFF Subimage Tile Information into Tile Headers **424** (LoadTiffTilesStd) which is called by function LoadTIFF **408**. LoadTiffTilesStd manages the loading of TIFF images with strip structure.

The LoadTiffTilesStd function **424** calls a function Store Tile Information in Tile Headers **425** (LoadSubImDiskCache). LoadSubImDiskCache loads the tile directory of the specified subimage with information about the location, size and format of individual image tiles contained in a disk-resident tiled image file. It is the low-level interface for the "indirect file load" capability. The tile headers are assumed to be completely zeroed when this function is called.

The InitImageAccess function **412** calls a function Save Region For Undo **426** (SaveRegionForUndo). SaveRegionForUndo saves the specified region on the undo stack. It is called from within InitImageAccess if the SaveForUndo flag is true. It can also be used for low level operations that do not go through InitImageAccess. SaveRegionForUndo can then be called multiple times for different documents and different regions within a document so that arbitrarily complex editing operations can be easily undone.

The ReadRowToRow function **414** calls a function Lock Expanded Image Tile Group **428** (ExpTileLock). ExpTileLock "locks" memory handles referring to expanded image tiles. (The notion of locking and unlocking memory blocks is further discussed below with reference to FIG. **14**.) It also updates the associated tile header structure as appropriate for the operating system.

The ReadRowToRow function **414** also calls a function Unlock Expanded Image Tile Group **430** (ExpTileUnlock). ExpTileUnlock unlocks memory handles referring to expanded image tiles. It also updates the associated tile header structure as appropriate for the operating system.

The function ExpTileUnlock **430** calls a function Unlock Expanded Tile **432** (UnlockExpHandle). UnlockExpHandle unlocks an individual expanded tile handle. The lock count is decremented as appropriate. The tile is not actually swapped out of cache at this point but it becomes a candidate for swapping.

The function ExpTileLock **428** calls a function Lock Expanded Tile **434** (LockExpHandle). LockExpHandle locks an individual expanded tile handle. The lock count is incremented and the status flags are set as appropriate.

The LockExpHandle function calls a function Create Tile From Higher-Resolution Tiles **436** (LoadSubImTile). LoadSubImTile creates a valid expanded version of the specified tile by scaling down from the next higher-resolution subimage. This function is called recursively as necessary to get to a higher-resolution subimage where there is valid data. (Note: the tiles in the full-resolution subimage are always valid and loaded although not necessarily present in the cache memory.)

The function LockExpHandle **434** next calls a function Allocate Space for Uncompressed Version of Tile **438** (AllocExpHandle). AllocExpHandle allocates space in cache memory for a single expanded tile.

The function LockExpHandle **434** also calls a function Create Uncompressed Version of Tile From Compressed Version **440** (ExpandTile). ExpandTile uses a tile that exists in compressed form but not expanded form, allocates space for an expanded tile and decompresses the image data into that space.

The function LoadSubImTile **436** calls a function Create Compressed Lower-Resolution Tile From Compressed Higher-Resolution Tiles **442** (CompCopyToOview). CompCopyToOview creates a valid compressed version of the specified tile by scaling down from compressed or expanded version of the given higher-resolution subimage tiles. The function LoadSubImTile **436** also calls a function Copy Uncompressed High-Resolution Tiles to Uncompressed Low-Resolution Tile **444** (CopyTileToOview). CopyTileToOview updates the region of the next low-

er-resolution overview corresponding to the specified tile.

The Function CompCopyToOview **442** calls a function Collect Freeable Cache Memory **446** (CollectFreeCache). CollectFreeCache collects freed memory states or enlarges the cache file and adds the new memory capacity to the reserve list. This function is called when the cache manager usage exceeds preset limits. Therefore it makes sense to take time to free up as much memory as is convenient at this opportunity.

The function CollectFreeCache calls a function Free Uncompressed Version of Tile **448** (FreeExpHandle). FreeExpHandle frees space used for storage of expanded image tiles.

The function CollectFreeCache **446** also calls a function Create Compressed Version of Tile From Uncompressed Version **450** (CompressTile). CompressTile uses a tile that exists in expanded form but not compressed form, allocates space for a compressed tile and compresses the image data into that space.

FIG. **12** is the top-level control flow for the tile manager **192** (also called "Main"). The tile manager **192** can be executed on a number of operating systems or without an operating system. However, the workstation **150** (FIG. **6**) preferably includes the Unix compatible operating system **188**. Another preferred operating system is Microsoft MS-DOS running with or without Microsoft Windows 3.0.

Moving from a start state **470** to an initialization state **404**, the tile manager **192** performs an initialization of the image data cache **194** to determine the available memory space, or the amount of physical RAM and disk space available for a cache "file". At this point, the cache appears to the tile manager **192** as one contiguous range of physical addresses in memory. If the tile cache has already been initialized, this step is skipped. The possibility of multiple image access contexts (discussed below) allows multiple simultaneous requests.

The tile manager **192** has another parameter which is called the fast memory portion of the image data cache **194**. This parameter is particularly relevant when working on top of another virtual operating system such as Unix. The fast memory limit specifies approximately how much of the image cache file is actually kept in RAM memory at any moment by the native operating system (e.g., Unix). The balance of data (the less recently used portion) is likely to have been swapped out to the disk. The tile manager attempts to limit the amount of cache space used to store expanded tiles to less than the fast memory limit, but the limit can be exceeded if necessary with some degradation in performance. However, the total cache size limit is never exceeded. In operating systems without virtual memory capabilities built in (e.g., MS-DOS), the fast memory limit is the same as the total cache size limit.

Then the tile manager **192** moves to a function **472** wherein the tile manager **192** loads a tiled raster image file. The function **472** (comprising the function **408**, for example) loads any type of image file, and preferably a tiled image, into the memory address space configured by the tile manager **192**. If the image to be modified is already loaded, this step is skipped. Then the tile manager **192** moves to a function **410** where the tile manager **192** marks the beginning of an undoable raster operation if the tile manager **192** is writing to the image. The function **410** is an optional state and it is only used if the user wants to be able to undo the operation that modifies the image.

Any time that a region of the image needs to be accessed (for reading or writing) an image access context is created. This image access context is used to define the region for use by the tile manager. The creation is performed automatically by the file manager without effort by the user. For example, an image access context is created when the user draws a line in a region of the image.

Referring back to FIG. **12**, the tile manager **192** transitions to a function **412** to create the image access context. The image access context contains all of the state information about the access operation. It is possible to have multiple access contexts opened simultaneously with each access having stored state information contained in the access context. Thus, the tile manager **192** is re-entered and re-used by interleaved operations without confusion due to the unique access contexts of each image operation.

The tile manager **192** proceeds to a loop state **474** wherein the tile manager **192** begins a FOR-loop for all of the rows or columns in the region. The FOR-loop is executed multiple times if the operation specified by the user is a row or column strip oriented access. Strips are composed of one or more rows or one or more columns of data. For each of the strips, the tile manager **192** reads or writes the rows or columns of data in the strip in a function **476**. The function **476** actually comprises a set of functions including ReadRowtoRow **414** (FIG. **11**) and WriteRowtoRow **416**.

When the tile manager **192** has processed all the row and columns in the region, the tile manager **192** moves to a function (EndImageAccess) **418** where the tile manager **192** closes the image access context which frees all of the temporary buffers that were allocated for the image access context.

The tile manager **192** transitions to an undo previous raster operation function (UndoPreviousRasOp) **420**. This causes a modified image to revert to its previous state. The image tiles that had been modified are replaced by their original versions. This again is an optional step that the user initiates, if a mistake is made.

If the raster image is required for future operations, the tile manager moves to state **422**. Otherwise, moving to a state **478**, the tile manager **192** unloads the raster image. Unloading the raster image simply frees the memory that had been associated with that particular raster image. This is not a save raster image operation which would be slightly more complicated, but a save operation could be executed here. Of course, the image processing application **190** supports loading and saving raster images.

If more operations will be performed the tile manager moves to state **480**. Otherwise, from state **478**, tile manager **192** moves to a quit cache manager function (EndCacheManager) **422**. Herein, the tile manager **192** frees the image data cache **194** (FIG. **6**). Presumably, all of the images have been unloaded as in the state **478** so that this operation frees the image data cache memory and prepares the system for shut down. Lastly, the tile manager **192** terminates at an end state **480**.

FIG. **13** illustrates the initializing of the cache manager function **404**. The function **404** is entered by the task manager **192** at a state **488**. Then, moving to a state **490**, the task manager **192** initializes the cache usage variables. Of course, in the beginning, all of the cache space is available for use, in what is called the free-memory reserve list. That is, no cache memory is being used for expanded or compressed image data.

At state **492**, the task manager **192** allocates tile cache memory by requesting a portion of the address space from the memory space owned by the operating system. In a virtual memory system such as Unix, the request is handled by memory mapping a large file. The operating system does not allocate any memory, but it reserves an address space. Moving to a state **494**, the task manager **192** allocates a common blank tile. When dealing with binary images, space is reserved for one blank tile, which is kept around at all times for common usage by any number of operations, or access contexts.

At state **496** a compression buffer is allocated to be used as a scratch buffer when compressing data since, in general, the size of the resulting compressed data is unknown before a tile of image data is compressed. Hence, compressed data blocks will be variable sized. The tile manager **192** then exits the InitCacheManager function **404** at an end state **498**.

FIG. **14** illustrates a general memory state diagram with reference to a block of memory being "locked" or "unlocked". In the diagram, ovals are states and rectangular blocks are operations.

The state diagram is entered at a start state **502** by a new memory block. There are three basics states. "FREE" is a state **504** where there is no memory allocated. Actually, a block of memory is considered free if it is in one of the memory free lists, i.e., the "reserve free list", the "compressed free list" or the "expanded tile free list". It should be understood that the free list for the compressed tiles are actually composed of many lists based on the varying sizes of memory blocks.

Within a tile header (FIG. **10**) the tile manager **192** controls a memory handle which is a structure that has a pointer to (or location of) image data in the cache and a lock count (not shown) for both compressed and expanded versions of a tile.

A memory block transitions from the free state to unlocked, but allocated is through a state **506** for allocating the memory handle, which moves the block out of the free list and into use by a tile. As opposed to free, unlocked means that the memory block contains valid data and that it is associated with a tile but not currently being accessed. That is, the block is not being read or written at the time.

Now, the tile is unlocked at a state **508** but it contains valid data. Therefore, the next step is to lock the block, or lock the memory handle at a state **512** and then it becomes a locked memory state at a state **514**. That means it contains valid data and it is currently in use. The block can be locked more than once, each time just incrementing the lock count.

The lock count may be incremented multiple times, for example, when two access contexts (operations) are accessing the same region of memory. Hence, both contexts lock the block of memory or tile by incrementing the lock count. When the first access context is done it decrements the lock count. But the tile manager **192** knows that that tile is still in use by an access because the locked count is still non-zero.

The inverse operation is to unlock the handle at a state **516** and as long as the lock count is not decremented to zero at state **518**, it stays locked. Once the lock count is decremented to zero, it becomes unlocked again at the state **508**.

An unlocked tile is fair game for the tile manager **192** when the memory manager needs to find some space to lock a new tile. Therefore, when the tile manager **192** is looking for space, unlocked memory blocks may be freed and returned to the free memory lists.

The way to go from the unlocked state **508** to the free state **504** is by freeing the handle in which case the memory block is moved onto the free memory list.

Referring now to FIG. **15**, the flow diagram for the InitImageAccess function **412** shows the operation where the tile manager **192** creates the image access context starting at a state **530**. At a state **532** the input parameters are validated. If there is an error with the input parameters, the function ends immediately at an end state **534**.

Input parameters include a document handle indicating which image that the user wants to read or write from. Thus, the document handle must be validated. Another parameter is whether the user wants to read or write to the image. A transformation matrix, also input, basically directs how to scale, rotate, shear, etc., the image data.

If the input parameters are valid, the tile manager **192** locks the document handle at a state **536**. The document handle locks and unlocks just like other structures and resources in the tile manager and it prevents one user of a particular document or image from modifying or deleting that image while another operation or another access context is still using that document.

Then, at a state **538**, the tile manager **192** tests whether a non-orthogonal rotation has been specified. For example, a rotation of 30° causes the tile manager **192** go into a special operation that initializes the access with rotation. That also creates an access context but after a more involved process. Then the tile manager **192** ends the function **412** at a state **534** with a valid access context for rotations.

If an orthogonal rotation is specified then the tile manager **192**, allocates a conventional access context at a state **542**. Then the tile manager **192** continues to a decision state **544** wherein the subimage selection criterion is specified. For instance, the user may request the "low resolution" option which selects the lowest resolution subimage in the document's image stack. (In the context of an image editor, this may be the best solution during zooming or panning.) The user may also specify "most available"—i.e., whatever subimage has tiles currently in cache memory, regardless of the resolution. In either case, the tile manager **192** proceeds to a state **546** to select the reduced resolution subimage that is appropriate to that particular choice, i.e., either the one that has the resolution just greater than what was requested or a subimage whose tiles covering the access region are currently in cache. Now, at a state **548**, the tile manager **192** adjusts the transformation matrix so as to now refer to the reduced resolution subimage rather than the full resolution subimage by adjusting scale factors.

Alternatively, if the state **544** determines that the full resolution subimage is selected then the transformation matrix is unchanged. Proceeding to a state **552**, the pixel and tile limits of the affected image region are calculated. Knowing these limits, in a state **554**, the tile manager **192** creates a temporary directory for the tiles in that region. This directory is a two-dimensional array that references the tiles that contains the affected pixels. Later on the tile manager **192** refers to the region tile directory because it is specific to tiles that are inside the affected region.

The tile manager **192** then initializes the image scaling functions in a state **556**. Such scaling functions presently

used are the subject of applicant's concurrent application entitled "Process for High Speed Rescaling of Binary Images" (U.S. Ser. No. 08/014,085, filed Feb. 4, 1993, which is a continuation of Ser. No. 07/949,761 filed Sep. 23,1992, now abandoned, which is a continuation of Ser. No. 07/693,010 filed Apr. 30, 1991 now abandoned.

Moving on, the tile manager **192** tests whether polygonal clipping is required at a state **558**. For example, a request may be made to only read from within a specific polygonal region. If that is the case, the tile manager **192** initializes the polygonal region clipping functions in the tile manager **192** by passing in the boundary lists. The polygonal clipping function translates the boundary lists into edge lists that are used to very efficiently read out the rows or columns of data.

For example, suppose a "flood" request is made to turn all of the pixels black within an octagonal region. One way to accomplish the operation is to specify the points of the corners of the octagon in image coordinates and pass that in with the initialization of access context request, which would pass those vertices of the polygon into the polygonal clipping function set up function.

Then the tile manager **192** comes to a state **562**, where the tile manager **192** allocates buffers for scaling, if necessary. This is the situation where intermediate copies of the rows or columns of data may need to be kept during the process of scaling. Then the tile manager **192** tests whether the user specified that the region needed to be saved for undoing, at a decision state **564**.

An important feature of the present invention is an "undo" operation that is integrated with the image memory management so that only compressed tiles need to be saved after an undoable edit operation. In this way, a user can easily and quickly retract an edit operation that is no longer desired. For example, in mapping applications, e.g., USGS Quadrangle maps, the impression of a very large map is desired, but it is really composed of smaller map quadrants that were separately scanned, trimmed, adjusted and fit together. The smaller maps can be visually and logically joined into a single, large image. Using the present invention, a user can add a feature, such as a new sub-division, town, or road, that crosses a map boundary, specifying that the feature is undoable. Later, the user can remove the feature modification to the image by specifying the undo operation.

Now at a decision state **568**, the question is whether to update the subimages during the operation. If this is a write operation the tile manager **192** always writes into the full resolution subimage and the changes "trickle down" into the low resolution subimages. But the tile manager **192** has an option as to whether the lower-resolution tiles are updated during the modification operation or later when the tiles are requested for viewing operations. There are advantages in doing them both ways.

For example, if the affected region is small, it is more efficient to update the subimages while progressing through the operation. In this mode, when the tile is unlocked, the manager **192** immediately copies the data down into the next lower subimage tile but only one of the corners of the tile is affected. Thus, only portions of the low resolution subimage tiles need to be modified.

If, however, the subimages are not updated during the operation, then as soon as the image access context is created all of the subimage tiles that overlap the affected region are invalidated (they become "not loaded"). Hence, when the memory manager goes to access them again at some later time, it has to reconstruct them from the higher-resolution tiles. The advantage of that is that the memory requirement at any one moment is half of that of if the tile manager **192** was updating all of the tiles simultaneously. In this way, the tile manager **192** sets a flag at a state **570**.

In state **572** the tile manager **192** "preserves" the affected tiles in the affected subimages. Again, it relates to whether the tile manager **192** is updating subimages or not. If the tile manager **192** is reading, then it preserves only the tiles in the region of the subimage that will be accessed.

The ability to "preserve", or preferentially retain tiles that will be accessed in the course of the operation, is an important feature of the present invention that can yield significantly higher performance in certain situations where memory capacity limitations are encountered. When a tile is "preserved" for a particular access operation, it's preserve count **360** is incremented. The cache manager treats tiles with non-zero preserve counts differently from tiles with zero preserve count. The cache manager will discard unlocked unpreserved tiles before discarding older preserved tiles. (The cache manager normally discards older or less recently used tiles before discarding newer or more recently used tiles.)

Then, within the creation of the access context, the tile manager **192** actually locks down the first row or column of tiles in the region to establish the cache memory requirement for this operation, at a state **574**. If this succeeds, then the caller is assured that there will be sufficient cache space for the entire operation.

The tile manager **192** can perform row or column accesses. However, the following discussion only refers to a row access.

Then, at a decision state **576**, if the tile manager **192** cannot satisfy the request to lock down that first row of tiles, the function **412** terminates at the end state **578**. Otherwise, at state **580** the tile manager **192** initializes the row access functions.

Now, once the tile manager **192** has initialized the row access function in state **580** the tile manager **192** invalidates the affected subimage tiles if the tile manager **192** is writing to the full resolution subimage at a state **582**. Finally, in a state **584** the tile manager **192** returns the handle or a pointer to this access context to the user. From then on the user just uses this pointer to the access context and pointers to input and output buffers to get the next row or column of data.

FIG. **16** illustrates the access context structure **600**. The structure **600** operates on a high level to hide the low level operation from the user and contains bookkeeping information along with some memory management information. The access context **600** contains the following information:

602 Pointer to affected doc. Pointer to the document being accessed.

604 "Subimage Choice" option value. Specifies how to choose which of the subimages will be read from or written to.

606 Index of affected subimage. Index of the specific subimage directly affected by this access context.

608 Access quantum. Specifies "granularity" of image access.

610 Read/write option. Specifies what type of image memory accesses to prepare for (e.g., read or write).

612 Basic orthogonal rotation value. Specifies the image rotation in terms of how the bits in each buffer row are read from or written to the image (e.g., write buffer row to image column with increasing "y" coordinate).

614 Pixel combination operation. Specifies the pixel operation performed when combining the buffer contents and image contents. The results of the operation are stored in the output buffer when reading. The results go into image memory when writing.

616 Scaler type operation. Specifies the type of scaler preferred. In other embodiments, this may include fast low-accuracy scaling and line width-preserving scaling.

618 "Update overviews" flag. True flag indicates overview subimages should be updated in the course of this modification of the full resolution image. This causes the overviews to be correct when the access is complete.

620 I/O buffer width & height. Width (i.e., row length), total number of rows to process and pitch in pixels of the input/output bitmap.

622 I/O buffer pitch (bytes/row). Pitch of the input-/output buffer in bytes used for multi-row accesses. The input/output buffer is assumed to be a contiguous memory bitmap at least as large as the access quanta. It is always read or written in the natural order (by rows, low address to high). Flipping and rotation is always done on the image memory side.

624 I/O buffer bit offset to start of run. Indicates where the buffer's x=0 pixel lies within the first long word of the buffer's storage space. It must be between 0 and 31 inclusive. This parameter allows the caller to match up with arbitrary bit alignments.

626 Rows per strip (for AQ_STRIP access quantum). When operating in the AQ_STRIP mode, this specifies the maximum number of rows per input/output strip. Fewer rows may be written into the last strip if the end of the access region is hit before the strip is filled.

628 Number of I/O buffer rows yet to be processed. This variable is used in the access routines to keep track of the number of input/output rows remaining for the access operation.

630 Pointer to access function used in "SeqBufImageAccess". Pointer to the image access function that is tailored to the specific access mode requested.

632 Stepping directions for image row and column indices. The stepping increment each time the input/output buffer is advanced one row and one pixel. The allowed values are +1, 0, and −1.

634 Pointer to polygon clipping information. Refers to an edge table structure for controlling polygonal boundary clipping.

636 Pointer to raster scaling information. Tile level access information used by lower level modules in the course of the operation.

638 Pointer to uncompressed data in currently locked tiles. Pointer to an array of pointers directly into expanded tile image data. This list is used to accelerate sequential access into image memory. As each new tile row or column is encountered in a sequential access, this array is set to point directly into the affected tiles, which have been brought into cache memory and locked down. In other embodiments this could also be used to point to compressed tiles.

640 Pointer to region tile directory. Pointer to a 2-dimensional array of pointers to the tiles in the affected region of the subimage.

642 Next image row & column to be accessed. The index of the next image row and column to be accessed in sequential row and column operations.

644 Terminal row & column of access region. Stopping values for sequential row and column operations.

646 Unclipped extent of access region. Defines the image region that will be accessed over the course of the operation.

648 Clipped extent of access region. Defines the portion of the requested image region that actually falls within the boundaries of the image. Pixels outside of this rectangle are treated as background pixels.

650 Clipped image buffer bit offset and length. These values specify where, in the intermediate image row or column buffer, the first bit from the clipped image region is located and how many bits are to be read from or written to tiled image memory.

652 Number of tile rows & cols in access region. Number of tile columns and rows in the affected region.

654 Row & column of currently locked tiles. Column and/or row index of the currently locked tile or tiles.

656 Image row & col at origin of first tile in access region. Pixel coordinates of the upper-left pixel in the upper-left tile of the affected region.

658 Number of I/O buffer rows held over for next strip. Number of rows of output data that did not fit into the previous row and must be returned in the next and subsequent rows when expanding while reading image data.

660 Pointer to image tiling/untiling buffer. Points to a temporary buffer to hold data extracted from tiled memory prior to scaling when reading from image memory.

662 Number of bytes in tiling/untiling buffer. Size of buffer in bytes.

664 Bit offset for tiling/untiling buffer. Bit offset to the first valid pixel in tiling/untiling buffer.

666 Access transformation matrix. The transformation matrix mapping input/output buffer pixels onto the pixels of this subimage.

FIG. 17 illustrates the flow diagram for the "Save Region for Undo" function 426 as referenced in FIG. 15. The tile manager 192 starts at a state 680, moves to 682 where the tile manager 192 locks the document handle of the affected document that contains the region to save for undo. The tile manager 192 can save multiple regions from multiple documents sequentially and then undo them all in one operation later. Thus, the application programmer is allowed to easily undo multiple-region operations with a single undo call at a later point.

Moving to a state 684, the tile manager 192 clips the modified region to the image boundaries since there is no information to save outside of the image. Then the tile manager 192 moves to a decision state 686 wherein the tile manager 192 tests whether the affected region overlaps the image. If there is no overlap, that is to say, there is no image data to save, then the tile manager 192 moves to a state 688 where the tile manager 192 unlocks

the document handle and terminates the function **426** at an end state **690**.

If, however at state **686**, the modified region does overlap the image, the tile manager **192** moves to a state **692** wherein the tile manager **192** allocate memory for an "undo region header". The undo region header is similar to a document header, but reduced comparatively in the amount of data conveyed therein. The undo region header will be associated with tile header information, etc.

The tile manager **192** then moves to a state **694** where the tile manager **192** allocates memory for "undo region tile headers". These tile headers will be used to store copies of the original versions of the tiles in the affected region. The tile manager **192** then proceeds to a state **696** wherein the tile manager **192** makes an "undo tile directory".

Then the tile manager **192** moves to a loop state **698** where the tile manager **192** loops for each tile row in the region. The tile manager **192** then transitions to a loop state **700** wherein the tile manager **192** loops again for each tile column in the region (Thus, there is a two-dimensional loop.)

The tile manager **192** moves from the state **700** to a decision state **702** where the tile manager **192** checks to see if that particular tile in the document is loaded in the image cache memory. If the tile is not loaded, the tile manager **192** skips to the next tile in the region by returning to the loop state **700**. OtherWise, if the tile is loaded, the tile manager **192** marks the undo copy of the tile as loaded in a state **704**.

Note that there are two tiles. One is the original version of the tile that is still associated with the document and the second is the copy that the tile manager **192** is going to make and associate with the undo region header.

At a decision state **706**, a test determines whether the document tile is blank. If the tile is blank (i.e., all background color), then the tile manager **192** moves to a state **708** and simply marks the undo tile as "blank" and returns to the FOR-loop at **700**. If the document tile is not blank, then the tile manager **192** moves to a state **710** and the tile manager **192** marks the undo tile as "not blank" and moves to a state **712** Wherein the tile manager **192** tests whether the document tile has a valid copy of compressed data on the disk.

If a valid copy of compressed data does reside on disk, the tile manager **192** moves to a state **714** and simply copies the compressed tile disk location and size information from the document tile header to the undo tile header. Note that it is possible for a particular tile to have multiple representations of the same data. That is, a compressed version and an expanded version of the tile may exist in cache simultaneously. And a tile may have a compressed version in cache as well as on the disk. For undo, the strategy is to store the most compact version possible. The most compact version with regard to cache memory usage is to have a copy of the compressed tile on the disk.

If there is no compressed copy of the tile on the disk, the tile manager **192** proceeds to a decision state **716** wherein the tile manager **192** determines whether an uncompressed copy of the document tile resides on the disk. If the test succeeds, the tile manager **192** enters a state **718** and copies the uncompressed tile disk location and size information from the document tile to the undo tile and then returns to the inner FOR-loop at a loop state **700**.

If, at state **716**, there is no uncompressed tile information on the disk, the tile manager **192** continues execution to a state **720** in FIG. **17B** wherein the tile manager **192** locks the compressed version of the document tile. This locking of the compressed version of the document tile may cause an expanded version of the document tile to be compressed and a compressed version created. Therefore, there is a possibility of an error and that is checked at the decision state **722**.

If there is an error than the tile manager **192** unlocks the document handle at a state **724** and terminates with an error condition at the end state **726**. If there was no error in locking the compressed version of the tile then the tile manager **192** moves from the state **722** to a state **728** wherein the tile manager **192** allocates and locks down cache memory for a copy of the compressed data to be associated with the undo header. There is another error possibility at this point and the tile manager **192** checks for an error at a decision state **730**. If there is an error then the tile manager **192** returns to a state **724** and thereafter terminates the function **426**.

If there was no error in locking cache memory at the state **730**, the tile manager **192** moves to a state **732** and copies the compressed data from the document tile to the undo tile. The tile manager **192** actually copies the data that is stored within the tile—i.e., the compressed image data is copied from the document version to the undo version. Then the tile manager **192** moves to a state **734** and unlocks the compressed version of the document tile. Now, at a state **736**, the tile manager **192** unlocks the compressed version of the undo tile and the tile manager **192** returns to the inner FOR-loop at state **700** on FIG. **17A** where the tile manager **192** loops back to continue the loop for all of the tiles in the affected region.

When the tile manager **192** is done with all of the tiles in the affected region, the tile manager **192** moves to a state **738** where the tile manager **192** links the new undo header into the undo region list. Thus, multiple regions can be saved in the undo list and then in one operation, by calling undo previous raster operation, all of the operations that had been accumulated, can be undone. Then the tile manager **192** moves to a state **742** wherein the tile manager **192** unlocks the document handle and terminates the function **426** normally.

FIG. **18** shows the load tile to raster image function (LoadTiff). FIG. **18** is a flow diagram for the part of LoadTiff that loads tiled images only. In reference to FIG. **18**, the overall process may be understood whereby an existing file on the disk, i.e., an image file on disk, is mapped into memory. As described below, the overall process permits loading large images in a short time period relative to how long it would take to actually copy all of the image data into the computer's memory. In accordance with the present invention, the process shown in FIG. **18** is called the indirect loading capability. As shown in FIG. **18**, the tile manager **192** begins the LoadTIFF function **408** at a start state **750** and moves to a state **752** where the tile manager **192** opens the input file that is on the disk. If there is an error on the disk, the tile manager **192** prints an error message at a state **754** and terminates at an end state **756**. If no error exists, then the tile manager **192** moves to a state **758** and checks for the TIFF header structure that identifies that the input file is in fact a TIFF file. While the disclosure below discusses a TIFF file, it is to be understood that the process shown in FIG. **18** may be per-

formed on all types of tiled files, such as a MIL-R-28002A Type II file or an IBM IOCA tiled file.

Still referring to FIG. 18, if the tile manager 192 finds something other than TIFF header structure at state 758, the tile manager 192 moves to state 754 to indicate an error, and then exits at the end state 756. If the tile manager 192 finds a TIFF header structure while at state 758, the tile manager 192 move to a state 760, wherein the tile manager 192 counts the number of subimages in the TIFF file, one or more of which may exist in a TIFF file.

Next, the tile manager 192 moves to a state 762 and reads the full resolution subimage information which constitutes the basic information about the image, e.g., the image width and height, the size of the tiles, the compression format that is used, and the resolution. If the basic image information is not present and in proper form, the tile manager 192 moves to the state 754 to indicate an error. On the other hand, if no error is indicated at state 762, the tile manager 192 moves to state 764, wherein the tile manager 192 creates a skeleton document and locks that document. The skeleton document at this point contains no cache memory but only tile directory and tile headers that represent in a virtual sense the tiles that compose the image.

The tile manager 192 next moves to a state 766 where the TIFF full resolution subimage tile information is loaded into the tile headers for the full resolution subimage, as more fully disclosed below in reference to FIG. 19. Next, the tile manager 192 moves to a loop state 768 where there is a loop for each of the remaining lower resolution subimages. While in this loop, the tile manager 192 accesses a decision state 770, wherein the tile manager 192 determines whether

$$fr/lr=2^n \quad (1)$$

where

fr is the full resolution subimage resolution in pixels per inch; and

lr is the particular low resolution subimage resolution in pixels per inch.

If the ratio of fr to lr is a power of two, then a successful test is indicated, and the tile manager 192 moves to a function 424 and loads the TIFF subimage tile information into the tile headers for that particular subimage level. On the other hand, if the ratio of fr to lr is not a power of two, as indicated at the decision state 770, then the tile manager 192 ignores the particular subimage under test and returns to the state 768 until all of the subimages in the file are processed. When all subimages have been processed, the tile manager 192 moves to a state 772 and unlocks the document handle of the newly created document and terminates normally at an end state 756.

Now referring to FIG. 19, the function 424 whereby the tile manager 192 loads the TIFF subimage tile information into tile headers is shown. More particularly, the tile manager 192 begins at a start state 780 and moves to a state 782 wherein the tile manager 192 reads the number of tiles in the subimage. Then the tile manager 192 moves to a state 784 wherein the tile manager 192 allocates temporary buffers for the tile mode offset and byte count lists. These three lists have one entry each per tile in the subimage. If the tile manager 192 cannot properly allocate the temporary buffers, then the tile manager 192 exits with an error condition at an end state 786.

Upon successful allocation of the buffers, the tile manager 192 moves to a state 788 where the tile manager 192 reads the tile offset and byte count information from the disk file into the allocated buffers. In the TIFF file standard, all tiles are stored in the same mode (e.g., compressed). However, other tiled file formats (e.g., MIL-R-28002A Type II) specify the storage mode for each tile. The tile mode simply states whether a particular tile is stored in compressed form, in uncompressed form, or whether the tile is all foreground or background color. The tile manager 192 next moves to a state 790 where the tile manager 192 fills in the tile storage mode list. At state 790, the tile manager 192 synthesizes the tile mode information that the TIFF file does not contain itself. Then the tile manager 192 moves to the function 425 wherein the tile manager 192 stores the information in the subimage tile headers (FIG. 10), and terminates at an end state 786.

Now referring to FIG. 20, the function 425 whereby the tile manager 192 stores file information in tile headers is shown. The tile manager 192 begins this process at a start state 800 and moves to a state 802 where the tile manager 192 locks the document handle of the document for which the tile manager 192 is loading the subimage for. This function is performed once per subimage in the file and there may be multiple subimages in the file. Consequently, the locking of the document handle function can be performed several times in the process of loading a single document.

As shown in FIG. 20, in the event that an error occurs in locking the document handle the tile manager 192 terminates at an end state 804. On the other hand, if the tile manager 192 successfully locks the document handle at state 802, the tile manager 192 moves to a state 806 where the tile manager 192 determines whether the number of tiles in the file matches the number of tiles expected for the particular subimage in the particular file or document. If a mismatch exists between the actual and expected number of tiles, the tile manager 192 moves to a state 808 to print an error message and then terminates at the end state 804. On the other hand, in the event that the number of actual tiles matches the number of expected tiles, the tile manager 192 moves to a loop state 810 where the tile manager 192 enters the first part of a FOR-loop for each tile row. Still referring to FIG. 20, the tile manager 192 moves from state 810 to state 812 for each tile column. Accordingly, it will be understood that the tile manager 192 is processing a two-dimensional array at the states 810, 812.

In accordance with the present invention, the tile manager 192 processes, at states 810, 812, all of the tiles required to cover the particular subimage. Next, the tile manager 192 moves to a decision state 814 wherein the tile manager checks the value in the tile mode entry to determine whether the tile data is compressed. If the tile data is compressed, the tile manager 192 moves to a state 816 and stores the file offset and byte count in the compressed tile handle. The compressed tile handle is a part of the tile header structure, and the file offset is the location of the compressed data for the particular tile within the file as measured by a byte offset from the start of the file. The byte count represents the number of bytes of compressed data associated with the particular tile starting at the offset that is provided at the tile. From state 816, the tile manager moves to state 828, wherein the tile manager sets a flag to indicate that the particular tile is not blank.

In the event that the tile manager determines at state 814 that the tile data is not compressed, the tile manager

192 moves to a decision state 818 where the tile manager 192 checks to see if the data is uncompressed. If the data is uncompressed on the disk, the tile manager 192 stores the file offset byte count information in the uncompressed tile handle in state 820. From state 818, the tile manager moves to state 828, wherein the tile manager sets a flag to indicate that the particular tile is not blank.

If the tile manager 192 determines at state 818 that the tile data is not uncompressed, then the tile manager 192 moves to state 822, wherein the tile manager 192 checks to see whether the tile is all foreground at a state 822. For example, in a black and white drawing engineering document, foreground color is black, so the tile manager 192 treats a foreground as a black tile. If the tile is determined to be a foreground tile, the tile manager 192 proceeds to state 824, wherein the tile manager 192 creates an all foreground tile, and then sets the flag as not blank at state 828. As an example, if the image being processed is a color image, the tile manager 192 could fill the tile with the foreground color at the state 824.

On the other hand, if the tile is not all foreground, the tile manager proceeds to state 826 to determine whether the tile is all background. As discussed above, binary images usually have background pixels which are white or zero value. If a particular tile is blank, the tile manager 192 moves to a state 828 where the tile manager 192 sets the blank flag to indicate that the tile is indeed a blank tile. If at the state 826 the tile manager 192 determines that the tile is not all background, the tile manager 192 terminates with an error at an end state 830. In other words, having determined at state 822 that the particular tile was not all foreground, the only possibility left at state 826 is that the tile is all background. Consequently, a determination at state 826 that the tile is not all background indicates an error.

From state 828, the tile manager 192 moves to a state 832 and sets the loaded flag to true indicating that a valid image information set has been associated with the particular tile. The tile manager 192 completes the loop described above for each tile. After having processed each tile in the particular image, the tile manager 192 exits the two FOR-loops and moves to a state 834 where the tile manager 192 unlocks the document handle and then terminates normally at the end state 830.

Now referring to FIG. 21, the tile manager 192 performs a function which for purposes of the present invention will be termed "Undoable Raster Operation". The function shown in FIG. 21 is performed by the tile master 192 in the function "Begin Undoable Ras-Op", and is a relatively simple function, the purpose of which is to clear the undo region list. More particularly, in the process shown in FIG. 21, the tile manager 192 frees all of the undo regions associated with the previous operation to prepare for a new undo operation. Indeed, the present invention could be configured to have multiple level undo, i.e., the system of the present invention could undo two or three or more operations going into the past and also to be able to redo all of those operations at the user's choice. For example, the last three operations could be undone and then the oldest of those operations redone.

In specific reference to FIG. 21, the tile manager 192 begins at a start state 840 and then proceeds to loop state 842, in which the tile manager 192 executes a FOR-loop for each undo region in the current list. The tile manager 192 loops to a state 844 where the tile manager 192 frees all of the memory associated with that undo region. This may include freeing compressed data that is stored in cache or expanded data that is stored in cache and associated with the undo region. When the tile manager 192 finishes all of the regions, the tile manager 192 terminates at an end state 846.

Now referring to FIGS. 22A and 22B, there is shown the control flow for the ReadRowToRow function 414 which produces one or more rows of scaled image data each time it is performed. It is one of the basic image access functions. It should be understood that the tile manager 192 can also read columns of an image, etc., so as to produce a rotated output.

The tile manager 192 enters the function 414 by moving to a start state 850 and proceeds to a decision state 852 where the tile manager 192 checks for a region overrun. In other words, when the access context is created, the region that is going to be read in the course of the overall operation is specified, and in the event that the read row to row subfunction is accessed too many times, the region will be overrun. Any such overrun is detected by the tile manager 192 at state 852 and reported at state 854. In the event of an overrun, the tile manager 192 terminates at an end state 856.

If, on the other hand, no region overrun has occurred, the tile manager 192 moves to a decision state 858 where the tile manager 192 checks to see whether old results are carried over to the new strip. Such a carryover could occur when, for example, raster data is being enlarged by expanding one or more lines from the image. For example, when raster data is being enlarged by 4×, each line of input generates four (4) lines of output. Accordingly, three (3) output rows could be carried over for later strips. With this eventuality in mind, the tile manager 192 ascertains whether any data is being carried over and if so, the tile manager 192 uses the carried-over data before generating a new row. Consequently, if there is new data carried over, the tile manager 192 moves to a state 860 where new rows are generated from the carried over data.

Next, the tile manager 192 moves to a state 862 where the tile manager 192 checks to see if a particular strip is full. For purposes of the present invention, a strip is a collection of rows, i.e., a set of numbers arranged in rows As indicated at state 862, if the strip is full, then the tile manager 192 ends at the end state 856.

If the strip is not full and the tile manager 192 has used up all the carried over data, then the tile manager 192 moves to a decision state 864 where the tile manager 192 checks for ghosting, i.e., the skipping of some rows of data in order to produce a low quality image while panning or zooming. If ghosting is in effect, the tile manager 192 moves to state 866, wherein the tile manager 192 calculates the number of blank lines to create. The system then moves to a state 868 where the tile manager 192 writes the blank lines to the output strip buffer.

From state 864, if no ghosting was detected, or state 868, if ghosting is not in effect, the system moves to state 870 where the tile manager 192 again checks to see if the strip buffer is full. If it is, the tile manager 192 exits at the end state 856. If it is not, the tile manager 192 checks to see that there are still input rows to read in a decision state 872. If there aren't, the tile manager 192 has reached the end of the specified image region, and proceeds to state 874 to obtain another row of output data by flushing the scaler buffers. In accordance with the present invention, in the state 874 the tile manager 192 sets a flag that is subsequently passed down to the

scaler functions to flush intermediate results from the scaler functions. This is the case when for reducing data, i.e., if a plurality of rows is being combined into one output row. That is how the last output row is produced.

From state **874**, the system moves to state **894**, shown in FIG. **22B**. On the other hand, in the event that there are no unread image rows at state **872**, the system moves to decision state **876**, where the system determines whether the row is outside of the valid image boundaries. If yes, the system moves to a state **878**, where the tile manager **192** substitutes blank lines for the input. The tile manager proceeds from state **878** to a state **894**, shown in FIG. **22B**. If the answer to the decision at state **876** is no, the system moves to a decision state **880**, shown in FIG. **22B**, to check whether the row is contained in the currently locked tile row.

At state **880**, the tile manager **192** moves down the image, and the system sequentially passes through successive tile rows. Each tile contains, e.g., **512** rows, so when a particular tile row is locked it stays locked until all **512** image rows in that tile row have been read. Each time the system arrives at a new row it tests to see that the row is contained in the currently locked tile row. If it is not, the system moves to the state **430** (function ExpTileUnlock) to unlock the old tile row and lock down the new tile row (at state **428**). In addition, the tile manager **192** has to unpreserve the row of tiles that was just unlocked. Unpreserving them tells the memory manager that those tiles are no longer needed for this access operation and it can do what it wishes with them.

Next, the system proceeds to a decision state **882** to determine whether any tiles are blank. If they are, the tile manager **192** substitutes a reference to a "common blank tile" and that common blank tile is used, as indicated at state **884**. All tiles that are blank are mapped onto this common blank tile. Consequently, the tile manager **192** uses less image memory.

From state **884**, **882**, or **880**, as appropriate, tile manager **192** proceeds to a decision state **886** to check for polygonal clipping. If the tile manager **192** is doing polygonal clipping then each input row of data is clipped as appropriate for that polygon in states **888** and **890**. The loop allows multiple clipped regions within each row. If there is no clipping, then the tile manager **192** simply copies the entire input row from the image into the input row buffer in a state **892**. Then the tile manager **192** move to a state **894** where the tile manager **192** passes these input rows through the scaler if the tile manager **192** is scaling the data. Finally, the tile manager **192** takes the results of the scalers and copies that information to the output strip buffer if necessary at a state **896**. The tile manager **192** then returns to the state **870** (shown in FIG. **22A**) where the tile manager **192** continues the process of retrieving input rows and scaling them until the tile manager **192** has filled the output strip buffer. The system then moves to the termination condition at the end state **856**.

Now referring to FIG. **23A**, a process which will be referred to as "Write Rows to Region" will be described. The tile manager **192** starts at state **900** and moves to state **902** where the tile manager **192** tests for region overrun. Region overrun can occur when the calling function attempts to write more rows to the image than was specified when the access context was created. If the region was overrun, the tile manager **192** reports an error at state **904** and terminates with an error at state **906**. If there is no region overrun, the tile manager **192** moves to the FOR-loop in state **908** where the tile manager **192** loops for each input row in the input buffer, which is the buffer that is passed in by the calling function. It contains the data that is to be processed and written to the image. The loop is executed for each row and moves to state **910** where the input data is passed through the scaler functions and put into a temporary buffer. If the scaler does not always produce an output row, as is the case when reducing the resolution, a plurality of input rows may have to be combined to produce a single output row. So, at the state **912**, the tile manager **192** determines whether an output row was produced after the input row is scaled. If not, the tile manager **192** goes back to the loop at state **908** and continues the process as described. On the other hand, when the tile manager determines at state **912** that an output row was produced, the tile manager **192** moves to state **914** which is a FOR-loop for each copy of the scale row to write to the image. It may be the case that more than one copy of the scaled row needs to be written into image memory. This is the case when the tile manager **192** is expanding the input image data. It may be that one input row is replicated four times to get a 4× expansion factor.

Next, the tile manager **192** moves to state **916** where the tile manager **192** checks to see if the destination row index is outside of the image's clipping boundaries. If so, the tile manager **192** simply ignores it and moves back to state **914**. If it is within the clip boundaries the tile manager **192** moves to state **918** where the tile manager **192** determines whether the destination row is in the currently locked tile row. If it is not, the tile manager **192** moves to state **920** where the tile manager **192** unpreserves and unlocks the old tile row that is currently locked. The tile manager **192** then moves to state **922** to determine whether the update overview flag is true. This is an option that is specified in the lo access context and it determines how lower-resolution tiles are updated when the full resolution subimage is modified. If the update overview flag is true, then the tile manager **192** moves to state **924** where the tile manager **192** unpreserves the low resolution tiles that will no longer be needed.

After the system has unpreserved the low resolution tiles that are no longer needed at state **924**, the system moves to state **926** and locks down the new tile row. Only the full resolution tile row is locked at this level. The low resolution tiles are actually updated when the call to unlock the old tile row is made.

Next, the tile manager **192** moves to state **928** to determine whether an error was detected when the new tile row was locked. If so, the system terminates with an error condition at state **906**. If there is no error or if in state **918** the tile manager **192** finds that the destination row is currently in the locked tile row, the tile manager **192** moves to state **930** in FIG. **23B**. At state **930**, the tile manager **192** determines whether polygonal clipping is activated. If it is, the tile manager **192** computes the clip points for the current image row, as indicated at state **932**, which results in a list of clip point pairs.

The tile manager **192** then moves to state **934**, wherein the tile manager **192** conducts a FOR-loop for each of the clip point pairs that the tile manager **192** computed in state **930**. As shown in FIG. **23B**, the tile manager **192** loops to state **936** where the tile manager **192** copies pixels from a scaler output buffer to the image row between each pair of clip points. When that loop terminates, the tile manager **192** returns to state

914 in FIG. 22A. On the other hand, if the tile manager determines at state 930 that polygonal clipping is not active, the tile manager 192 moves to state 938, wherein the tile manager 192 copies the scaler output buffer pixels to the image row without clipping. The tile manager 192 then proceeds to state 914.

Now referring to FIG. 24, the tile manager starts at state 950 in the end access function shown in FIG. 24 and proceeds to state 952. At state 952, the system cleans up after row or column access functions by freeing buffers used by the row or column access functions.

Next, at state 954, the tile manager 192 unlocks the last row or column of tiles accessed. Then, the system moves to state 956 where the tile manager 192 unpreserves any tiles in the region that are still preserved. The system may perform the functions at states 954, 956 when an operation was aborted in mid-progress and it cleans up after those partially completed operations.

At state 958, the tile manager 192 cleans up after the polygonal clipping function. If there was polygonal clipping involved in this access context the tile manager 192 has to free the buffers that contain the polygon edge information.

Next, the system moves to state 960, where the tile manager 192 frees scaler buffers, the temporary tile directory, etc.. From state 960, the system moves to state 962, wherein the tile manager 192 unlocks the document handle to indicate to the memory manager that the access context no longer is referring to the particular document associated with the document handle.

The tile manager 192 next moves to state 964 where the memory that was used to store the data for the access context is freed. Then, the system ends the clean up function at state 966.

Referring now to FIGS. 25A,B, a function is shown which, for purposes of the present invention, will be termed the "Undo Previous Raster Operations". The tile manager 192 starts at state 970 and moves to state 972, wherein the tile manager determines whether any undo regions exist in the list or if the list is empty. If no regions exist then the tile manager 192 moves to end state 974 and terminates normally.

If the tile manager 192 determines at state 972 that "undo" regions do exist, the tile manager 192 moves to state 976, where the tile manager 192 enters a loop for each undo region in the list. In this loop, the tile manager 192 moves to state 978 where the tile manager 192 locks the affected document handle. The document handle that is locked is the one that was stored in the undo region header that tells where that particular undo region came from. The tile manager 192 moves from state 978 to state 980 where the tile manager 192 saves the current document region to support redo (i.e. an "undo" operation following by another "undo" operation). Then the tile manager 192 moves to state 982 to invalidate the affected tiles in the lower-resolution subimages. The strategy represented by states 980, 982 in FIG. 25A is to save the minimum amount of information that is needed to reconstruct the image, which means the tile manager 192 saves only the affected tiles in the full res subimage.

Next, the system moves to a loop indicated by the states 984, 986. In this loop, for each tile, the tile manager 192 moves to state 988, discarding the document tile image data. Then the tile manager 192 moves to state 990 to determine whether the undo tile is loaded. If it is not loaded, the tile manager 192 moves to state 992 where the tile manager 192 marks the document tile as "not loaded". If the tile is determined to be loaded at state 990, the tile manager 192 moves to state 994 to mark the document tile as "loaded". From state 994, the system moves to state 996 in FIG. 25B.

At state 996, shown in FIG. 25B, the tile manager 192 determines whether the undo tile is marked as blank. If it is, the tile manager 192 moves to state 998, wherein the tile manager marks the document tile as blank, and then the system loops back to state 986. If the undo tile is determined to be not blank at state 996, the tile manager 192 move to state 1000. At state 1000, the tile manager 192 checks to see if the undo tile points to compressed data on the disk. If it does, the tile manager 192 moves to state 1002 and copies the disk location and size information about the compressed data into the document tile header and loops back around. If there is no compressed data on the disk, then the tile manager 192 moves from state 1000 to state 1004, wherein the tile manager 192 determines whether uncompressed data exists on the disk associated with the undo tile.

If so,the tile manager 192 moves to state 1006, wherein the file manager 192 copies the disk location and size information about the uncompressed data into the document tile header and loops back to state 986. If the system determines at state 1004 that there is no uncompressed data on the disk, the tile manager 192 proceeds to state 1008, wherein the tile manager 192 determines whether the undo tile "points" to uncompressed data in cache memory. If it does, the tile manager 192 moves to state 1010, wherein the tile manager 192 copies the pointer to the uncompressed data from the undo header to the document tile header.

From state 1010, the system returns to state 986. If no uncompressed data exists in the cache, however, as determined in state 1008, the tile manager 192 stores a pointer to the compressed data in cache in the document tile header and returns to state 986.

Referring back to FIG. 25A, when the tile manager 192 has completed the loop described above, the system moves to state 1014, unlocking the document handle. From state 1014, the tile manager 192 proceeds to state 1016, wherein the tile manager 192 frees the memory associated with the undo header. The tile manager 192 then moves to state 976. Thus, the system returns to state 976 for each undo region in the list. As intended by the present invention, the tile manager 192 continues the loop for all of the regions in the list. The undo regions are restored in "last-in-first-out" order. At the completion of the looping process described above, the system moves to state 974.

Now referring to FIG. 26, when the tile manager 192 ends the cache management, the tile manager 192 starts the process shown in FIG. 26 at state 1020 and proceeds to state 1022 wherein the system frees the compression buffer. From state 1022, the system proceeds to state 1024, wherein the system frees the common blank tile. Next, the system moves to state 1026 to free the tile cache memory. The system then ends the process shown in FIG. 26 at state 1028.

FIG. 27 provides an explanation of the function exp tile lock. The tile manager 192 starts at state 1040 and moves to state 1042 where the tile manager 192 enters a FOR-loop for each tile row to be locked. In accordance with the present invention, the system in the exp tile lock function is capable of locking down all the tiles in a two dimensional region.

For each tile in the specified region, the system moves to state **1046**, wherein the tile manager **192** determines whether the particular tile is blank. To make this determination, the system examines flags in the tile header itself or checks the image data for that tile to determine if there are any non-background pixels. If it is not a blank tile, the tile manager **192** move to state **434** where the tile manager **192** locks the uncompressed version of the tile. Then the tile manager **192** proceeds to state **1050**, wherein the tile manager **192** determines whether an error had occurred in the process of creating the uncompressed version of the tile. If no error is found at state **1050**, the tile manager **192** continues to loop to the next tile in the region by returning to state **1044**. If an error did occur, as determined at state **1050**, the system proceeds to state **430** to unlock previously locked tiles, and then ends at state **1056**.

In the event that the tile manager **192** at state **1046** detected that the particular tile was a virtual blank tile, i.e., a tile that exists only by virtue of the fact that there is a tile directory entry for that tile, the tile manager **192** take no action, other than to loop back to state **1044** for further processing.

FIG. **28** illustrates the control flow for the "lock expanded tile" function **434** wherein the tile manager **192** takes a single tile and locks the expanded version of the tile in the image data cache **194**. The tile manager **192** enters the function **434** at a start state **1060**, and proceeds to a decision state **1062** wherein the tile manager **192** tests whether the tile is marked as "loaded". As already mentioned, a loaded tile is one that either contains or references valid image data, is either uncompressed or compressed image data, and it either resides in cache memory or on the disk. If the tile is not loaded, the tile manager **192** moves to a function **436** wherein the tile must be created from higher resolution tiles which are loaded. Afterwards, the tile manager **192** determines if there was an error in a decision state **1066**. If there was an error, the tile manager **192** terminates the function **434** at an end state **1068** and reports the error condition. Otherwise, if there was no error in creating the tile, the tile manager **192** continues, moving from the state **1066** to a decision state **1070**.

The tile to be locked is now loaded so the tile manager **192** tests whether the uncompressed version of the tile is in cache memory. The objective of the function **434** is to guarantee that there is an uncompressed version of the tile in cache memory. Now, if the uncompressed version is not in the cache, the tile manager **192** proceeds to a decision state **1072** to determine whether the selected tile is a blank tile.

If the tile is blank, the tile manager **192** proceeds to a state **438** to create a blank tile. Note here that the function ExpTileLock **428** (FIG. **27**) will detect a blank tile before calling the function **434** if it can take advantage of using a common blank tile at a higher level. In other words, if the tiles are locked for reading only, i.e., the image data will not be modified in any way, then all blank tiles can refer to the same section of blank memory. However, if the tiles are locked for writing, all tiles must have their own memory because different image data can be written to the different tiles.

At this point, state **438**, memory has presumably been allocated for a blank tile. Moving to a state **1074**, the tile manager **192** tests whether there was an error and moves to the end state **1068** if there was an error.

Returning in the discussion to the decision state **1072**, if the tile is not blank, then the tile manager **192** transitions to a decision state **1076** and tests whether there is a uncompressed version of that tile on the disk. If the uncompressed version is on disk, then the tile manager **192** reads that uncompressed version from the disk into cache memory at a state **1078**. Then the tile manager **192** moves to the state **1074** to test for errors.

If, at the state **1076**, there is not an uncompressed version on the disk, the tile manager **192** moves to the function **440** so as to create the tile from the compressed version. The compressed version can be either in cache memory or on the disk, and this is handled by the function **440**. Again, the tile manager **192** checks for an error at the state **1074**.

Now, assuming that there was no error found at the state **1074**, the result is that the tile manager **192** has an uncompressed version of the tile in cache. Therefore, the tile manager **192** proceeds to a decision state **1080** to verify that the uncompressed version is valid. It is sometimes the case that the uncompressed version of a tile is locked by one access context and then for come reason it is invalidated by another access context. This happens when the first access context is reading an uncompressed version of a tile from a lower resolution image, and another access context is actively modifying the full resolution subimage with a particular setting of parameters. If the tile not valid, the function **434** is terminated at the end state **1068**.

Alternatively, a valid tile that was determined at the state **1080** causes the tile manager **192** to increment the uncompressed data lock count for that tile at a state **1082**. The lock count starts out at zero for an unlocked tile and can increment as high as necessary. However, the lock count will be decremented once for each unlocking operation. It is important to match the number of times a tile is locked with the number of times the tile is unlocked. Otherwise, the tile would end up in a permanently allocated (unfreeable), locked state.

Proceeding to a decision state **1084**, the tile manager **192** tests whether the tile is locked for writing or for reading. If the tile manager **192** locked the tile for writing, the execution of the function **434** continues to a state **1086** wherein the "blank" status flag is invalidated. The blank status flag is actually a combination of two flags. One that says that the tile is blank or not blank and the second flag that says if the first flag is valid or not. The reason for two flags is that the way to detect that a tile is blank is by searching through all the pixels in that tile. To do so every time the file is accessed would be wasteful so occasionally, truly blank tiles won't be handled as blank tiles. Hence, there is a second flag that is set, in the state **1086**, when the first flag is invalid. The second flag indicates that the tile must later be examined to determine whether it is still blank.

The tile manager **192** next moves to a state **1088** to invalidate the disk-resident, uncompressed version of the tile, if one exists. This is because the tile manager **192** will modify the cache-resident version of the tile. To synchronize the cache-resident and disk-resident versions, the disk-resident version is invalidated. Then, at a state **1090**, the tile manager **192** invalidates and frees the compressed versions if they exist.

A compressed version of the tile may be in cache or on the disk and, at the state **1090**, the tile manager **192** cleans both out of memory. Thus, at the end of the "lock for writing" operation, the only valid version of the tile is the expanded version in cache, which at this point is locked. Then the tile manager **192** continues to a state **1092** to move the newly locked, expanded ver-

sion of the tile to the front of the "most recently used (MRU)" list of uncompressed tiles.

The MRU list is a doubly-linked list wherein, starting at the beginning, the tile is found that was most recently used, then the next most recently used, and so on, the last tile was used the longest time ago. That list is used by the cache manager to determine which tiles are least likely to be used again as a second level of criteria.

Finally, the tile manager **192** terminates the LockExpHandle at the end state **1068**.

FIG. **29** illustrates the control flow for the "unlocking expanded image tile group" function **430**. The function **430** is just the reverse of lock expanded image tile group. In other words, there is a region of locked tiles which must be unlocked because the access to the tiles is complete. Generally, the two functions, ExpTileLock and ExpTileUnlock are called for a row or column of image data rather than a region but an entire region lock/unlock is possible.

The tile manager **192** enters the function **430** at a start state **1110**. The loop states **1102** and **1104** represent the beginning of nested FOR-loops. That is, the outer loop, beginning at the state **1102**, unlocks a row of tiles, and the inner loop, beginning at the state **1104** unlocks a column of tiles. Moving from the state **1102**, to the state **1104**, and then to the function **432**, the tile manager **192** unlocks the uncompressed version of the tile. When all the tiles in the region are unlocked, the tile manager **192** terminates the function **430** at an end state **1108**.

Now referring to FIG. **30**, the tile manager **192** enters the UnlockExpHandle function **432**, referred to in FIG. **29**, at a start state **1110**. The tile manager **192** proceeds to a decision state **1112** to test whether the uncompressed version of the currently selected tile is in fact locked, i.e., whether the lock count is non-zero. If the tile is not locked, the tile manager **192** exits the function **432** at an end state **1114**.

If, at the state **1112**, the tile is found to be locked, the tile manager **192** moves to a state **1116** to decrement the lock count. Thereafter, the execution continues to a decision state **1118** wherein the tile manager **192** tests whether the "update overview" flag is set true. If the flag is set, the tile manager **192** moves to a state **1120** to update the corresponding lower-resolution tiles. In the process of modifying tiles, the tile manager **192** locks a tile down in the image data cache to write to it. When the tile is unlocked, that is a signal to the memory manager to update the lower resolution tiles that correspond to the higher resolution tile. Thus, the image data in the high resolution tile being unlocked is copied down into the lower resolution tiles, all the way down to the bottom of the image stack.

Once the lower resolution images are modified, or if the overviews are not being updated, the tile manager **192** proceeds to a decision state **1122** to test whether the lock count is exactly zero. If the lock count is not zero, the tile manager **192** terminates the function **432** at the end state **1114**.

Otherwise, the tile manager **192** moves to a state **1124** to clear the "cache collection delay" flag. The cache collection delay flag is set by the tile manager after unsuccessfully trying to reduce the expanded memory usage of the cache file. It is cleared in the function **432** because there is now the possibility of freeing the tile that was just unlocked. In other words, the tile can be removed from the cache to create some space. This flag prevents the tile manager or the cache manager from making repeated, unsuccessful attempts to create space.

After the tile manager **192** clears the flag, execution proceeds to a decision state **1126** to determine whether the uncompressed version of the tile is invalid. As explained hereinabove, it is possible for one access context to have the expanded version of the tile locked down and another access context to invalidate the data in that tile. The tile must remain in memory until the first access context unlocks the tile. Once it is unlocked and the lock count is decremented to zero, if the tile is invalid, the tile manager **192** moves to a state **1128** to free the uncompressed tile version, or remove the tile from the image data cache. In either case, the tile manager **192** terminates the function **432** at the end state **1114**.

FIG. **31** illustrates the control flow for the "create tile from higher-resolution tiles" function **436** referred to in FIG. **28**. The tile manager **192** begins the function **436** at a start state **1140** and proceeds to a decision state **1142** to determine whether the tile is in fact already loaded, in which case no further processing is needed and the tile manager **192** terminates the function **436** at an end state **1144**. Assuming that the tile is not loaded, the tile manager **192** moves to a decision state **1146** to test whether a higher resolution subimage exists.

This function is called only for lower resolution subimages where the tile manager **192** can create the lower-resolution tiles from higher-resolution tiles. Hence, higher-resolution subimages must exist for the function to succeed. If no higher-resolution subimages exist, the tile manager **192** reports the error and terminates the function **436** at the end state **1144**.

If the higher-resolution subimage does exist, the tile manager **192** proceeds to a state **1150** to calculate the indices of, or locate, the four higher-resolution tiles that reduce to this tile. There are four tiles involved because the preferred resolution step between subimage levels is two in the presently preferred embodiment. Thus, since there are two dimensions, four higher-resolution tiles are required to produce each next lower resolution tile.

Thereafter, the tile manager **192** enters a FOR-loop at a loop state **1152**. For each of the four higher-resolution tiles, the tile manager **192** tests whether the tile is loaded in the image data cache, at a decision state **1154**. If the tile is not loaded, then the tile manager **192** moves to a state **1156** wherein a recursive call is made to the "load subimage tile" function to create the corresponding higher-resolution tile from yet higher-resolution tiles. This case occurs if a the tile is a few layers down in the image stack and the tiles in all but the full resolution subimage had been invalidated. Therefore, the function **436** invokes itself to work all the way back up to the top level, recreate the higher-resolution tiles and then work back down to the tile of interest. Only higher-resolution tiles that map to the particular lower-resolution tile need be loaded

Assuming that all the higher-resolution tiles have been loaded, the FOR-loop terminates and the tile manager **192** proceeds to test whether all of the higher-resolution tiles are blank. If all four of the high resolution tiles mapped to this low resolution are blank, the tile manager **192** transitions to a state **1160** to mark the low resolution tile as blank. The tile manager **192** does not create any image data for the blank, lower-resolution tile. The tile manager **192** and terminates the function **436** at the end state **1144**.

If, however, one or more of the higher-resolution tiles is not blank, the tile manager **192** moves to a state **1162** to make a determination as to whether it is faster to create the lower-resolution tile by scaling the com-

pressed version of the higher-resolution tiles or the expanded version of the higher-resolution tiles. An algorithm is used at the state **1162** to decide which is faster and depends on the machine that the program is running on, and other considerations. If it is faster to scale the compressed data the tile manager **192** moves to the function **442** to create the compressed, lower-resolution tile directly from the compressed higher-resolution tiles.

Now, if it is determined that it is faster to scale the expanded version of the data, the tile manager **192** moves from the state **1162** to a state **1166** to allocate memory for the uncompressed version of the lower-resolution tile. From the state **1166**, the tile manager **192** moves to the beginning of a FOR-loop at a loop state **1168** wherein for each of the higher-resolution tiles the tile manager **192** scales the expanded version of the higher-resolution tile directly into the proper position in the lower-resolution tile using the function **444**. When the tile manager **192** has scaled each of the four high resolution tiles, the tile manager **192** has completed the creation of the expanded version of the low resolution tile.

The tile manager **192** then proceeds, from either of the states **1168** or **442** to a decision state **256** wherein the tile manager **192** determines if an error was incurred in that process. If there was an error, the tile manager **192** moves to a state **1172** to report the error. From either of the states **1170** (if no error) or **1172**, the tile manager terminates the function **436** at the end state **1144**.

FIG. **32** contains the flow diagram for the "allocate space for uncompressed version of tile" function **438** referred to in FIG. **28**. The tile manager **192** enters the function **438** at a start state **1180** and moves to a decision state **1182** to test whether the "soft" uncompressed cache usage limit is exceeded. The soft uncompressed cache limit is a number that is cast into the tile manager **192** during initialization and it basically sets a guideline for how much of the image data cache is to be devoted to uncompressed image data. If the cache manager gets a request for uncompressed cache space and finds that this soft limit has been exceeded, it attempts to reduce the amount of expanded image data that is held in cache either by compressing expanded tiles or by discarding expanded tiles that have valid compressed versions or some other way to recreate them.

If the tile manager **192** finds that the soft limit is exceeded, the tile manager **192** moves to a state **1184** to first check whether the "cache collection delay" flag is set. This flag is set after an unsuccessful attempt to reduce cache memory usage and prevents repeated unsuccessful calls to collect free cache at a state **1186**.

Thus, the tile manager **192** will not try to reduce the expanded memory usage until the flag is cleared in the "unlock expanded tile handle" function **432** (FIG. **30**).

If the cache collection delay flag is not set, the tile manager moves to a state **1186** to collect free cache memory by freeing uncompressed tiles. After that, the tile manager **192** moves to a decision state **1188** to test whether the soft uncompressed cache usage limit is still exceeded after an attempt to reduce the memory usage. If the usage is still exceeded, the tile manager **192** prints a warning message on the video display **154** (FIG. **6**) at a state **1190** and then sets the cache collection delay flag at a state **1192**.

Returning in the discussion to the state **1182**, if the soft limit was not exceeded, or if it was not exceeded at the state **1188**, the tile manager **192** moves to a decision state **1194** to determine whether there is memory available in the uncompressed tile free list. If there is not memory available in the uncompressed tile free list, then the tile manager **192** moves to a decision state **1196** to determine whether there is memory available in the cache reserve list. If there is no memory available there, the tile manager **192** moves to a state **329** wherein the tile manager **192** again tries to collect free cache space by unlocking or freeing both uncompressed and compressed tiles. At this point, the tile manager **192** must free space in order to allocate space for this uncompressed tile. The tile manager **192** moves to a state **1200** to determine whether memory is now available in the cache reserve list. In the state **1198**, when the cache memory space is freed, it is placed into the cache reserve list. If memory is not available, then the tile manager **192** moves to a state **1202** and prints a "cache overflow" error message and terminates the function **438** with an error condition at the end state **1204**.

Now, taking an alternate path from the states **1194**, **1196** and **1200**, if the tile manager **192** can successfully get space for the uncompressed tile data, then the tile manager **192** moves to a state **1206** where the tile manager **192** finds the free block with the highest memory address. If there is a choice between two or more free memory blocks, the tile manager **192** chooses the one with the highest address to try to keep all of the expanded image data at the high address end of the cache file. Once the tile manager **192** finds the highest address block, it moves to a state **1208** to unlink the free block from the free memory link list.

There are actually two possibilities for the free memory link list when the tile manager **192** is looking for expanded memory. One is the uncompressed tile free list and the other is the cache reserve list. In either case, the tile manager **192** unlinks the block of memory that the tile manager **192** is interested in from the free list and relinks the remaining memory blocks of the affected free list.

The tile manager **192** then transitions to a state **1210** to initialize the newly allocated block to all background color. Then the tile manager **192** moves to a state **1212** to move the description of the memory block (a pointer to the tile header) to the front of the most recently used tile list. Moving to a state **1214**, the tile manager **192** updates the soft uncompressed cache memory usage counter that was checked at the state **1182**. The tile manager **192** continues to a state **1216** to store the memory address in the tile header. The memory block that the tile manager **192** has just allocated is a pointer that is stored in the tile header data structure. That is how the memory block is associated with the tile. Then the tile manager **192** terminates normally from the function **438** at the end state **1204**.

FIG. **33** illustrates the process by which the present invention expands the compressed version of a tile to create an uncompressed version. Specifically, as shown in FIG. **33**, the tile manager **192** starts at a start state **1220** and moves to a test function at state **1222**, where the tile manager **192** determines whether the compressed version of the tile, or the compressed tile data, is in cache memory. If it is not, then the tile manager **192** moves to state **1224**, wherein the system loads the necessary data from the disk. If there is an error detected at state **1224**, the tile manager **192** moves to state **1228** to terminate the process.

From state **1226**, if compressed data was successfully loaded from the disk or from state **1222** if it was in cache

to begin with, the tile manager **192** moves to state **1230**, wherein the tile manager **192** locks the compressed tile image data. This step simply increments the lock count on the compressed memory state. From state **1230**, the system moves to state **1232**, wherein the tile manager **192** allocates and locks the uncompressed tile memory block. The system then moves to state **1234** to determine whether an error occurred at state **1232**. If so, the tile manager **192** moves to state **1236** and unlocks the compressed tile data. From state **1236**, the system moves to state **1238** to report the error. The system then terminates at end state **1228**.

On the other hand, if no error existed as determined at state **1234**, the system moves to state **1240**, wherein the tile manager **192** uncompresses the compressed data. Next, the tile manager **192** moves to state **1242** to determine whether an error occurred at state **1240**. If an error occurred at state **1240**, the tile manager **192** moves to state **1236** and functions as described previously. Otherwise, the tile manager **192** moves to stat **1244** to unlock the compressed and uncompressed data, and then terminates at end state **1228**.

FIG. **34** illustrates a process for creating compressed low resolution tiles from compressed higher resolution tiles. The tile manager **192** starts at start state **1250** and proceeds to state **1252**, wherein the system enters a loop which is followed by the system for each of the four high resolution tiles required to produce a single low resolution tile. More specifically, at state **1252** the tile manager **192** locks the compressed version of the high resolution tile. The system then proceeds to state **1256**, wherein the tile manager **192** determines whether an error occurred at state **1254**. In the event that an error occurred, the tile manager proceeds to end state **1258** and terminates. If no error occurred, the tile manager **192** returns to state **1252** and continues the loop described above for each of the four high resolution tiles.

After processing all four high resolution tiles as described, the system proceeds to state **1260** where the tile manager **192** scales the compressed data to half resolution. The process performed at state **1260** results in a compressed version of the low resolution tile. Then the tile manager **192** moves to a loop represented by states **1262**, **1264**, wherein for each of the high resolution tiles the tile manage **192** unlocks the compressed version of the tile.

Next, the tile manager **192** moves to state **1266** where the tile manager **192** allocates and locks memory for the compressed version of the low resolution tile. At state **1266**, the tile manager **192** actually puts the compressed version of the low resolution tile in a general, common buffer that is large enough to hold the maximum possible size of the compressed results. The actual valid data is usually much less than that than the maximum possible size, so the tile manager **192** only saves the valid amount of data.

From state **1266**, the system moves to state **1268** to determine whether an error occurred at state **1266**. If an error occurred, the system moves to end state **1258** and terminates. Otherwise, the system moves to state **1270** where the tile manager **192** copies the compressed data out of the temporary compressed data buffer into the newly allocated space in the cache. Then the tile manager **192** moves to state **1272** where the tile manager **192** unlocks the compressed version of the low resolution tile that now contains valid data. The system then terminates normally at state **1258**.

Now referring to FIG. **35**, a process is shown whereby the system resamples uncompressed high resolution tiles to an uncompressed low resolution tile. The tile manager **192** starts at start state **1280** and moves to state **1282**, wherein the tile manager **192** locks the uncompressed version of a single high resolution tile. This function scales a single high resolution tile to update one quarter of a tile in the half-resolution subimage. That quarter tile is rescaled to update one-sixteenth of a tile in the quarter-resolution subimage. This continues to the lowest resolution subimage. Next, the tile manager **192** proceeds to state **1284** to determine whether an error occurred in locking the uncompressed version of the high resolution tile. If there Was an error, then the tile manager **192** proceeds to state **1286** and terminates with an error condition. Otherwise, the tile manager **192** moves to state **1288** where the tile manager **192** determines how many levels of the subimage are to be updated. This function can be used to update a subset of subimages or the entire image stack in the case where a single tile is modified in the full resolution subimage. It will propagate that change all the way down to the lowest-resolution subimage in the image stack.

Next, the tile manager **192** proceeds to state **1290** where the tile manage **192** determines the tile index that is to be updated. In accordance with the present invention, when a change is propagated from the higher resolution down to the low resolution of tiles, the system calculates which tile corresponds to the affected area. Then the tile manager **192** moves to state **1290** where the tile manager **192** determines whether the low resolution tile that the tile manager **192** is about to update is marked as loaded or not. This step is intended for the situation in which not all of the low resolution subimages are populated during the loading of a raster image.

If the system determines that one or more low resolution tiles are not loaded, the system proceeds to state **1294**, wherein the tile manager **192** invalidates all of the low resolution tiles that would otherwise be affected by the change. The system then exits normally at end state **1286**. If the low resolution tile is about to be modified is loaded, as determined at state **1292**, the tile manager **192** moves to state **1296**, wherein the system locks the uncompressed version of the low resolution tile. The tile manager **192** then moves to state **1298** to determine whether an error occurred at state **1296** and, if so, the system moves to end state **1286** to terminate. Otherwise, the system moves to state **1300**. wherein the tile manager **192** scales the raster data from the high resolution tile down to the low resolution tile. Then the tile manager **192** moves to state **1302** where the tile manager **192** unlocks the high resolution tile.

Next, the system moves to state **1304**, wherein the tile manager **192** recursively modifies the loop variables such that the low resolution tiles that the tile manager **192** just finished updating become the high resolution tiles for the next succeeding iteration. Once all the subimages have been updated as described, the system exits at end state **1286**.

Now referring to FIGS. **36A** and **36B**, a process to collect free cache is shown. This process can be called from several other processes. The tile manager **192** begins at start state **1310** in FIG. **36A** and moves to state **1312** to determine whether a cache collection operation is in process. If so, the system exits at end state **1314**. This prevents recursive calls to collect free cache which might otherwise occur. If the system at state

1312 determines that no collection is in progress, then the tile manager 192 moves to state 1316 where the tile manager 192 sets a flag indicating that a collection is in progress.

From state 1316, the system moves to state 1320, where the tile manager 192 estimates the number of memory blocks to free in this operation. The reason for freeing a number of blocks instead of just one block is to reduce the computational overhead associated with the cache collection operations. The tile manager 192 typically estimates the amount of memory required to equal the number of tiles in a single row of the full resolution subimage of the document associated with the most recently used tile.

Once this estimate has been made, the system proceeds to state 1322 wherein the tile manager 192 considers the options that the tile manager 192 passed into this function. There are three options. One, as indicated at state 1324, is to reduce the uncompressed cache usage only while not affecting the compressed data that is currently held in cache. The second option, indicated at state 1328, is to reduce the compressed cache memory usage only. The third option, indicated at state 1326, is to reduce the total cache memory usage including both compressed and uncompressed data.

From state 1324 or state 1326, the tile manager 192 moves to state 1330, where the tile manager 192 stores all of the free states currently in the uncompressed free list into the cache reserve list. As the tile manager 192 performs the process in state 1330, the tile manager 192 attempts to consolidate the memory blocks. That is, if there are two free blocks that are adjacent to one another, the system automatically turns them into a single, larger contiguous block. From state 1328, on the other hand, the system moves to state 1358, shown in FIG. 36B and discussed below.

From state 1330, the tile manager 192 moves to state 1332, wherein the tile manager 192 determines whether the tile manager 192 has created a memory block large enough to satisfy the initial request. If so, the tile manager 192 terminates normally at end state 1314. Otherwise, the tile manager 192 moves to state 1334 where the tile manager 192 frees any unlocked, uncompressed tiles which are blank. The tile manager 192 then moves to state 1336 where the tile manager 192 determines whether the tile manager 192 has free sufficient memory. If so, the tile manager 192 exits at end state 1314. Otherwise, the tile manager 192 moves to state 1338 where the tile manager 192 frees unlocked, unpreserved uncompressed tiles that have valid compressed versions in cache or are on a disk, or that have valid, uncompressed versions on the disk beginning with the least recently used tile. After having freed that particular class of tiles, if the tile manager 192 determines, at state 1340, that the memory request has been satisfied, the tile manager 192 moves to state 1314 and terminates. Otherwise, the tile manager 192 moves to state 1342, shown in FIG. 36B.

Now referring to FIG. 36B, the tile manager 192 begins at state 1342, wherein the tile manager 192 compresses the free unlocked, unpreserved uncompressed tiles that don't have a valid compressed version or other source from which the tile can be recreated. To do this the tile manager 192 processes expanded tile data through a compression algorithm. The tile manager 192 then creates a compressed version of that tile so that the uncompressed version of the tile can be discarded.

Next, the tile manager 192 moves to state 1344, wherein the system determines whether the request made at state 1342 has been satisfied. If so, the system terminates at end state 1346. Otherwise, the system moves to state 1348, wherein the tile manager 192 frees unlocked, but preserved uncompressed tiles that have valid compressed or uncompressed copies. The tile manager 192 preferentially frees the oldest such tiles.

From the state 1348, the tile manager 192 proceeds to a decision state 1350 to test whether the request made at the state 1348 was satisfied. If so, the function 446 is terminated at the end state 1346. Otherwise, the tile manager 192 moves to a state 1352 to compress and then free unlocked, but preserved, uncompressed tiles that do not have valid compressed versions.

Next, the tile manager 192 moves to state 1354, wherein the system determines whether the request made at state 1352 has been satisfied. If so, the system terminates at end state 1346. Otherwise, the system moves to state 1356, wherein the tile manager 192 determines whether to free data memory blocks. If not, the system terminates at state 1346. Otherwise, the system moves to state 1358, to free unlocked preserved, uncompressed tiles that don't have valid compressed versions already.

The system next moves to state 1360 to determine whether the request has been satisfied. If so, the system terminates at state 1346. Otherwise, the system moves to state 1362 to print an error message, and then terminate at state 1346.

Now referring to FIG. 37, the tile manager 192 starts at state 1380 and moves to state 1382 where the tile manager 192 determines whether the uncompressed version is in fact still locked—that is if the lock count for uncompressed version of that tile is non-zero. If the tile is still locked then the tile manager 192 moves to state 1384 and prints a warning message. Then the tile manager 192 terminates at end state 1386.

If, at state 1382, the system determined that the uncompressed version is not locked, then the tile manager 192 moves to state 1388 where the tile manager 192 determines whether the uncompressed data has already been freed. If it has then the tile manager 192 terminates at end state 1386. Otherwise, the tile manager 192 moves to state 1390 where the tile manager 192 unlinks the uncompressed memory state from the most recently used list.

From state 1390, the tile manager 192 moves to state 1392 where the tile manager 192 updates and decrements the total uncompressed memory usage counter by the appropriate amount. The tile manager 192 then moves to state 1394 where the tile manager 192 moves the memory block to the uncompressed memory free list. In accordance with the present invention, the tile manager 192 keeps the list sorted by decreasing address. Consequently, when the tile manager 192 allocates expanded memory blocks, the tile manager 192 tends to choose the preferred blocks that have higher addresses because they are at the front of the free list.

Next, the tile manager 192 moves to state 1396, wherein the tile manager 192 sets a pointer in the tile header to null and the tile manager 192 sets the uncompressed tile status flags. This ensures that the tile header reflects the fact that it no longer has an uncompressed data associated with it. Then the tile manager 192 terminates at end state 1386.

Now referring to FIG. 38, a process by which the system compresses a tile is shown. The system begins at

start state **1400**, and moves to state **1402**, wherein the tile manager **192** determines whether the uncompressed tile data is in cache memory. If it is not, the tile manager **192** moves to state **1404** and loads the uncompressed data into cache memory from the disk. The system then moves to state **1406**, to determine whether an error occurred at state **1404**. If so, the system terminates at end state **1408**. Otherwise, the system proceeds to state **1410**.

At state **1410**, the tile manager **192** locks the uncompressed tile data, and then moves to state **1412**, to determine whether an error occurred at state **1410**. If an error occurred, the system terminates at end state **1408**. Otherwise, the system moves to state **1414**, wherein the tile manager **192** compresses the image data into a common buffer. For binary images of text and line drawings, the tile manager **192** uses a CCITT group **4** encoding.

From state **1414**, the tile manager **192** moves to state **1416** to determine whether an error occurred at state **1414**. If an error indeed occurred, the system moves to state **1418** to unlock the uncompressed tiles, and then exits at end state **1408**. Otherwise, the system proceeds to state **1420**, wherein the tile manager **192** allocates and locks cache memory space for the compressed tile data.

From state **1420**, the system proceeds to state **1422** to determine whether an error occurred at state **1420**. If an error occurred, the system moves to state **1418** and proceeds as described above. Otherwise, the system moves to state **1424**, wherein the tile manager **192** copies the compressed data from the common buffer into the newly allocated cache memory state. The system moves from state **1424** to state **1426**, wherein the tile manager **192** unlocks the compressed and uncompressed tile data and then terminates at end state **1408**.

While the above detailed description has shown, described and pointed out the fundamental novel features of the invention as applied to various embodiments, it will be understood that various omissions and substitutions and changes in the form and details of the device illustrated may be made by those skilled in the art, without departing from the spirit of the invention.

What is claimed is:

1. An image memory management system, comprising:

a computer having a processor and an image memory, the image memory comprising a main memory and a secondary memory;

an image stack, located in the image memory, comprising a plurality of similar digital images, each digital image having a plurality of pixels grouped into at least one tile, and each digital image having a resolution different from the other digital images;

means for accessing a selected one of the tiles in the image stack;

first means for transferring a selected one of the tiles from the secondary memory to the main memory when the tile is accessed by the accessing means and the tile is absent from the main memory; and

second means for transferring a selected one of the tiles from the main memory to the secondary memory when the main memory is full.

2. The system defined in claim **1**, additionally comprising means for modifying a selected one of the tiles.

3. The system defined in claim **2**, wherein the second transferring means only transfer tiles that have been modified by the modifying means.

4. The system defined in claim **1**, wherein the main memory is semiconductor memory.

5. The system defined in claim **1**, wherein the secondary memory is a magnetic disk.

6. The system defined in claim **1**, wherein each tile is square.

7. The system defined in claim **1**, wherein a lowest resolution digital image comprises one tile.

8. The system defined in claim **1**, wherein a preselected digital image in the image stack is resampled to obtain another digital image in the image stack.

9. The system defined in claim **1**, wherein at least one of the digital images is compressed.

10. The system defined in claim **1**, wherein the accessing means is responsive to an image access operation selected by a user.

11. The system defined in claim **10**, wherein the image access operation is zooming or panning the image.

12. The system defined in claim **10**, wherein the image access operation is reversible.

13. A method of managing images in a computer having a processor and an image memory comprising a slower access memory and a faster access memory, comprising the steps of:

creating a digital image;

resampling the digital image so as to form an image stack comprising the digital image and one or more lower resolution digital images;

dividing each image into equal sized, rectangular tiles; and

evaluating a location in the image memory of tiles in each digital image of the image stack in a given region of interest.

14. The method defined in claim **13**, additionally comprising updating modified regions of all images when an edit operation is completed.

15. The method defined in claim **13**, wherein the evaluating step includes the following order of decreasing availability:

exists in the faster access memory in uncompressed form;

exists in the slower access memory in uncompressed form;

exists in the faster access memory in compressed form;

exists in the slower access memory in compressed form; and

must be constructed from higher resolution tiles.

16. The method defined in claim **13**, wherein the evaluating step includes the following order of decreasing availability:

exists in the faster access memory in uncompressed form;

exists in the slower access memory in uncompressed form;

exists in the slower access memory in compressed form; and

must be constructed from higher resolution tiles.

17. The method defined in claim **13**, wherein the evaluating step includes selecting the digital image with the lowest resolution higher than a requested resolution at a given view scale.

* * * * *

United States Patent [19]

Delorme

[11] **Patent Number: 4,972,319**

[45] **Date of Patent: Nov. 20, 1990**

[54] **ELECTRONIC GLOBAL MAP GENERATING SYSTEM**

[76] Inventor: **David M. Delorme**, 356 Range Rd., Cumberland, Me. 04021

[21] Appl. No.: **101,315**

[22] Filed: **Sep. 25, 1987**

[51] **Int. Cl.**[5] **G09B 29/00**

[52] **U.S. Cl.** **364/419;** 434/150; 340/990

[58] **Field of Search** 364/419, 449; 434/150, 434/130; 340/990

[56] **References Cited**

U.S. PATENT DOCUMENTS

400,642	4/1889	Beaumont	283/34
751,226	10/1899	Van Der Grinten	283/34
752,957	2/1904	Colas	283/34
1,050,596	1/1913	Bacon	283/34
1,610,413	12/1924	Balch	283/34
2,094,543	9/1937	Lackey et al.	353/11
2,354,785	8/1944	Von Rohl	434/150
2,431,847	12/1947	Dusen	353/11
2,650,517	9/1953	Falk	355/77
3,248,806	5/1966	Schrader	434/150
3,724,079	4/1973	Jasperson et al.	33/15 B
4,315,747	2/1982	McBryde	434/150
4,673,197	6/1987	Stipelman et al.	434/150
4,689,747	8/1987	Krouse et al.	364/449
4,737,927	4/1988	Hanabusa et al.	340/990

OTHER PUBLICATIONS

"Equal-Area Projections for World Statistical Maps", McBryde and Thomas, U.S. Dept. of Commerce, Coast and Geodetic Survey, Spec. Pub. 245, 1949.

"The Quadtree and Related Hierarchical Data Structures", Hanan Samet, Computer Surveys, vol. 16, No. 2, Jun. 1984.

Primary Examiner—Jerry Smith
Assistant Examiner—Kim T. Bui
Attorney, Agent, or Firm—Sughrue, Mion, Zinn, Macpeak & Seas

[57] **ABSTRACT**

A global mapping system which organizes mapping data into a hierarchy of successive magnitudes or levels for presentation of the mapping data with variable resolution, starting from a first or highest magnitude with lowest resolution and progressing to a last or lowest magnitude with highest resolution. The idea of this hierarchical structure can be likened to a pyramid with fewer stones or "tiles" at the top, and where each successive descending horizontal level or magnitude contains four times as many "tiles" as the level or magnitude directly above it. The top or first level of the pyramid contains 4 tiles, the second levle contains 16 tiles, the third contains 64 tiles and so on, such that the base of a 16 magnitude or level pyramid would contain 4 to the 16th power or 4,294,967,296 tiles. This total includes "hyperspace" which is later clipped or ignored. Digital data corresponding to each of the separate data base tiles is stored in the database under a unique filename.

33 Claims, 9 Drawing Sheets

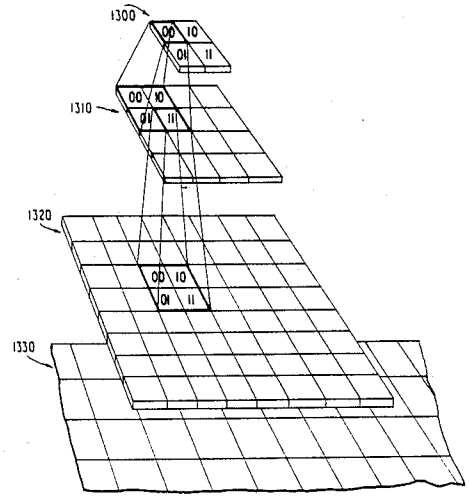


FIG. 1

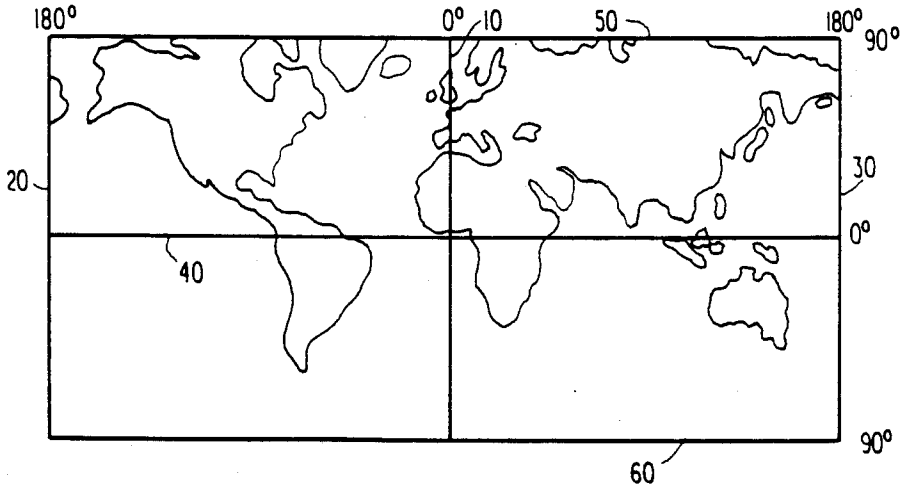


FIG. 2

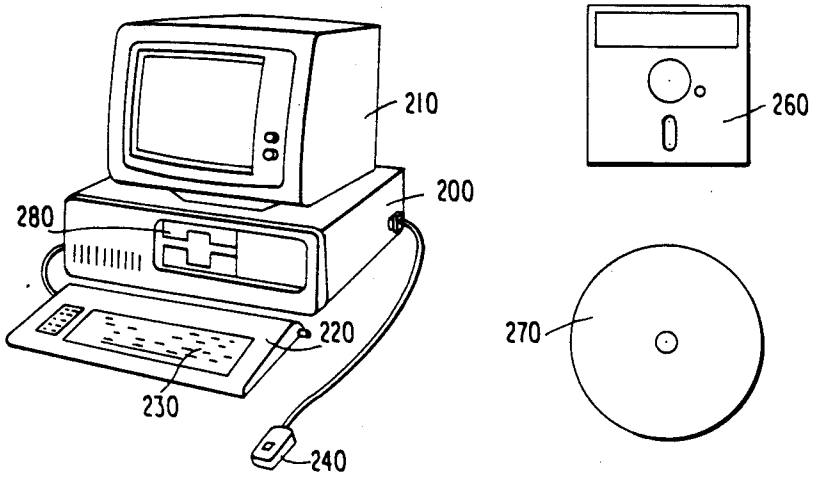


FIG. 3A

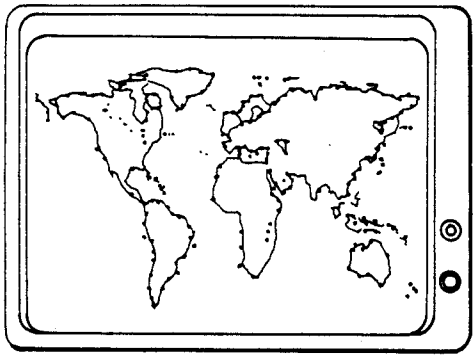


FIG. 3B

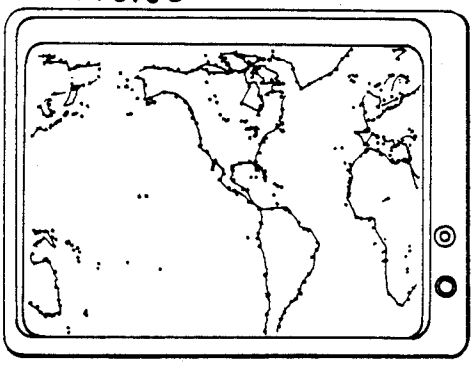


FIG. 3C

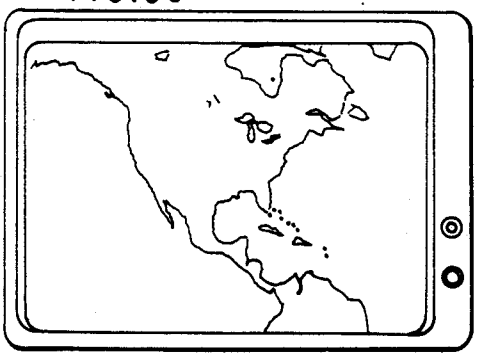


FIG. 3D

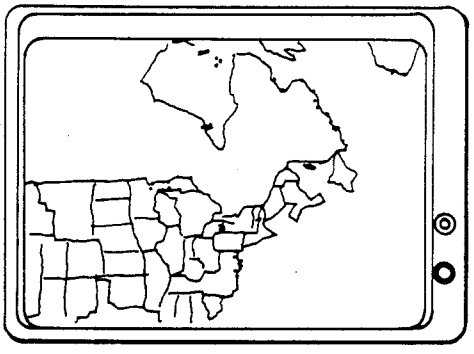


FIG. 3E

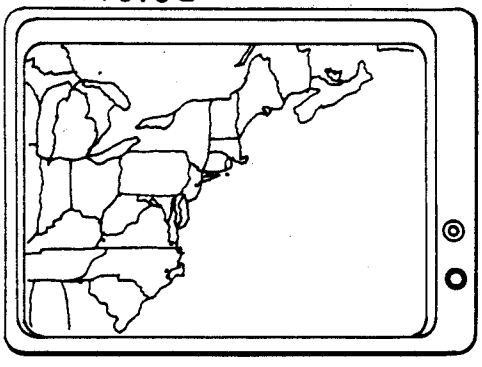


FIG. 3F

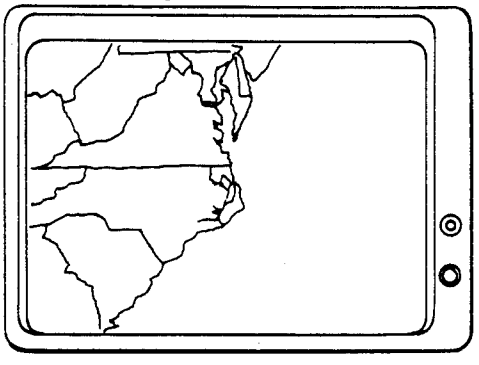


FIG.4

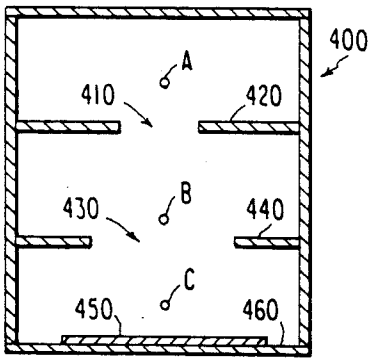


FIG.5A

FIG.5B

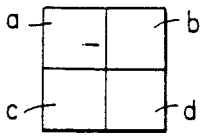


FIG.6

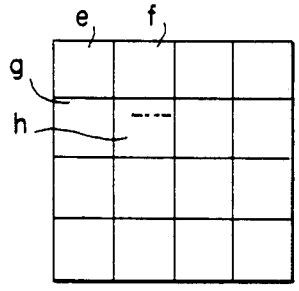


FIG.7

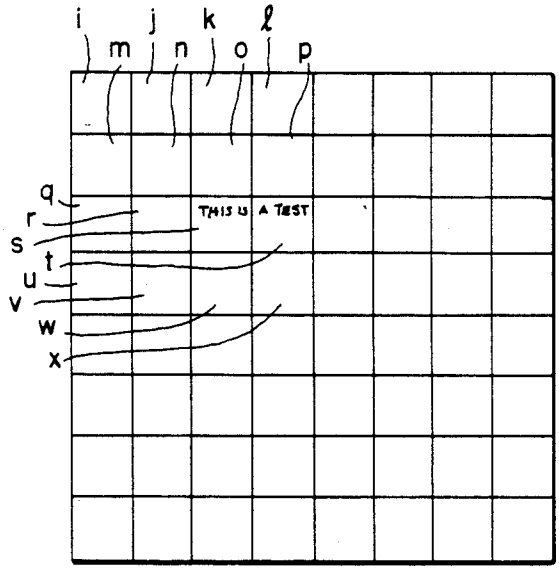
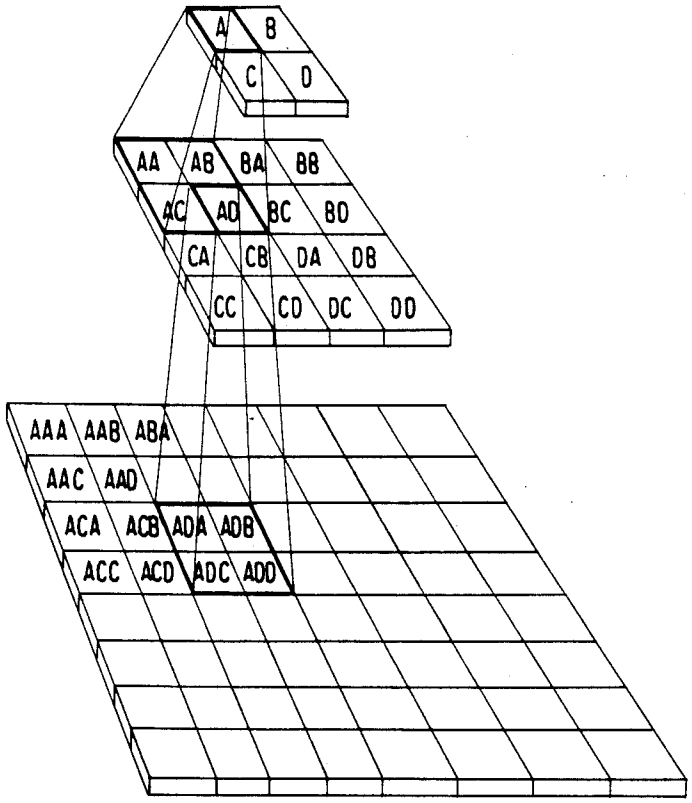


FIG. 8



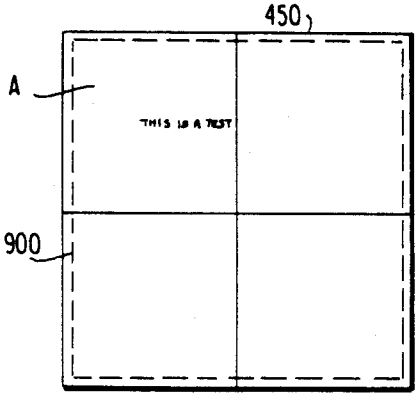
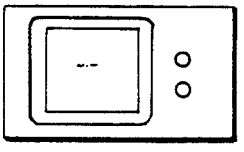


FIG. 9A

FIG. 9B



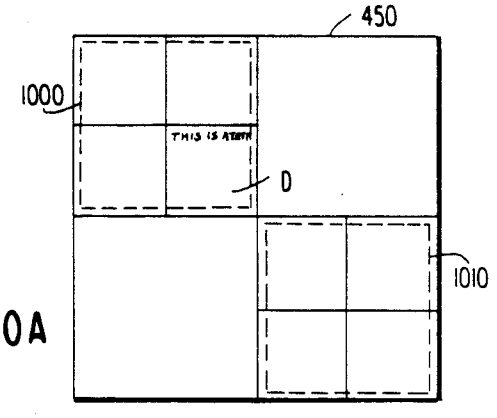
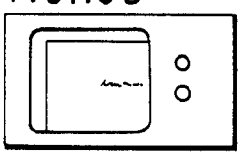


FIG. 10A

FIG. 10B



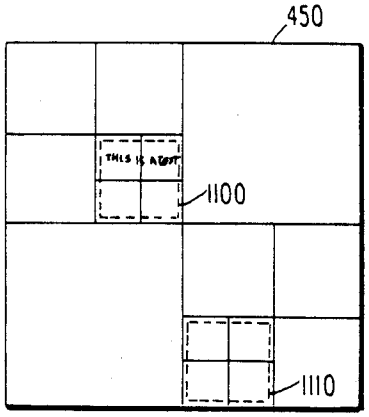


FIG.11A

FIG.11B

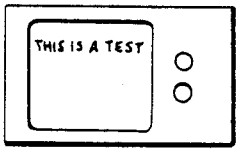
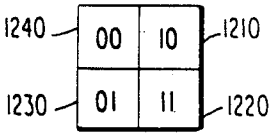


FIG.12



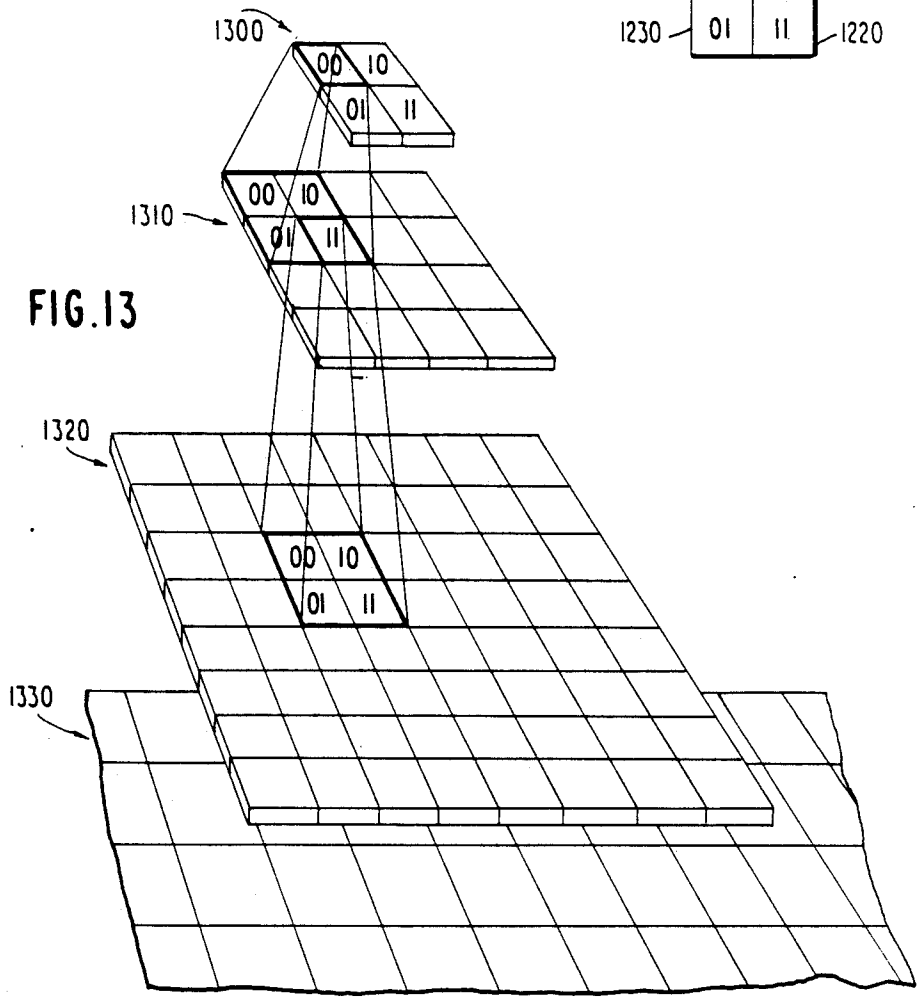


FIG.13

FIG.14

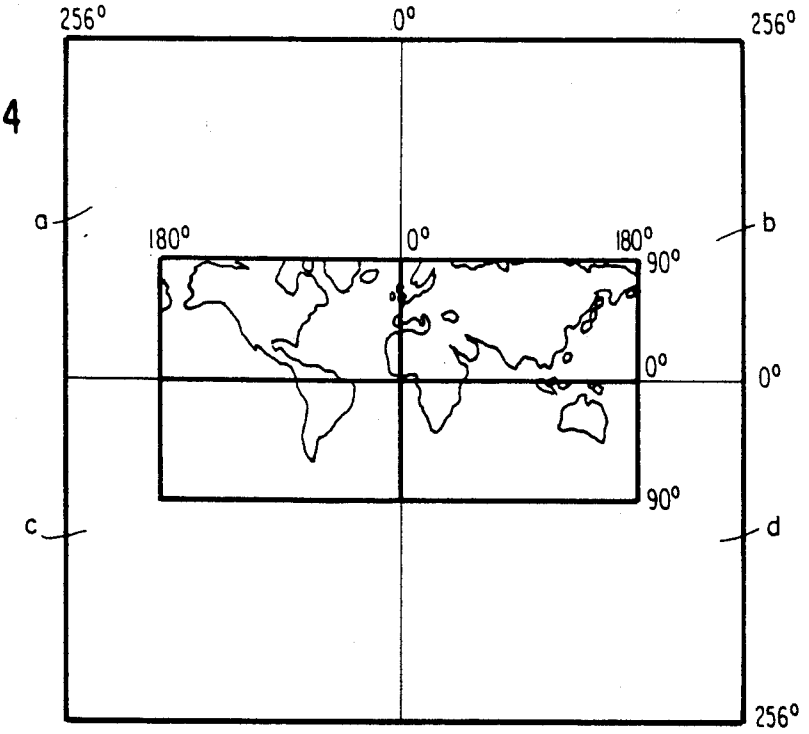

256°
0°
256°
a
b
180°
0°
180°
90°
0°
0°
90°
c
d
256°

FIG.15

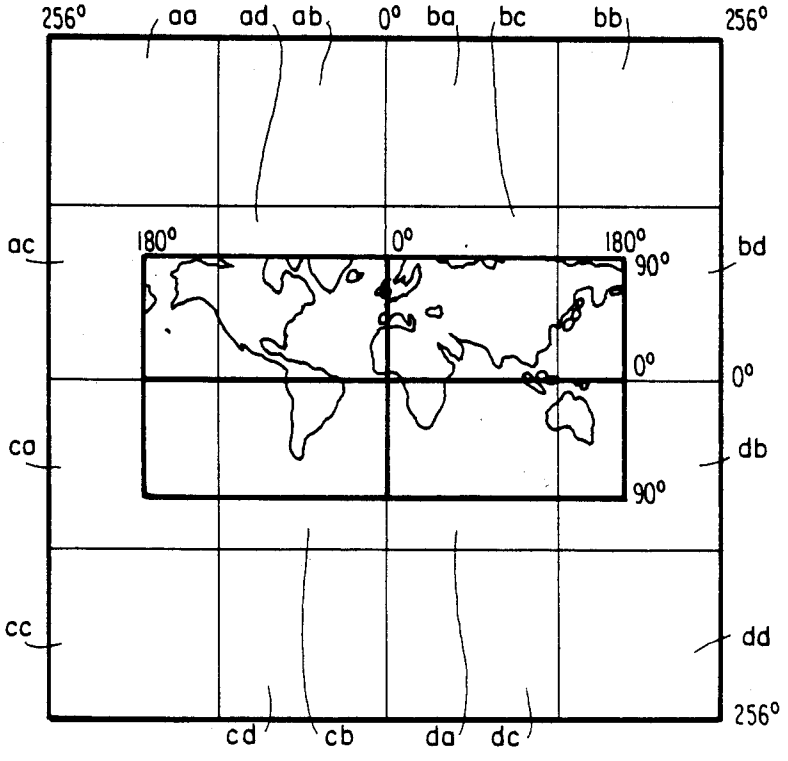

256°
aa
ad
ab
0°
ba
bc
bb
256°
ac
180°
0°
180°
90°
bd
0°
0°
ca
db
90°
cc
dd
256°
cd
cb
da
dc

FIG. 16

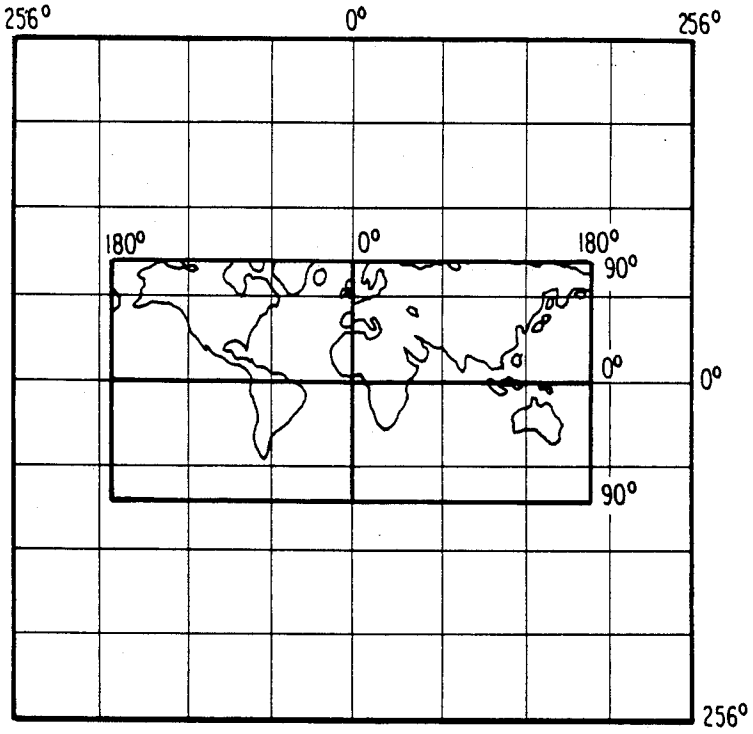


FIG. 17

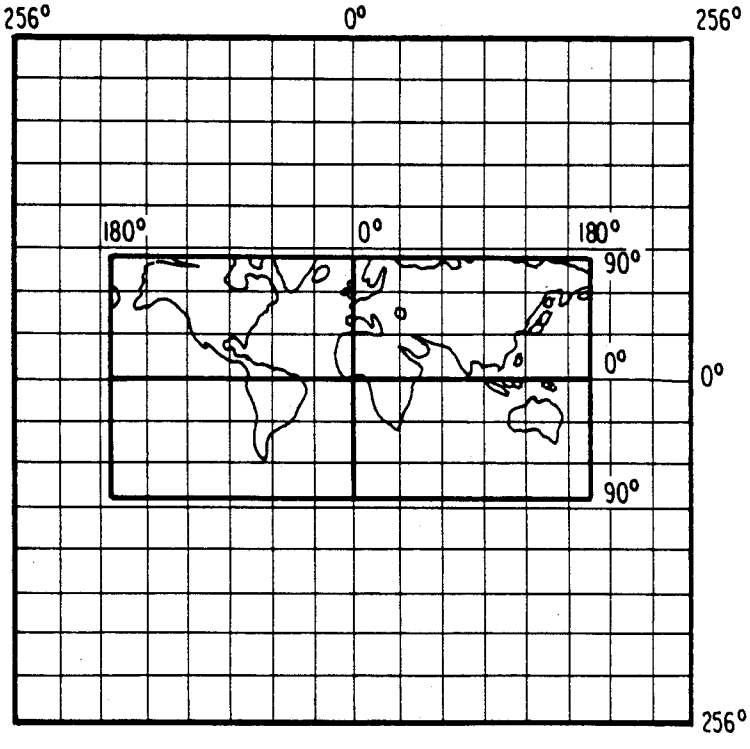


FIG. 18

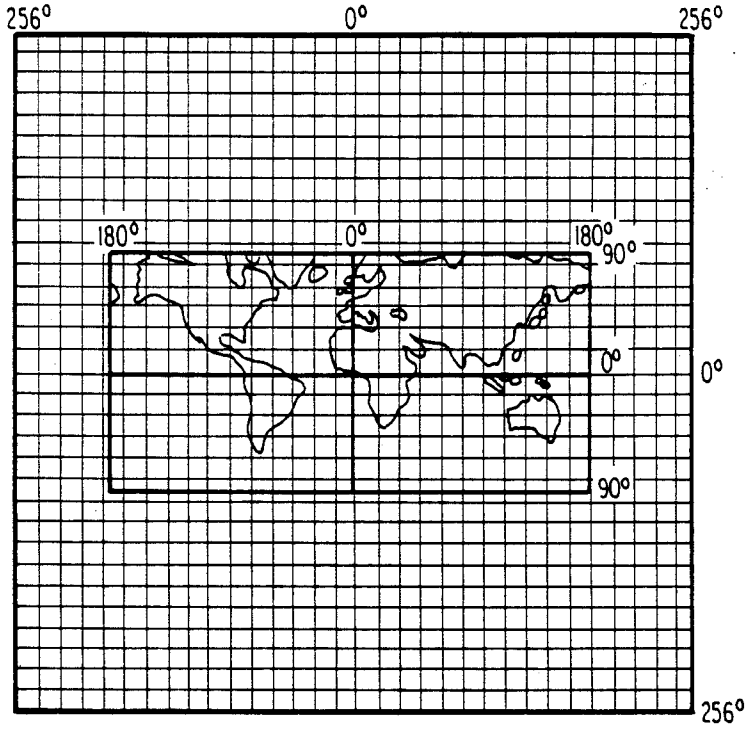


FIG. 19

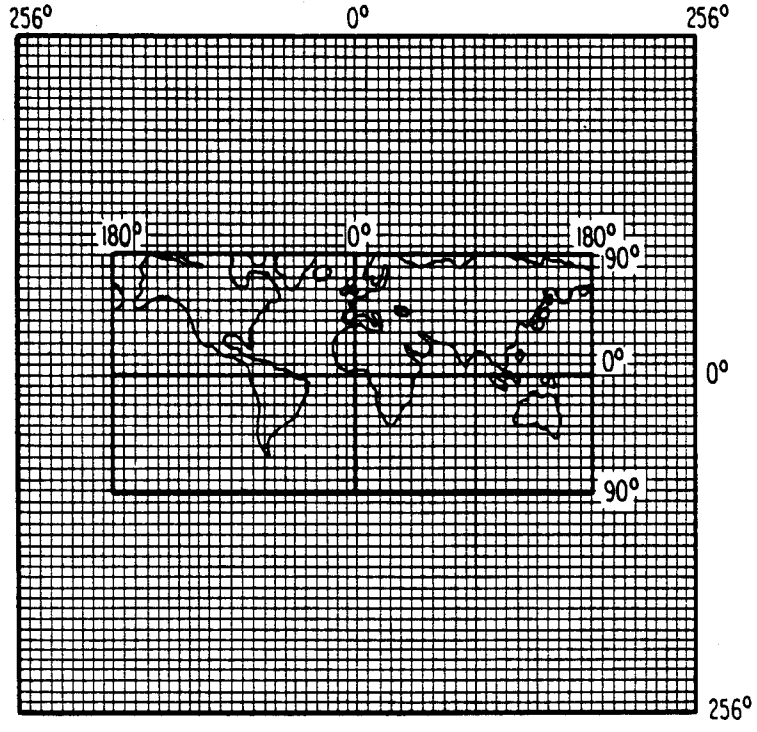
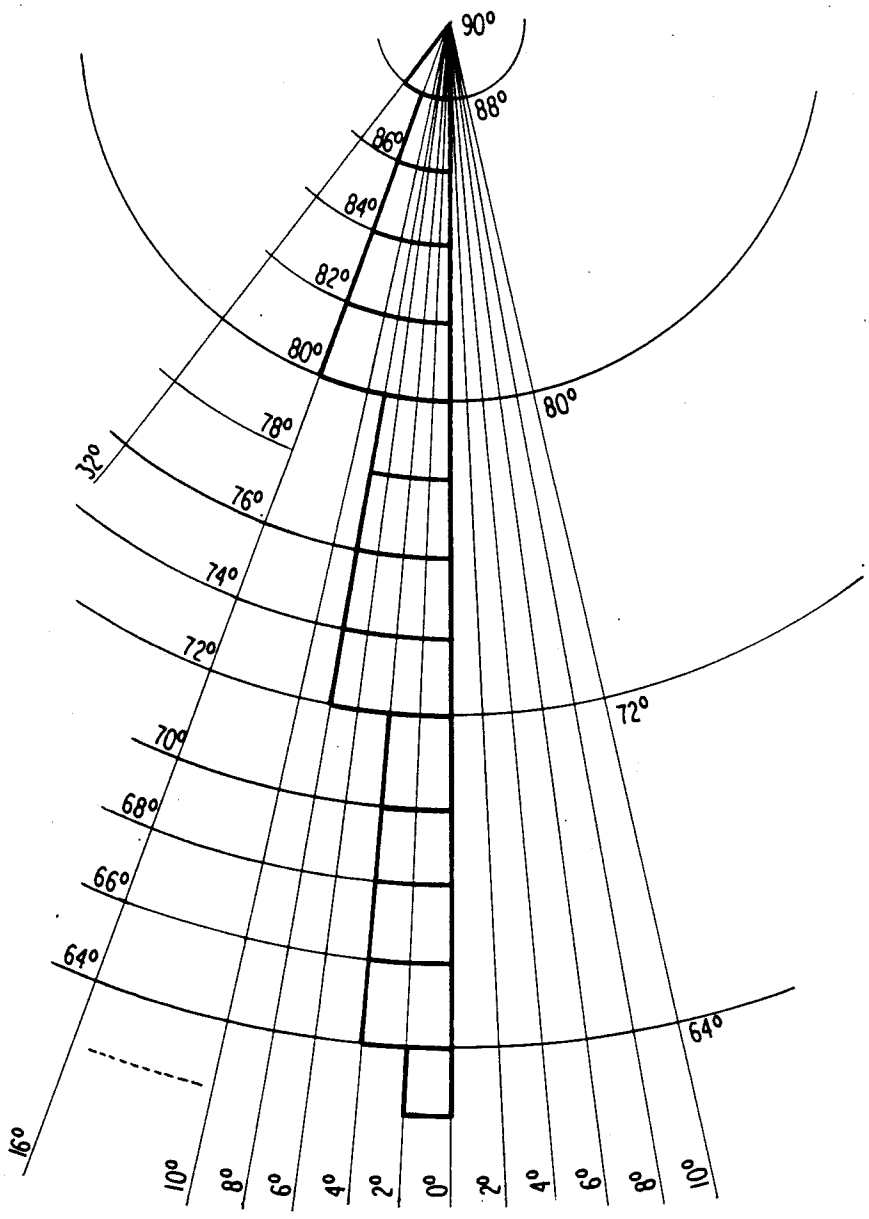


FIG. 20
ILLUSTRATION OF POLAR COMPRESSION
AT THE 8th MAGNITUDE



ELECTRONIC GLOBAL MAP GENERATING SYSTEM

BACKGROUND OF THE INVENTION

1. Technical Field

This invention relates to a new variable resolution global map generating system for structuring digital mapping data in a new data base structure. managing and controlling the digital mapping data according to new mapping data access strategies, and displaying the mapping data in a new map projection of the earth.

2. Background Art

Numerous approaches have been forwarded to provide improved geographical maps, for example:

U.S. Pat. No. 4,315,747, issued to McBryde on Feb. 16, 1982, describes a new map "projection" and intersecting array of coordinate lines known as the "graticule", which is a composite of two previously known forms of projection. In particular, the equatorial portions of the world are represented by a fusiform equal area projection in which the meridian curves, if extended, would meet at points at the respective poles, referred to as "pointed poles". In contrast, the polar regions of the world map are represented by a flat polar equal area projection in which the poles are depicted as straight horizontal lines with the meridians intersecting along its length. Thus, in a flat polar projection the meridian curves converge toward the poles but do not meet at a point and, instead, intersect a horizontal linear pole. The two component portions of the flat world map are joined where the parallels are of equal length. The composite is said to be "homolinear" because all of the meridian curves are similar curves, for example, sine, cosine or tangent curves, which merge where the two forms of projection are joined where the respective parallels are equal. The flat polar projections in the polar portions of the map provide a compromise with the Mercator cylinder projections, thereby greatly reducing distortion.

U.S. Pat. No. 1,050,596, issued to Bacon on Jan. 14, 1913, describes another composite projection for world maps and charts which uses a Mercator or cylindrical projection for the central latitudes of the earth and a convergent projection at the respective poles. In the central latitudes, the grids of the Mercator projection net or graticule are rectangular. In the polar regions, the converging meridians may be either straight or curved.

U.S. Pat. No 1,620,413, issued to Balch on Dec. 14, 1926, discusses gnomic projections from a conformal sphere to a tangent plane and Mercator or cylindrical projections from the conformal sphere to a tangent cylinder. Balch is concerned with taking into account the non-spherical shape of the earth, and therefore, devises the so-called "conformal sphere" which represents the coordinates from the earth whose shape is actually that of a spheroid or ellipsoid of revolution, without material distortion.

U.S. Pat. No. 752,957, issued to Colas on Feb. 23, 1904, describes a map projection in which a map of the entire world is plotted or transcribed on an oval constructed from two adjacent side by side circles with arcs joining the two circles. The meridians are smooth curves equally spaced at the equator, while the latitude lines are non-parallel curves.

U.S. Pat. No. 400,642 issued to Beaumont on Apr. 2, 1889, describes a map of the earth on two intersecting spheres, on which the coordinate lines of latitude and longitude are all arcs of circles.

U.S. Pat. No. 751,226, issued to Grinten on Feb. 2, 1904, represents the whole world upon the plane surface of a single circle with twice the diameter of the corresponding globe, the circle being delineated by a graticule of coordinates of latitude and longitude which are also arcs of circles.

U.S. Pat. No. 3,248,806, issued to Schrader on May 3, 1966, discloses a subdivision of the earth into a system of pivotally mounted flat maps, each map segment representing only a portion of the earth's surface in spherical projection on an equilateral spherical triangle to minimize distortion.

U.S. Pat. No. 2,094,543, issued to Lackey et al on Sept. 28, 1937, describes a projector for optically producing a variety of different map projections, including orthographic, stereographic and globular projections onto flat translucent screens and a variety of other projections on shaped screens.

U.S. Pat. No. 2,650,517, issued to Falk on Sept. 1, 1953, describes a photographic method for making geographical maps.

U.S. Pat. No. 2,354,785, issued to Rohl on Aug. 1, 1944, discloses two circular maps which are mounted side by side, and an arrangement for rotating the two maps in unison so that corresponding portions of the earth's surface are at all times in proper relationship.

U.S. Pat. No. 3,724,079, issued to Jasperson et al on Apr. 3, 1973, discloses a navigational chart display device which is adapted to display a portion of a map and enable a pilot to fix his position, to plot courses and to measure distances.

U.S. Pat. No. 2,431,847 issued to Van Dusen on Dec. 2, 1947, discloses a projection arrangement, in which a portion of the surface of a spherical or curved map may be projected in exact scale and in exact proportional relationship.

McBryde and Thomas, *Equal Area Projections for World Statistical Maps*, Special Publication No. 245, Coast & Geodetic Survey 1949.

In addition to the above further teachings as to geographical mapping can be found in the *Elements of Cartooraphy*, 4th edition which was written by Arthur Robinson, Randall Sale and Joel Morrison, and published by John Wiley & Sons (1978).

The present invention seeks to provide a low cost and efficient mapping system which allows the quick and easy manipulation of and access to an extraordinary amount of mapping information, i.e., a mapping system which allows a user to quickly and easily access a detailed map of any geographical area of the world.

Map information can be stored using at least three different approaches, i.e., paper, analog storage and digital storage, each approach having its own advantages and disadvantages as detailed below.

The paper mapping approach has been around since papyrus and will probably exist for the next thousand years.

Advantages of paper storage:

inexpensive.

once printed, no further processing is required to access the map information, so not subject to processing breakdown.

Disadvantages of paper storage:

can become bulky and unwieldy when dealing with a large geographical area, or a large amount of maps.

paper does not have the processing capabilities or "intelligence" of computers, and therefore does not support automated search or data processing capabilities.

cannot be updated cheaply and easily.

The analog mapping approach is used to provide what is commonly known as videodisc maps. The information is stored as still frames under N.T.S.C. (National Television Standards Committee) conventions. To make maps, a television camera moves across a paper map lying on a workbench. Every few inches a frame is recorded on videotape. After one row of the map is completely recorded, the camera is moved down to the next row of frames to be recorded. This process is repeated until frames representing a checkerboard pattern of the entire map are recorded. The recorded videotape could be used to view the map: however, access time to scan to different areas of the recorded map is usually excessive. As a result, a videodisc, with its quicker access time, is typically used as the medium for analog map storage. The recorded videotape is sent to a production house which "stamps" out 8 inch or 12 inch diameter, videodiscs.

Advantages of the analog storage approach:

one side of a 12 inch videodisc can hold 54.000 "frames" of a paper map. A frame is typically equal to $2\frac{1}{2}\times3$ inches of the paper map.

access time to any frame can be fast usually under 5 seconds.

once located on the videodisc, the recorded analog map information will be used to control the raster scan of a monitor and to produce a reproduction of the map in 1/30th of a second.

through additional hardware and software, mapping symbols, text and/or patterhsn can be overlaid on top of the recorded frame.

Disadvantages of the analog storage approach:

the "frames" are photographed from paper maps, which, as mentioned above, cannot be updated cheaply or easily.

due to paper map projections, mechanical camera movements, lens distortions and analog recording electronics, the videodisc image which is reproduced is not as accurate as the original paper map.

as a result of the immediately above phenomena, latitude and longitude information which is extracted from the reproduced image cannot be fully trusted.

if a major error is made in recording any one of the 54,000 frames, it usually requires redoing and re-stamping.

since frames cannot be scrolled, most implementations employ a 50% overlap technique. This allows the viewer to jump around the database with a degree of visual continuity: however, this is at a sacrifice of storage capacity. If the frame originally covered $2\frac{1}{2}\times3$ inches or approximately 8 square inches of the paper map, the redundant overlap information is 6 square inches, leaving only 2 square inches of new information in the centroid of each frame.

as a result of the immediately above deficiency, a 2×3 foot map containing 864 square inches would require 432 frames; thus, only 125 paper maps could be stored on one side of a 12 inch videodisc.

must take hundreds of video screen dumps to make a hard copy of a map area of interest and, even then, the screens do not immediately splice together because of the overlap areas.

the biggest disadvantage is that, since frames have to be arranged in a checkerboard fashion, there is no way to jump in directions other that north, south, east or west and maintain visual continuity. As an example, the visual discontinuity in viewing a "great circle" route from Alaska to New York would be unbearable for all but the most hearty.

The digital mapping approach has been around for at least 20 years and is much more frequently used than the analog approach. Digital data bases are stored in computers in a format similar to text of other databases. Unlike map information on a videodisc, the outstanding map features are stored as a list of objects to be drawn, each object being defined by a plurality of vector "dot" coordinates which define the crude outline of the object. As one example, a road is drawn by connecting a series of dots which were chosen to define the path (i.e., the "outline") of the road. Once drawn, further data and processing can be used to smooth the crude outline of the object, place text, such as the name or description of the object in a manner similar to what happens when drawing on a paper map.

Advantages of the digital approach:

digital maps are the purest form of geographical mapping data: from them, paper and analog maps can be produced.

digital maps can be quickly and easily updated in near real-time, and this updating can be in response to data input from external sources (e.g., geographical monitoring devices such as satellite photography).

digital maps can be easily modified to effect desirable mapping treatments such as uncluttering, enhancing, coloring, etc.

digital maps can be easily and accurately scaled, rotated and drawn at any perspective view point.

digital maps can be caused to reproduce maps in 3-D.

digital maps can drive pen-plotters (for easy paper reproductions), robots, etc.

digital maps can be stored on any mass storage device.

Disadvantages of the digital approach:

digital maps require the use or creation of a digital database: this is a very time-consuming and expensive process, but once it is made, the data base can be very easily copied and used for many different projects.

The digital approach is utilized with the present invention, as this approach provides overwhelming advantages over the above-described paper and analog approaches.

In designing any mapping system, several features are highly desirable:

First, it is highly desirable that the mapping system be of low cost.

Second, and probably most important, is access time. Not only is it generally desirable that the desired map section be accessible and displayed within a reasonable amount of time, but in some instances, this access time is critical.

In addition to the above, the present invention (as mentioned above), seeks to provide a third important feature,—a mapping system which allows the manipulation of and access to an extraordinary amount of mapping information, i.e., a mapping system which allows a user to quickly and easily access a detailed map of any geographical area of the world.

A tremendous barrier is encountered in any attempt to provide this third feature. In utilizing the digital approach to map a large geographical area in detail

(e.g., the earth), one should be able to appreciate that the storage of mapping data sufficient to accurately define all the geographical features would represent a tremendous data base.

While there have been digital mapping implementations which have successfully been able to manipulate a tremendous data base, these implementations involve tremendous cost (i.e., for the operation and maintenance of massive mainframe computer and data storage facilities). Furthermore, there is much room for improvement in terms of access time as these mainframe implementations result in access times which are only as quick as 20 seconds. Thus, there still exists a need for a low-cost digital mapping system which can allow the storage, manipulation and quick (i.e., "real time") access and visual display of a desired map section from a tremendous mapping data base.

There are several additional mapping system features which are attractive.

It is highly desirable that a mapping system be sensitive to and compensate for distortions caused by mapping curved geographical (i.e., earth) surfaces onto a flat, two-dimensional representation. While prior art approaches have provided numerous methods with varying degrees of success, there is a need for further improvements which are particularly applicable to the digital mapping system of the present invention.

It is additionally attractive for a mapping system to easily allow a user to change his/her "relative viewing position", and that in changing this relative position, the change in the map display should reflect a feeling of continuity. Note that the "relative viewing position should be able to be changed in a number of different ways. First, the mapping system should allow a user to selectively cause the map display to scroll or "fly" along the geographical map to view a different (i.e., "lateral") position of the geographical map while maintaining the same degree of resolution as the starting position. Second, the mapping system should allow a user to selectively vary the size of the geographical area being displayed (i.e., "zoom") while still maintaining an appropriate degree of resolution, i.e., allow a user to selectively zoom to a higher "relative viewing position" to view a larger geographical area with lower resolution regarding geographical, political and cultural characteristics, or zoom to a lower "relative viewing position" to view a smaller geographical area with higher resolution. (Note that maintaining the appropriate amount of resolution is important to avoid map displays which are effectively barren or are cluttered with geographical, political and cultural features.) Again, while prior art approaches have provided numerous methods with varying degrees of success, there is a need for further improvements which are particularly applicable to the digital mapping system of the present invention.

The final feature concerns compatibility with existing mapping formats. As mentioned above, the creation of a digital database is a very tedious, time-consuming and expensive process. Tremendous bodies of mapping data are available from many important mapping authorities, for example, the U.S. Geological Survey (USGS), Defense Mapping Agency (DMA), National Aeronautics and Space Administration (NASA), etc. In terms of both being able to easily utilize the mapping data produced by these agencies, and represent an attractive mapping system to these mapping agencies, it would be highly desirable for a mapping system to be compatible with all of the mapping formats used by these respective agencies. Prior art mapping systems have been deficient in this regard; hence, there still exists a need for such a mapping system.

SUMMARY OF THE INVENTION

The present invention provides a digital mapping method and system of a unique implementation to satisfy the aforementioned needs.

The present invention provides a computer implemented method and system for manipulating and accessing digital mapping data in a tremendous data base, and for the reproduction and display of electronic display maps which are representative of the geographical, political and cultural features of a selected geographical area. The system includes a digital computer, a mass storage device (optical or magnetic), a graphics monitor, a graphics controller, a pointing device, such as a mouse, and a unique approach for structuring, managing, controlling and displaying the digital map data.

The global map generating system organizes the mapping data into a hierarchy of successive magnitudes or levels for presentation of the mapping data with variable resolution, starting from a first or highest magnitude with lowest resolution and progressing to a last or lowest magnitude with highest resolution. The idea of this hierarchical structure can be likened to a pyramid with fewer stones or "tiles" at the top, and where each successive descending horizontal level or magnitude contains four times as many "tiles" as the level or magnitude directly above it. The top or first level of the pyramid contains 4 tiles, the second level contains 16 tiles, the third contains 64 tiles and so on, such that the base of a 16 magnitude or level pyramid would contain 4 to the 16th power or 4,294,967,296 tiles. This total includes "hyperspace" which is later clipped or ignored. Hyperspace is that excess imaginary space left over from mapping of 360 deg, space to a zero magnitude virtual or imaginary space of 512 deg, square.

A first object of the present invention is to provide a digital mapping method and system which are of low cost.

A second and more important object of the present invention is to provide a unique digital mapping method and system which allow access to a display of the geographical, political and cultural features of a selected geographical area within a minimum amount of time.

A third object of the present invention is to provide a digital mapping method and system which allow the manipulation of and access to an extraordinary amount of mapping information, i.e., a mapping method and system which allow a user to quickly and easily access a detailed map of any geographical area of the world.

Another object of the present invention is to provide a digital mapping method and system which recognize and compensate for distortion introduced by the representation of curved (i.e., earth) surfaces onto a flat two-dimensional display.

Still a further object of the present invention is to provide a digital mapping method and system which allow a user to selectively change his/her "relative viewing position", i.e., to cause the display monitor to scroll or "fly" to display a different "lateral" mapping position of the same resolution, and to cause the display monitor to "zoom" to a higher or lower position to display a greater or smaller geographical area, with an appropriate degree of resolution.

A fifth object of the present invention is to provide a digital mapping method and system utilizing a unique

mapping graticule system which allows mapping data to be compatibly adopted from several widely utilized mapping graticule systems.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, structures and features of the present invention will become more apparent from the following detailed description of the preferred mode for carrying out the invention; in the description to follow, reference will be made to the accompanying drawings in which:

FIG. **1** is an illustration corresponding to a flat projection of the earth's surface.

FIG. **2** is an illustration of a digital computer and mass storage devices which can be utilized in implementing the present invention.

FIGS. **3A–3F** are illustrations of monitor displays showing the ability of the present invention to display varying sizes of geographical areas at varying degrees of resolution.

FIG. **4** is a cross-sectional diagram of a simple building example explaining the operation of the present invention.

FIG. **5A** and B are plan view representations of a paper **450** as it is viewed from the relative viewing position A shown in FIG. **4**.

FIG. **6** is a plan view representation of a paper **450** as it is viewed from the relative viewing position B shown in FIG. **4**.

FIG. **7** is a plan view representation of a paper **450** as it is viewed from the relative viewing position C shown in FIG. **4**.

FIG. **8** is a pyramidal hierarchy of the data base file structure showing an example of the ancestry which exits between files.

FIG. **9A** is a plan view representation of a paper **450**, with the paper being divided into a first level of quadrant areas.

FIG. **9B** is an illustration of a monitor displaying a digital map of the area enclosed by the dashed portions in FIG. **9A**.

FIG. **10A** is a plan view representation of a paper **450**, with the upper-left and lower-right paper quadrant areas being further divided into quadrants.

FIG. **10B** is an illustration of a monitor displaying a digital map of the area enclosed by the upper-left dashed portion in FIG. **10A**.

FIG. **11A** is a plan view representation of a paper **450**, with several sections of the second level of quadrants being further divided into additional quadrants.

FIG. **11B** is a higher resolution display of the area enclosed within the dashed portion in FIG. **11A**.

FIG. **12** is a plan view illustration of a quadrant area division, with a two-bit naming protocol being assigned to each of the quadrant areas.

FIG. **13** is a pyramidal hierarchy of the data base files using the two-bit naming protocol of FIG. **12**, and showing an example of the ancestry which exits between files.

FIG. **14** is a plan view illustration of a 360°×**180°** flat projection of the earth being impressed in the 512°×512° mapping area of the present invention, with a first quadrant division dividing the mapping area into four equal 250°×256° mapping areas.

FIG. **15** is the same plan view illustration of FIG. **14**, with a second quadrant division dividing the mapping area into 16 equal 126°×128° mapping areas.

FIG. **16** is the same plan view illustration of FIG. **15**, with a third quadrant division dividing the mapping area into 64 equal 64°×64° mapping areas.

FIG. **17** is the same plan view illustration of FIG. **16**, with a fourth quadrant division dividing the mapping area into 256 equal 32°×32° mapping areas.

FIG. **18** is the same plan view illustration of FIG. **17**, with a fifth quadrant division dividing the mapping area into 1024 equal 16°×16° mapping areas.

FIG. **19** is the same plan view illustration of FIG. **18**, with a sixth quadrant division dividing the mapping area into 4096 equal 8°×8° mapping areas.

FIG. **20** is an illustration showing the application of polar compression at the 8th level or magnitude of resolution.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS OF THE INVENTION

Before turning to the detailed description of the preferred embodiments of the invention, it should be noted that the map illustrations used throughout the drawings are only crude approximations which are only being used to illustrate important features and aspects and the operation of the present invention; therefore. the geographical political and cultural outlines may very well differ from actual outlines.

FIG. **1** is a crude representation of what the earth's surface would look like if it were laid flat and viewed from a "relative viewing position" which is a great distance in space. Shown as vertical lines are: **10**, corresponding to the 0° meridian extending through Greenwich, England; **20**, corresponding to the 180° west meridian: and, **30**, corresponding to the 180° east meridian. Shown as horizontal lines are: **40**, corresponding to the equator: **50**, corresponding to 90° north (i.e.. the north pole): and **60**, corresponding to 90° south (i.e.. the south pole).

Note that at this "relative viewing position", not much detail as to cultural features is seen; i.e., all that is seen is the general outline of the main geographical masses of the continents.

The present invention seeks to provide a low cost and efficient computer-based mapping method and system having a unique approach for arranging and accessing a digital mapping database of unlimited size, i.e., a mapping method and system which can manipulate and access a data base having sufficient data to allow the mapping system to reproduce digital maps of any geographical area with different degrees of resolution. This can be most easily understood by viewing FIG. **2** and FIGS. **3A–F**.

Because of the overwhelming advantages over the paper and analog mapping approaches, the digital mapping approach is utilized with the present invention; thus, there is shown in FIG. **2**, a digital computer **200**, having a disk or hard drive **280**, a monitor **210**, a keyboard **220** (having a cursor control portion **230**), and a mouse device **240**. As mentioned previously, in a digital mapping approach, mapping information is stored in a format similar to the text of other databases, i.e., the outstanding map features are stored as a list of objects to be drawn, each object being defined by a plurality of vector "dot" coordinates which define the crude outline of the object. (Note: the reproduction of a digital map from a list of objects and "dot" vectors is well known the art, and is not the subject matter of the present invention; instead, the invention relates to a unique

method and system for storing and accessing the list of objects and "dot" vectors contained in a tremendous digital data base.)

Once a geographical map has been "digitized",—i.e., converted to a list of objects to be drawn and a plurality of vector "dot" coordinates which define the crude outline of the object —, the mapping database must be stored in the memory of a mass storage device. Thus, the digital computer **200**, which is to be used with the mapping method and system of the the present invention, is shown associated with the magnetic disk **260** (which represents any well-known magnetic mass storage medium, e.g., floppy disks, hard disks. magnetic tape, etc.), and the CD-ROM 270 (which represents any well-known optical storage medium, e.g. a laser-read compact disk). Alternatively, the digital mapping database can be stored on, and the digital computer can be associated with any well known electronic mass storage memory medium (e.g., ROM, RAM, etc.). Because of every increasing availability. reductions in cost, and tremendous storage capacities, the preferred memory mass storage medium is the CD-ROM, i.e., a laser-read compact disk.

The discussion now turns to FIGS. **3A–F**, showing illustrations of monitor displays which provide a brief illustration of the operation of the present invention. Although the digital nature of the maps of FIGS. **3A–3F** can easily be detected due to the jagged outlines, it should be understood that these geographical outlines could easily be smoothed using any of a number of "smoothing" techniques which are well-known to those skilled in the digital mapping art.

In FIG. **3A**. the digital computer has retrieved relevant mapping information from the digital mapping database, and has produced a monitor display of a digital map substantially corresponding to the flat projection of the earth's surface which was shown in FIG. **1**. In FIG. **3A**, the monitor display reflects a "relative viewing position" which is a great distance in space, and hence, only the crude geographical outline of the continents is shown with sparse detail.

Suppose a user wishes to view a map of the states of Virginia and Maryland in greater detail. By entering the appropriate commands using the keyboard **220** or the mouse device **240**, a user can cause the monitor display to "zoom" to a lower "relative viewing position", such that the monitor displays a digital map of a smaller geographical area which is shown at a higher degree of resolution. Thus, in FIG. **3B** the a digital map of the continents of the western hemisphere is displayed in greater detail.

By entering additional commands, a user can cause the monitor display to further "zoom" to the following displays: FIG. **3C** showing North America in greater detail; FIG. **3D** showing the eastern half of the United States in greater detail: FIG. **3E** showing the east coast of the United States in greater detail; and. FIG. **3F** showing Virginia and Maryland in greater detail.

Although in this example, the monitor display was caused to "zoom" to Virginia and Maryland, it should, be appreciated that the present invention allowed a user to selectively zoom into any geographical area of the earth, and once a user has reached the desired degree of mapping resolution, the mapping system of the present invention also allows the user to "scroll" or "fly" to a different lateral position on the map.

Furthermore, although the drawings illustrate the monitor display zooming to display state boundaries, and features, it should be further appreciated that the present invention is by no means limited to this degree of resolution. In fact, the degree of resolution capable with the present invention will be shown to be limited only by the operating system of the digital computer **200** with which the present invention is used. In one demonstration, the monitor display has been shown to be able to zoom to resolution where the outlines of streets were displayed. Even further degrees of resolution are possible as will be more fully understood after the discussions below.

In digitally mapping a large geographical area (e.g., the earth) in detail, —especially in the degree of resolution mentioned above —, one should be able to appreciate that the storage of digital mapping data sufficient to accurately define all the geographical, political and cultural features would represent a tremendous digital mapping database. In order to provide a low cost mapping system having quick access time and allowing a high degree of resolution, what is needed is a mapping system having an effective approach for arranging an accessing the digital database. Prior art mapping systems have been deficient in this regard.

The mapping system of the present invention utilizes a new and extremely effective approach, which can be most easily understood using the following simplified example.

In FIG. **4**, there is shown the cross-section of a building **400**, with a square hole **410** (shown in cross-section) cut through the third level floor **420**. with a larger square hole **430** (shown in cross-section) cut in the second level floor **440**, and with a large square piece of paper **450** (shown in cross-section) laid out on the first level floor **460**. Suppose it was desired to build up a digital data base which could be used to reproduce a digital map of the paper **450** with varying degrees of resolution.

First, one would take the "relative viewing position" A, and view the paper **450** through the square hole **410** in the third level floor **420**. At this level. the paper **450** appears small (FIG. **5A**), and the degree of resolution is such that the message appears only as a series of dots. In order to build up a digital mapping database, the visual perception (FIG. **5A**) is imagined to be divided into four equal quadrants a, b, c, d (FIG. **5B**), and visual features appearing in each respective area is digitized and stored in a separate database file. Thus, four separate database files can be utilized to reproduce a digital map of the paper **450** as viewed from position A (FIG. **4**).

In order to digitize and record data corresponding to a second (or higher) degree of resolution, the next "relative viewing position" B (FIG. **4**) is taken to view the paper **450** through the square hole **430**. At this level, the paper **450** appears larger (FIG. **6**), and the degree of resolution is such that the message now appears as a series of lines. At this second level, the map is imagined as being divided into four times as many areas as the first imaginary division, and then, the visual information contained within each area is digitized and stored in a separate database file. Thus, 16 files can be used to reproduce a digital map of the paper **450**, as viewed from the relative viewing position B (FIG. **4**).

In order to digitize and record data corresponding to a third (or higher) degree of resolution. the next "relative viewing position" C (FIG. **4**) is taken to view the paper **450**. At this level, paper **450** now appears larger (FIG. **7**) and has visual features of higher resolution.

The paper **450** is imagined as being divided into four times as many areas as the second imaginary division, and the visual information is digitized and stored. Thus, 64 files could be used to reproduce a digital map of the paper **450**, as viewed from the relative viewing position C (FIG. **4**).

Once digital data has been entered for the above three "relative viewing positions" A, B, C (FIG. **4**), the digital mapping database contains 4+16+64 or 84 files which can be conceptually envisioned as being arranged in a pyramid structure as shown in FIG. **8**. In order to allow a user to selectively display any desired map section at the desired degree of resolution, the digital computer **200** must be able to know which of the 84 files to access such that the appropriate mapping data can be obtained. The present invention accomplishes this by conceptually arranging the files in a pyramidal structure, and assigning a file name to each file which is related both to the file's position and ancestry within the pyramidal structure. This can be more specifically described as follows:

A file's ancestry can be explained using the illustrations of FIGS. **5B**, **6** and **7**. In FIG. **5B**, the paper **450**, as viewed from "relative viewing position" A (FIG. **4**), is subjected to an imaginary division into four quadrants a, b, c, and d. Quadrants a, b, c, d are related to one another in the sense that it takes all four areas to represent the paper **450**: hence quadrants a, b, c, d can be termed as brothers and sisters.

FIG. **6** is an illustration of the paper **450** as it appears from the relative viewing position B (FIG. **4**). with the paper **450** being subjected to an imaginary division into 16 areas. Note that the areas e, f, g, h (FIG. **6**) represent the same area of paper **450** as the quadrant a (FIG. **5B**). In effect, quadrant a has been enlarged (to show a higher degree of resolution) and divided into quadrants e, f, g, h. Thus, it can be said that quadrant a (FIG. **5B**) is the parent, and that quadrants e, f, g, h (FIG. **6**) are brothers and sisters and the offspring of ancestor a. Similar discussions can be made for quadrants b, c and d and the remaining area of FIG. **6**.

FIG. **7** is an illustration of the paper **450** as it appears from the relative viewing position C (FIG. **4**). with the paper **450** being subjected to an imaginary division into 64 areas. In a manner similar to the discussion above, note that areas s, t, w, x (FIG. **7**) represent the same area of paper **450** as the quadrant h (FIG. **6**). In effect, quadrant h has been enlarged (to show a higher degree of resolution) and divided into quadrants s, t, w, x. Thus, it can be said that quadrant a (FIG. **5B**) is the grandparent, quadrant h (FIG. **6**) is the parent, and quadrants s, t, w, x (FIG. **7**) are the brothers and sisters and offspring of ancestors a and h.

As described previously, once FIGS. **5B**, **6** and **7** are subjected to the imaginary divisions, the visual information in each area (or quadrant) is digitized and stored in a separate file. The 84 resulting files can be conceptually envisioned as the pyramidal structure shown in FIG. **8**. In FIG. **8**, dashed lines are utilized to show the lineage of the files just discussed.

FIG. **8** is further exemplary of one file naming operation which can be utilized with the present invention.

At the top of the pyramidal structure (FIG. **8**). each of the four quadrant files is arbitrarily assigned a different character. A, B, C, D, (Note: The characters assigned are not critical with regard to the invention and hence it should be noted that any characters can be assigned, e.g., **0,1,2,3**, etc.)

In moving down one level in the pyramidal structure,, the filenames for each of the respective files on the second level is increased to two characters.

In calculating the filenames, it is convenient to first divide the second level files into groups of four, according to parentage. To maintain a record of ancestry, the ancestor filename of each file is maintained as the first part of the filename. In determining the second part, the naming protocol which was utilized to name the quadrant files of the top level, is also utilized in naming the respective quadrant files on the second level. Thus, parent file A is shown as being related to descendent (i.e., brother and sister) files AA, AB, AC, AD. Similar discussion can be made for the remaining files along these two level.

A similar process can be utilized in providing the unique filenames to the third level files. At this level, the filenames consist of three characters. Again, the ancestor filename of each file would be maintained as a first filename part, in order to maintain a record of ancestry. In the example illustrated (FIG. **8**), parent file AD is shown as being related to descendent (i.e., brother and sister) files ADA, ADB, ADC, ADD. Similar discussions can be made for the remaining files along these two levels, and furthermore, similar discussions can be made each time a pyramidal level is added.

From the above discussion, one should be able to realize that the above-described naming convention is particularly useful in programming a digital computer to move through the pyramidal file structure to access the appropriate data corresponding to varying degrees of resolution. More particularly, one should be able to realize that, since file names increase one character in length each time there is a downward movement through the pyramidal structure and the protocol for naming descendent files is known, the digital computer can be programmed to quickly and easily access the appropriate files for a smaller mapping area with a greater degree of resolution. Similarly, one should be able to realize that, since the filenames decrease one character in length each time there is an upward movement through the pyramidal structure, the digital computer can be programmed to quickly and easily access the appropriate files for a greater mapping area with a smaller degree of resolution.

The following example is believed to provide an increase in the understanding of the present invention.

In the example, it is assumed that the digital database corresponding to the three resolutions of the paper **450** (as shown in FIGS. **4**, **5A–B**, **6**, **7**) have been loaded to be accessible from the memory mass storage device, and furthermore, it is assumed that the mapping system is programmed to initially access and display a digital map corresponding to the digital mapping data in the files A, B, C, D (FIG. **8**). Thus, the monitor (FIG. **9B**) would display (in low resolution) the entire area enclosed within dashed portion **900** illustrated on the paper **450** (FIG. **9A**). (Note: The reproduction of a digital map from digital data from several different files or sources is well-known in the art and is not the subject matter of the present invention.)

Suppose the user notices the dotted area on the low resolution map and wishes to investigate this area further. By using the appropriate keys (e.g. ↖, ↗, ↙, ↘) and/or a mouse device, a user can give the mapping system an indication that he/she wishes to see the smaller area (i.e., quadrant A) at a higher degree of resolution. Upon receiving this preference, the mapping

system can use its knowledge of the file naming operations to quickly determine the names of the files which must be accessed. More specifically, using A as the parent file name and following the existing quadrant naming protocol the mapping system is quickly and easily able to calculate that it is files AA, AB. AC, AD which it needs to access. Once these files are accessed, the monitor in FIG. **10B** displays (in higher resolution) the area enclosed within the dashed portion **1000** as illustrated on the paper **450** (FIG. **10A**).

If a user is still not satisfied with the degree of mapping resolution, the user can again use the appropriate keys or mouse device to indicate that he/she wishes to see the smaller area (e.g., quadrant D; FIG. **10A**) in a higher degree of resolution. In using AD as the parent filename and following the existing quadrant naming protocol, the mapping system is quickly and easily able to calculate that it is files ADA, ADB, ADC, ADD which it needs to access. Once these files are accessed, the monitor (FIG. **11B**) displays (in higher resolution), the area enclosed within the dashed portion **1100** as illustrated on the paper **450** (FIG. **11A**).

One skilled in the digital mapping and computer programming art should recognize that "scrolling" or "flying" to different lateral "relative viewing positions" to display a different lateral portion of the map is also provided by the present invention. Instead of adding or removing filename characters as in a change of resolution, in this instance, the mapping system must be programmed to keep track of the filenames of the current position and also, the orderly arrangement of filenames so that the appropriate filenames corresponding to the desired lateral position can be determined. As an example if the user desired to scroll to the right border of the paper **450**, the mapping system would respond by accessing and causing the monitor to display the digital maps corresponding to the following sequence of files: (Note: In this example, it is assumed that it takes 4 files to provide sufficient digital data to display a full digital map on a monitor) ADA, ADB, ADC, ADD; ADB, ADD, BCA, BCC; BCA, BCB, BCC, BCD; BCB, BCD, BDA, BDC; and BDA, BDB, BDC, BDD. If the user, then desired to scroll to the bottom (right corner) of the paper **450**, the mapping system would respond by accessing and causing the monitor to display the digital maps corresponding to the following files: BDA, BDB, BDC, BDD; BDC, BDD, DBA, DBB; DBA, DBB, DBC. DBD; DBC, DBD, DDH, DDB; DDA, DDB, DDC, DDD. In effect as all of the files in the above example correspond to the same level of resolution all these files (and any group of files which exist on the same level of resolution) can be taken as being related as cousins.

FIGS. **9A**, **10A**, **11A** can also be used to illustrate the operation of moving toward the display of a larger mapping area with a lower degree of resolution.

Assume that after lateral "scrolling" or "flying", that the monitor is now displaying (not shown) a digital map corresponding to the enclosed area **1110** shown in FIG. **11A**. (Note: at this position the mapping system is accessing and display a digital map corresponding to the digital data in the files DCA, DCB, DCC, DCD). Suppose the user now wishes to cause the "relative viewing position" to zoom upward, such that the monitor will display a larger portion of the paper **450** at a lower degree of resolution. By using the appropriate keys or a mouse device, the user indicates his/her preference to the mapping system. Upon receiving this preference, the mapping system is programmed to quickly determine the names of the files which must be accessed. More specifically, the mapping system is able to look at the first portion of the filenames currently being used (i.e., DCA, DCB, DCC, DCD), to immediately determine that these files have the ancestry DC, i.e., have a grandfather D and a parent DC. The mapping system then immediately determines brother and sister files of parent file DC as being DA, DB and DD. The mapping system then accesses these files and causes the monitor to display a digital map (not shown) corresponding to the enclosed portion **1010** (FIG. **10A**) of the paper **450**.

Suppose the user again indicate a preference to cause the "relative viewing position" zoom upward. Upon receiving this preference, the mapping system again goes through a process similar to that discussed immediately above. However, this time the mapping system looks at the filenames currently being used (i.e., DA, DB, DC, DD) and determines that parent file D has brother and sister files A, B and C. The mapping system then immediately accesses these files and causes the monitor to display a digital map (FIG. **9B**) corresponding to the enclosed portion **900** (FIG. **9A**) of the paper **450**.

The text now turns to a description of the operation for assigning unique filenames in the currently preferred embodiment, i.e., in a digital mapping system which is implemented in a DOS operating system.

As anyone skilled in the computer art will know. every computer operating system has its own unique set of rules which must be followed. In an implementation of the present invention in a DOS operating system. the DOS rules must be followed. Since a critical feature of the present invention is the division of the digital mapping database into a plurality of files (each having a unique filename), of particular concern with the present invention is the DOS rules regarding the naming of filenames.

A DOS filename may be up to eight (8) characters long, and furthermore. may contain three (3) additional trailing characters which can represent a file specification. Thus, a valid DOS filename can be represented by the following form:

---

where "–" can be replaced by any ASCII character (including blanks), except for the following ASCII characters:

. " / \ [ ] : ¦ < > + ; ,

and ASCII characters below **20H**. The currently preferred embodiment stays within these DOS filename rules by using the file naming operations which are detailed below.

Because the assigned filenames will be seen to be related to hexadecimals, a useful chart containing the hexadecimal base and also a conversion list (which will be shown to be convenient ahead), is reproduced below:

Column 1	Column 2	Column 3
0000	0	G
0001	1	H
0010	2	I
0011	3	J
0100	4	K
0101	5	L
0110	6	M

-continued

Column 1	Column 2	Column 3
0111	7	N
1000	8	O
1001	9	P
1010	A	Q
1011	B	R
1100	C	S
1101	D	T
1110	E	U
1111	F	V

The first column contains a list of all the possible 4-bit binary combinations: the second column contains the hexadecimal equivalent of these binary numbers: and the third column concerns a "mutant-hex" conversions which will be shown to be important in the discussion to follow. In the operatidns to assign unique filenames for use in a DOS operating system, the present invention looks at each of the eight DOS filename characters as hexadecimal characters rather than ASCII characters. Hence, while the following discussion will center around determining unique filenames using hexadecimal (and "mutant-hexadecimal") characters, it should be understood in an actual DOS implementation, the hexadecimal filenames must be further converted into the equivalent ASCII characters such that the appropriate DOS file naming rules are followed.

At this point, it is also useful to note that the file naming operation of the preferred embodiment is not concerned with the trailing three character filename extension. However, it should be further noted that this three character filename extension may prove useful in specifying data from different sources, and allowing the different types of data to reside in the same database. As examples, the filename extension ".spm" might specify data from scanned paper maps, the filename extension ".si" might specify data from satellite imagery, the filename extension ".ged" might specify gridded elevation data, etc.

As a result of the foregoing and following discussions, it will be seen that the naming operation of the preferred embodiment is concerned only with a filename of the following form:

-

where each "–" represents a character which is a hexadecimal character within the character set of "0–9" and "A–F", or is a "mutant-hexadecimal" character within the character set of "G–V".

Several more important file naming details should be discussed.

First, it should be pointed out that the first four (4) filename characters is designated as corresponding to the "x" coordinate characters, and the last four (4) filename characters are designated as corresponding to the "y" coordinate characters.

Second, during the file naming operations, often it is necessary to convert the filename characters into the equivalent binary representation. As each hexadecimal character can be converted into a four bit binary number, it can be seen that the first four (4) filename characters (designated as "x" coordinate characters) can be converted into sixteen (16) binary bits designated as "x" bits, and similarly, that the last four (4) filename characters (designated as "y" coordinate characters) can be converted into sixteen (16) binary bits designated as "y" bits. As will become more apparent ahead, each of these sixteen (16) "x" and "y" bits corresponds to a filename bit which can be manipulated when assigning filenames at a corresponding magnitude or level of mapping resolution, e.g., the first "x" and first "y" bits correspond to filename bits which can be manipulated when assigning unique filenames at the first magnitude, the second "x" and second "y" bits correspond to filename bits which can be manipulated when assigning unique filenames at the second magnitude, etc.

Third, FIG. **12** corresponds to the naming protocols which are utilized to modify and relate a parent filename to four (4) quadrant filenames. Note that there is a two-bit naming protocol in each of the quadrant files. As will become more clear ahead, the first bit of each protocol determines whether the current "x" filename bit will be modified (i.e., if the first protocol bit is a "1", the current "x" filename bit is changed to a "1", and if first protocol bit is a "0", the current "x" filename bit is maintained as a "0"), and the second bit determines whether the current "y" filename bit will be modified (in a similar manner).

The text now turns to a file naming example which is believed to provide further teachings and clarity to the currently preferred file naming operation.

FIG. **13** is an illustration of a portion of the preferred digital data base, with the plurality of files (partially shown) being arranged in a conceptual pyramidal manner in a manner similar to that which was described with reference to FIG. **8**. More specifically, there are shown four files **1300** having digital data corresponding to a first level or magnitude of mapping resolution, sixteen files **1310** having digital data corresponding to a second level or magnitude of mapping resolution, sixty-four files **1320** having digital data corresponding to a third level or magnitude of mapping resolution, and a partial cut-away of a plurality of files **1330** having data corresponding to a fourth level or magnitude of mapping resolution. Although not shown, it is to be understood that, in the preferred embodiment, additional pyramidal structure corresponding to levles magnitudes five through sixteen similarly exist. As examples of the file naming operation, filenames will now be calculated for the files which essentially occupy the same positions as the files which were outlined in FIG. **8**.

We begin with the initializing eight (8) character filename:

0	0	0	0	0	0	0	0

which can be converted to the binary equivalent:

0000	0000	0000	0000	0000	0000	0000	0000

This binary representation is the basic foundation which will be used to calculate all of the filenames for the files on the first level (**1300**). Note, that the first and last four filename characters, and the first and last sixteen bits are slightly separated in order to conveniently distinguish the "x" and "y" coordinate characters and bits. Both the first (leftmost) "x" bit and the first (leftmost) "y" bit are the bits which can be manipulated in assigning a unique filename to the files on the first level.

File naming begins with the first (upper-rightmost) file on the first level **1300**. The naming protocol assigned to this quadrant file is the two-bit protocol "10".

As the first protocol bit is a "1", this means that the current "x" bit must be changed to a "1". As the second protocol bit is a "0", this means that the current "y" bit is maintained as a "0". As a result of the foregoing, the first (upper-rightmost) file is assigned the filename having the binary equivalent of:

1000	0000	0000	0000	0000	0000	0000	0000

which can be converted to the hex characters:

8	0	0	0	0	0	0	0.

In proceeding clockwise, next is the second (lower-rightmost) file on the first level **1300**. The naming protocol assigned to this quadrant file is the two-bit protocol "11". As the first protocol bit is a "1", the current "x" bit is changed to a "1": similarly, as the second protocol bit is a "1", the current "y" bit is changed to a "1". As a result of the foregoing, the second (lower-rightmost) file is assigned the filename having the binary equivalent of:

1000	0000	0000	0000	1000	0000	0000	0000

which can be converted to the hex characters:

8	0	0	0	8	0	0	0.

Continuing clockwise, next is the third (lower-leftmost) file on the first level **1300**. The naming protocol assigned to this quadrant file is the two-bit protocol "01". As the first protocol bit is a "0", the current "x" bit is maintained at 0. As the second protocol bit is a "1", the current "y" bit is changed to a "1". As a result of the foregoing. the third (lower-leftmost) file is assigned the filename having the binary equivalent of:

0000	0000	0000	0000	1000	0000	0000	0000

which can be converted to the hex characters:

0	0	0	0	8	0	0	0.

Finally, there is the fourth (upper-leftmost) file on the first level **1300**. The naming protocol assigned to this quadrant is the two-bit protocol "00". As neither of the protocol bits is a "1", it can be easily seen that neither of the current "x" and "y" bits changes, and hence, the fourth (upper-leftmost) file is assigned the filename having the binary equivalent of:

0000	0000	0000	0000	0000	0000	0000	0000

which can be converted to the hex characters:

0	0	0	0	0	0	0	0.

In further discussions of the example, it is important to note that the initializing (8) character filename of 0000 0000 (which was utilized to calculate the filenames of the files on the first level 1300) is not utilized in assigning filenames on subsequent levels. In naming files from the second level or magnitude downward, the binary equivalent of the parent file's name is utilized as the foundation from which the descendent file's name is derived. It is only coincidental that the filename of the parent file 00000000 (located in the user-left most corner of the first level **1300**) is the same as the initializing filename. Use of the parent's filename to calculate the descendent's filename will become more readily apparent ahead in the example.

In continuing the file naming example, the fourth (upper-leftmost) file (having filename 00000000) in the first level **1300** can be viewed as being the parent file of the four (highlighted) quadrant files in the second level **1310**. As stated above, the binary equivalent of parent file's 00000000 name is utilized as the foundation for calculating the descendent file's filenames. At this second level or magnitude, the second "x" and "y" bits from the left in the parent's binary filename are taken as the "current" bits which can be manipulated to provide a unique filename for the descendent files.

As the calculation of the filename for the fourth (upper-leftmost) file of the second level **1310** illustrates a very important modification in the file naming operation, the example will first continue with discussions corresponding to this file.

As the naming protocol assigned to the fourth (upper-leftmost) file of the second level **1310** is two-bit protocol "00", it can be seen that neither of the current "x" and "y" bit would be changed. Hence the parent's filename 00000000 is unchanged, and is attempted to be adopted as the descendent's filename. However, note that this is extremely undesirable as the operation of the present invention is based on assigning each data file a unique filename, and furthermore, a DOS operation system will not allow the same filename to be assigned to two different files. To avoid this clash. the preferred file naming operation of the present invention incorporates a further step which can be detailed as follows:

First calculate the filename as explained above. Once the binary filename is obtained, convert to the eight character hexadecimal equivalent.

Next, take the decimal number of the current level or magnitude and subtract one (1) to result in a decimal magnitude modifier. Convert the decimal magnitude modifier into a four-bit binary magnitude modifier, and line these four bits up with the four hexadecimal "x" filename characters. Whenever a "1" appears in the binary magnitude modifier. the corresponding aligned "x" filename character is converted to a "mutant-hexadecimal" character. i.e., a decimal 16 value is added to convert the aligned filename character into a one of the "mutant-hexadecimal" characters in the character set of "G–V".

Conversions from a hexadecimal character to a "mutant-hexadecimal" character can be most readily made using the chart detailed above. As an example, if decimal 16 is added to the hex character "0" (Column 2), there is a conversion to the "mutant-hexadecimal" character "G" (Column 3). Similarly, if decimal 16 is added to the hex character "1" (Column 2), there is a conversion to the "mutant-hexadecimal" character "H" (Column 3). Similar discussion can be made for the remaining hex and "mutant-hexadecimal characters in the chart.

Once correspondingly aligned filename characters are converted to "mutant-hexadecimal", the resultant

eight (8) characters correspond to the file's unique filename.

The above processing will now be applied to the fourth (upper-rightmost) file of the second level **1310** (which was recently discussed above). The resultant binary filename:

0000	0000	0000	0000	0000	0000	0000	0000

is converted to the hex characters:

0	0	0	0	0	0	0	0.

The level or magnitude two (2) minus one (1) results in a decimal magnitude modifier of one (1). The decimal magnitude modifier is converted to the four-bit binary equivalent and is aligned with the "x" filename characters above, as follows:

0	0	0	1

Only the fourth bit of the binary magnitude modifier is a "1", so only the fourth "x" filename character needs to be converted to "mutant-hexadecimal". From the chart, the hexadecimal character "0" is shown to convert to a "mutant-hexadecimal" character "G". Thus. the unique filename which is assigned to the fourth (upper-leftmost) file of the second level **1310**, is:

0	0	0	G	0	0	0	0.

In continuing the example to calculate the filename for the first (upper-right-quadrant) file of the second level **1310**. it can be seen that this file is assigned the two-bit naming protocol "10". The first protocol bit is a "1" which indicates that the current (second from the left) "x" bit of the parent file's binary filename must be changed to a "1", In contrast, the second protocol bit is a "0", which indicates that the current (second from the left) "y" bit is maintained as "0" Thus the parent filename:

0000	0000	0000	0000	0000	0000	0000	0000

is converted to:

0100	0000	0000	0000	0000	0000	0000	0000

which results in the hex characters:

4	0	0	0	0	0	0	0.

The level or magnitude two (2) minus one (1) results in a decimal magnitude modifier of one (1). The decimal magnitude modifier is converted to the four-bit binary equivalent and is aligned with the "x" filename characters above, as follows:

0	0	0	1

Only the fourth bit of the binary magnitude modifier is a "1", so only the fourth "x" filename character needs to be converted to "mutant-hexadecimal". From the chart, the hexadecimal character "0" is shown to convert to a "mutant-hexadecimal" character "G". Thus, the unique filename which is assigned to the first (upper-right-quadrant) file of the second level **1310**, is:

4	0	0	G	0	0	0	0.

Turning now to the second (lower-right-quadrant) file, this file is assigned the two-bit naming protocol "11". The first protocol bit is a "1" which indicates that the current (second from the left) "x" bit of the parent file's binary filename must be changed to a "1", and similarly, the second protocol bit is a "1", which indicates that the current (second from the left) "y" bit of the parent file's binary filename must be changed to a "1" Thus the parent filename:

0000	0000	0000	0000	0000	0000	0000	0000

is converted to:

0100	0000	0000	0000	0100	0000	0000	0000

which results in the hex characters:

4	0	0	0	4	0	0	0.

The level or magnitude two (2) minus one (1) results in a decimal magnitude modifier of one (1). The decimal magnitude modifier is converted to the four-bit binary equivalent and is aligned with the "x" filename characters above, as follows:

0	0	0	1

Only the fourth bit of the binary magnitude modifier is a "1", so only the fourth "x" filename character needs to be converted to "mutant-hexadecimal". From the chart, the hexadecimal character "0" is shown to convert to a "mutant-hexadecimal" character "G". Thus, the unique filename which is assigned to the second (lower-right quadrant) file of the second level **1310**, is:

4	0	0	G	4	0	0	0.

In applying the above operations to the third (lower-left-quadrant) file of the second level **1310**, it can be easily calculated that the resultant filename is:

0	0	0	G	4	0	0	0.

The example of the file naming operation is further extended to the third level or magnitude. as this example is illustrative of both the use of the parent file's binary filename to calculate the descendent's filename, and the removal of "mutant-hexadecimal" conversions before calculating the descendent's filename.

In FIG. 13. the third (lower-right-quadrant) file of the second level **1310** is shown as being the parent of the four (4) quadrant files highlighted in the third level or magnitude **1320**.

The discussion centers on the calculation of the unique filename for the second (lower-right-quadrant) file in the third level **1320**. Before the parent filename can be used as the foundation for calculating the descendent's filename. all "mutant-hexadecimal" conversions must be removed. Thus the parent filename:

4	0	0	G	4	0	0	0.

is converted back to:

4	0	0	0	4	0	0	0.

which is further converted to the binary equivalent:

0100	0000	0000	0000	0100	0000	0000	0000

In continuing the calculation, this second (lower-right-quadrant) file is assigned the two-bit naming protocol "11". The first protocol bit is a "1" which indicates that the current (third from the left) "x" bit of the parent file's binary filename must be changed to a "1", and similarly, the second protocol bit is a "1", which indicates that the current (third from the left) "y" bit of the parent file's binary filename must be changed to a "1". Thus the parent filename:

0100	0000	0000	0000	0100	0000	0000	0000

is converted to:

0110	0000	0000	0000	0110	0000	0000	0000

which results in the hex characters:

6	0	0	0	6	0	0	0.

The level or magnitude three (3) minus one (1) results in a decimal magnitude modifier of two (2). The decimal magnitude modifier is converted to the four-bit binary equivalent and is aligned with the "x" filename characters above, as follows:

0	0	1	0

Only the third bit of the binary magnitude modifier is a "1", so only the third "x" filename character needs to be converted to "mutant-hexadecimal". From the chart, the hexadecimal character "**0**" is shown to convert to a "mutant-hexadecimal" character "G". Thus, the unique filename which is assigned to the second (lower-right-quadrant) file of the third level **1320**, is:

6	0	G	0	6	0	0	0.

The filenames for several additional third level files will be given to give the patent reader further practice.

In applying the above operations to the first (upper-right-quadrant) file of the third level **1320**, it can be easily calculated that the resultant filename is:

6	0	G	0	4	0	0	0.

In applying the above operations to the third (lower-left-quadrant) file of the third level **1320**, it can be easily calculated that the resultant filename is:

4	0	G	0	6	0	0	0.

Finally, in applying the above operations to the fourth (upper-left-quadrant) file of the third level **1320**, it can be easily calculated that the resultant filename is:

4	0	G	0	4	0	0	0.

As a result of all of the foregoing teachings, one skilled in the art should now be able to calculate the filename of any other of the 1.4 billion files which would be required to provide digital maps corresponding to sixteen (16) resolutions of any geographical area on earth. Furthermore, once a file is being accessed, by understanding the rules and operations of the file naming operation one skilled in the are should be able to calculate any other related files, i.e., parent files. and brother/sister/cousin files.

While the unique approach for storing and accessing files in the pyramidal file structure has been particularly pointed out. further discussion is needed as to an additional advantageous feature of the present invention.

As mentioned previously, the creation of a digital database is a very tedious, time consuming and expensive process. Tremendous bodies of mapping data are available from many important mapping authorities, for example, the U.S. Geological Survey (USGS), Defense Mapping Agency (DMA), National Aeronautics and Space Administration (NASA), etc.

The maps and mapping information produced by the above recited agencies, is always based on well established mapping area divisions. As a few examples, the Defense Mapping Agency (DMA) produces maps and mapping information based on the following mapping areas: GNC maps which are $2°\times2°$: JNC maps which are $1°\times1°$; ONC maps which are $30'\times30'$: TPC maps which are $15'\times15'$; and JOG maps which are $7.5'\times7.5'$. As a further example, the U.S. Geological Survey (USGS) also produces maps and utilizes mapping information based on $15'\times15'$ and $7.5'\times7.5'$.

In terms of both being able to easily utilize the mapping data produced by these agencies, and represent an attractive mapping system to these mapping agencies, it would be highly desirable for the mapping system of the present invention to be compatible with all of the mapping formats used by these respective agencies. Such is not the case when the mapping database is based on a graticule system corresponding to 360°

If one were to apply multiple quadrant divisions to the $360°\times180°$ flat map projection of the earth (FIG. 1). one would result in the following mapping area subdivisions:

Level of quadrant div.:	Resultant mapping area:
1	(4) 180° × 90°
2	(16) 90° × 45°
3	(64) 45° × 22.5°
4	(256) 22.5° × 11.25°
5	(1024) 11.25° × 5.625°
etc.	

Note that these mapping area subdivisions are very awkward, and do not match any of the well settled mapping area subdivisions. (It should be further noted that no better results are obtained if the initial map projection is imagined as being a 360°×360° square instead of a rectangle.)

In order to avoid these awkward mapping subdivisions, and result in quadrant divisions which precisely match widely used mapping area subdivisions, the present invention utilizes a unique initial map projection.

More specifically, as can be seen in FIG. **14**, the present invention initially begins with a unique 512°×512° initial map projection. Shown centered in the 512°×512° map projection is the now familiar 360°×180° flat projection of the surface of the earth. Although the 512°×512° projection initially appears awkward and a waste of map projection space, the great advantages which are resultant from the use of this projection will become more apparent in the discussions to follow.

To aid in this discussion, provided on the next page is a chart which details these important advantages as well as other useful information regarding the use of this map projection.

less complicated, the non-DOS file naming operation will be used in the discussion.

The digital mapping of the earth surfaces begins in FIG. **14**. The visual perception of the earth surfaces is experienced as being centered, and occupying only a portion of the 512°×512° projection. A first quadrant division is applied to result in four equal 256°×256° mapping areas. The visual information in each of the areas is digitized, and stored in a separate file, Thus, it can be seen that one would have to access four files a, b, c, d in order to reproduce a digital map corresponding to the earth surfaces as viewed from this "relative viewing position."

One skilled in the art, might, at this point, wonder if the massive blank portions of the 512°×512° projections result in large blank portions on the digital map display. The preferred embodiment avoid this phenomena, through a simple watchdog operation, i.e., the computer is programmed to keep track of longitudinal and latitudinal movements from an initial position of 0° longitude and 0° latitude, and the computer does not allow scrolling of the monitor display beyond 90° north or south.

As to side to side movements, the computer allows scrolling beyond 180° east or west by patching the appropriate data files together to perform a "wrap around" operation. Note that, with the knowledge of the logical file naming operation, the computer can quickly and easily calculate the appropriate files to access.

Before moving to the next level or magnitude of mapping resolution, it is beneficial to note the correspondence between our findings and the enties in the above-indicated chart.

MAGNITUDE EQUIVALENCY CHART FOR DELORME PROJECTION
Chart assumes 69 statute miles per degree at equator

MAG-NI-TUDE	Window Size without overlap	Ht of window statute miles	Ht of window kilometers	# Windows per MAG	# Windows/ MAG w/polar com-pression	Pixel resolution 480 monitor (ft)	Data reso-lution (ft) 1024-based window	Equivalent Paper Map Scales	Size of paper map image at equator (in)
1	256° × 256°	17664	28421	4	4		91080		
2	128° × 128°	8832	14211	8	8		45540		
3	64° × 64°	4416	7105	24	24	48576	22770	1:100 million	2.8 × 2.8
4	32° × 32°	2208	3553	72	72	24288	11385	1:50 million	2.8 × 2.8
5	16° × 16°	1104	1776	288	288	12144	5693	1:30 million	2.3 × 2.3
6	8° × 8°	552	888	1152	858	6072	2846	1:16 million	2.2 × 2.2
7	4° × 4°	276	444	4232	3432	3036	1423	1:10 million	1.7 × 1.7
8	2° × 2°	138	222	16200	12808	1518	712	1:5 million	1.7 × 1.7
9	1° × 1°	69	111	64800	51210	759	356	1:2 million	2.2 × 2.2
10	30′ × 30′	34.5	55.5	259000	204840	380	178	1:1 million	2.2 × 2.2
11	15′ × 15′	17.25	27.8	1036800	813600	190	89	1:500,000	2.2 × 2.2
12	7.5′ × 7.5′	8.625	13.9	4147200	3277440	95	44	1:250,000	2.2 × 2.2
13	3.75′ × 3.75′	4.312	6.9	16588800	13109760	47.4	22	1:125,000	2.2 × 2.2
								1:100,000	2.73 × 2.73
								1:80,000	3.4 × 3.4
14	1.875′ × 1.875′	2.156	3.5	66355200	52439040	23.7	11.1	1:62,500	2.2 × 2.2
								1:50,000	2.73 × 2.73
								1:40,000	3.4 × 3.4
15	0.9375′ × 0.9375′	1.078	1.7	265420800	209756160	11.9	5.6	1:24,000	2.8 × 2.8
								1:20,000	3.4 × 3.4
16	0.46875′ × 0.46875′	0.539	0.9	1016683200	839024640	5.9	2.8	1:12,000	2.8 × 2.8

The best way to see the advantages of the 512°×512° mapping projection, is to use it with the previously, taught, quadrant division and pyrimidal file structure to show how this unique mapping projection can provide digital maps of any geographical areas of the earth, with 16 levels or magnitudes of resolution. As it is slightly

In looking at the left-most column, and tracing down to magnitude **1**, note that the 256°×256° window size exactly matches our determination. Furthermore, note that our findings is also in agreement with the number of widows i.e., 4. It is also interesting to note from the third column, that the height or "relative viewing posi-

tion" of this magnitude or level would be 17, 664 statute miles above the earth's surface.

Turning now to the second level or magnitude of resolution (FIG. **15**). a further quadrant division is applied, resulting in sixteen (**16**) mapping areas of 128°×128°. The respective filenames which are assigned to each of the mapping areas is shown. In viewing FIG. **15**, note that there are eight (8) mapping areas which are not intersected by the earth's surface. In order to save valuable memory space, the preferred embodiment will ignore, and in fact will never create these files. Note that there is no use for these files as they do not contain any digital mapping data nor will they ever have any descendents which hold mapping data. In order to implement this "file selectivity", the preferred embodiment again utilizes a watchdog approach. More specifically, as the computer already knows the degree (°) size of the earth's surface and the degree (°) size of each of the mapping areas (i.e., at each level or magnitude of resolution), it can be seen that the computer can easily calculate the filenames which will not intersect the earth's surface.

Again it is useful to correspond our findings with the entries in the chart.

Our findings are substantiated, as, at a magnitude of 2, the window size is shown as being 128°×128°, and there are shown to be eight (8) pertinent windows or files at this magnitude. Again, it is interesting to note that the height or "relative viewing position" of this window would be 8,832 statute miles above the earths earth's surface.

It is important to note that, although the "relative viewing position" of each level or magnitude is moving closer to the earth, the visual perception of the earth (as seen in FIGS. **14–19** is not illustrated as getting larger with a greater degree of detail. This is because or the paper size limitations.

In the third level or magnitude of resolution (FIG. **16**). a further quadrant division is applied, resulting in sixty-four (64) mapping areas of of 64°×64°. As the projection is beginning to represent a large plurality of mapping areas, the filenames have been ommitted. However, it should be understood that the filename assigned to a respective file in this and subsequent degrees of resolution, can easily be calculated by following the previously described file naming operation. In this projection, it can be seen that 40 mapping areas or files are not used, resulting in 24 files which contain the digital mapping data of this resolution. Note that the observed window, and used files again correlates to the entries in the chart. Furthermore, it can be seen that the height or "relative viewing position" is at 4,416 statute miles above the earth.

Further quadrant divisions and the corresponding data can be seen in the FIGS. **17–19** and the chart. From the foregoing discussions, prior teachings, and data from the chart, one skilled in the art should be able to quickly appreciate that a mapping system can be constructed which can provide digital maps corresponding to a plurality of resolutions, of any geographical area of the world.

The chart can now be used to observe the tremendous advantage provided by the 512°×512° projection. In the second column of the chart, one can view the sizes of the mapping area divisions which are produced as a result of the continued quadrant division of the 512°×512° projection. One skilled in the mapping art will be able to fully appreciate that the resultant mapping area divisions exactly correspond to well settled and widely used mapping area formats.

Having described all of the important operations of the present invention, the following further conclusions, comments and teachings can be made.

With the mapping system of the present invention, the mapping data are structured at each magnitude or level into windows, frames or tiles representing subdivisions or partitions of the surface area at the specified magnitude. The windows, frames or tiles of all magnitudes for whatever resolution are structured to receive substantially the same amount or quantity of mapping data for segmented visual presentation of the mapping data by window.

As a further improvement, the lapping system of the present invention can further store and organize mapping data into attributed or coded geographical and cultural features according to the classification and level or resolution or magnitude for presentation on the map display. Several examples of this was previously discussed with regard to the use of the filename extension. If this further improvement is used, the computer can be programmed and arranged for managing and accessing the mapping data, and excluding or including coded features in tiles of a particular magnitude according to the resolution and density of mapping data appropriate to the particular magnitude of the window. The selective display of attributed geographical and cultural features according to resolution maintains or limits the mapping data entered in each tile to no greater than a specified full complement of mapping data for whatever magnitude.

In reviewing the file naming operations which were described, one can see that the global map generating system data base structure relates tiles of the same magnitude by tile position coordinates that are keyed to the control corner of each tile and maintained in the name of the "tile-file". Continuity of same scale tiles is maintained during scrolling between adjacent or neighboring tiles in any direction. The new data base structure also relates tiles of different magnitudes by vertical lineage through successive magnitudes. Each tile of a higher magnitude and lower resolution is an "ancestor tile" encompassing a lineage of "descendant tiles" of lower magnitude and high resolution in the next lower magnitude. Thus the present invention permits accessing, displaying and presenting the structured mapping data by tile, by scrolling between adjacent or neighboring tiles of different magnitude in the same vertical lineage for varying the resolution.

In its simplest form the coordinate system is Cartesian, but the invention contemplates a variety of virtual tile manifestations of windowing the mapping data at each magnitude: for example: tilting the axes; scaling one axis relative to another; having one or both axes logarithmic; or rendering the coordinate space as non-Euclidean all together.

When dealing with vector or point information and gridded data, the most common method is to describe individual points as an x-y offset from the control corner of the tile. In this way the mapping data exist as pre-processed relative points on a spherical surface in a de-projected space. The mapping data can then be projected at the user interface with an application program. When projected, all data ultimately represent points of latitude and longitude. Tiles may also contain mapping data as variable offsets of arc in the x and y directions. The tile header may carry an internal descriptor defin-

ing what type of mapping data is contained. The application or display program may then decode and project the data to the appropriate latitude or longitude positions.

The map generating system contemplates storing analog mapping data in electronic mapping frames in which the raw analog data would be scanned and converted digitally to the tile structure and then later accessed and projected for the purpose of displaying continuous analog mapping data.

In the preferred example embodiment, the digital mapping data are structured by window or tile in a substantially rectangular configuration encompassing defined widths and heights in degrees of latitude and longitude for each magnitude. The mapping data representing each magnitude or level are stored in a de-projected format according to mapping on an imaginary cylindrical surface. For display of the maps, however, the data base manager accesses and presents the tiles in a projected form, according to the real configuration of the mapped surface, by varying the aspect ratio of latitude to longitude dimensions of the tiles according to the absolute position of the window on the surface area.

For example, for a spherical or spheroidal globe having an equator and poles, such as the earth, the mapping data are accessed and displayed by aspecting or narrowing the width in the west-east dimension of the tiles of the same magnitude, while scrolling from the equator to the poles. This is accomplished by altering the width of the tile relative to the height. In the graphics display of each window or tile on the monitor, the tiles are presented essentially as rectangles having an aspect ratio substantially equal to the center latitude encompassed by the tile. Thus, the width of the visual display windows is corrected in two respects. First, the overall width is corrected by aspecting to a narrower width, during scrolling in the direction of the poles, and to a wider width during scrolling in the direction of the equator. Second, the width of the tile is averaged to the center latitude width encompassed by the tile throughout the tile height to conserve the rectangular configuration. Alternatively, or in addition, further compensation may be provided by increasing the number of degrees of longitude encompassed by the tiles during scrolling from the equator to the poles to compensate for the compound curvature of the globe.

A feature and advantage of this new method and new system of map projection are that the dramatic and perverse distortion of the globe near the poles, introduced by the traditional and conventional Mercator projection is substantially eliminated. According to the invention, the compensating aspect ratio of latitudinal to longitudinal dimension of aspecting is a function of the distance from the equator, where the aspect ratio is one, to the poles where the aspect ratio approaches zero, all as described for example in Elements of Cartography, 4th edition. John Wiley & Sons (1978) by Arthur Robinson, Randall Sale and Joel Morrison.

The new system contemplates "polar compression" (FIG. 20) in the following manner. Starting at 64 degrees latitude, the width of each tile doubles for every eight degrees of latitude. From 72 degrees to 80 degrees latitude, there are 4 degrees of longitude for 1 degree of latitude. From 80 degrees to 88 degrees latitude, it becomes eight to one, and from 88 degrees to the pole (90 degrees) it becomes 16 to one (see illustration of polar compression). (FIG. 20)

Another feature and advantage of the way in which the new map system and new projection handle polar mapping data are in the speed required to access and display polar data. The new polar compression method drastically minimizes tile or window seeks and standard I/O time. Also, without compressing the poles, the Creation/Edit Software would have to work on increasingly narrow tiles as the aspect ratio approached zero at the poles.

The invention embodies an entirely new cartographic organization for an automated atlas of the earth or other generally spherical or spheroidal globe with 360 degrees of longitude and 180 degrees of latitude, an equator and poles. The digital mapping data for the earth is structured on an imaginary surface space having 512 degrees of latitude and longitude. The imaginary 512 degree square surface represents the zero magnitude or root node at the highest level above the earth for a hierarchial type quadtree data base structure. In fact, the 512 degree square plane at the zero magnitude encompasses the entire earth in a single tile. The map of the earth, of course, fills only a portion of the root node window of 512 degrees square, and the remainder may be deemed imaginary space or "hyperspace".

In the preferred example embodiment from a zero magnitude virtual or imaginary space 512 degrees square, the data base structure of the global map generating system descends to a first magnitude of mapping data in four tiles, windows or quadrants, each comprising 256 degrees of latitude and longitude. Each quadrant represents mapping data for one-quarter of the earth thereby mapping 180 degrees of longitude and 90 degrees of latitude in the imaginary surface of the tile or frame comprising 256 degrees square, leaving excess imaginary space or "hyperspace". In the second magnitude, the digital mapping data are virtually mapped and stored in an organization of 16 tiles or windows each comprising 128 degrees of latitude and longitude.

The map generating system supports two windowing formats, one based on the binary system of the 512 degree square zero magnitude root node with hyperspace and the other based on a system of a 360 degree square root node without hyperspace. A feature and advantage of the virtual 512 degree data base structure with hyperspace are that the tiles or windows to be displayed at respective magnitudes are consistent with conventional mapping scale divisions, for example, those followed by the U.S. Geological Survey (USGS). Defense Mapping Agency (DMA). National Aeronautics and Space Administration (NASA) and other government mapping agencies. Thus, typical mapping scale divisions of the USGS and military mapping agencies include scale divisions in the same range of 1 deg, 30 minutes. 15 minutes. 7.5 minutes of arc on the earth s surface. This common subdivision of mapping space does not exist in a data structure based on a 360 degree model without hyperspace (see chart).

Thus, according to the present invention, the world is represented in an assemblage of magnitudes, with each magnitude divided into adjacent tiles or windows on a virtual or imaginary two-dimensional plane or cylinder. At higher magnitudes the quadtree tiles of mapping data do not fill the imaginary projection space. However, from the seventh magnitude down, the mapping data fills a virtual closed cylinder, and no hyperspace exists at these levels.

In the preferred example embodiment the invention (running on a 16 bit computer) has sixteen magnitudes

or levels (with extensions to 20 levels) representing sixteen altitudes or distances above the surface of the earth. At the lowest (16th) magnitude of highest resolution and closest to the earth, the data base structure contains over one billion tiles or windows (excluding hyperspace), each encompassing a tile height of approximately one half statute mile. At this level of resolution, one pixel on a monitor of 480 pixels in height represents approximately 6 feet on the ground. Mapping data are positioned within each tile using a 0 to 1023 offset coordinate structure, resulting in a data resolution of approximately 3 feet at this level of magnitude (see chart). The contemplated 20th magnitude tile or window height is approximately 175 feet, which results in a pixel resolution of about 4 inches on a monitor of 480 pixels in height and a data resolution of about 2 inches, when utilizing the 0 to 1023 offset coordinate structure. Alternatively, the map-generating system contemplates an extended offset from 10 bits (0 to 1023) to an offset of 16 bits (0 to 65,535). In this case, the extended 20th magnitude results in a data resolution of 3 hundredths of an inch.

For still more resolution, the map generating system contemplates 32 magnitudes on a 32 bit computer and representing 32 altitudes or distances about the surface of the earth. Each level of magnitude may define mapping data within each tile using a 32 bit offset coordinate structure, thereby giving relative mathematical accuracy to a billionth of an inch. In all practicality, 20 separate magnitudes or levels are more than sufficient to carry the necessary levels of resolution and accuracy.

The new invention provides users with the ability graphically to view mapping data from any part of the world-wide data base graphically on a monitor, either by entering coordinates and a level of zoom (or magnitude) on the keyboard, or by "flying" to that location in the "step-zoom" mode using consecutive clicks of the mouse or other pointing device. Once a location has been chosen (this point becomes the user-defined screen center). the mapping software accesses all adjacent tiles needed to fill the entire view window of the monitor and, then, projects the data to the screen. Same scale scrolling is accomplished by simply choosing a new screen center and maintaining the same magnitude.

Vertical zooming up or down is accomplished by choosing another magnitude or level from the menu area with the pointing device or by directly entering location and magnitude on the keyboard. An advantage of this vertical lineage of tiles organized in a quadtree structure is that it affords the efficient and easily followed zooming continuity inherent in the present invention. Further discussion of such quadtree data organization is found in the article. "The Quadtree and Related Hierarchical Data Structures", by Hannan Samet, Computer Surveys. Volume 16 , No. 2, (June 1984), Pages 187 et seq.

The map-generating system also supports many types of descriptive information such as that contained in tabular or relational data bases. This descriptive information can be linked to the mapping data with a latitude and longitude coordinate position but may need to be displayed in alternate ways. Descriptive information is better suited for storage in a relational format and can be linked to the map with a "spatial hook".

In summary, the present invention provides a new automated world atlas and global map generating system having a multi-level hierarchial quadtree data base structure and a data base manager or controller which permits scrolling, through mapping tiles or windows of a particular magnitude, and zooming between magnitudes for varying resolution. While the data base organization is hierarchial between levels or magnitudes, it is relational within each level, resulting in a three dimensional network of mapping and descriptive information. The present invention also provides a new mapping projection that has similarities to the Mercator projection but eliminates drastic distortions near the poles for the purpose of presentation through a method of "aspecting" tile widths as a function of the latitudinal distance from the equator.

While the invention has been particularly shown and described with reference to the preferred embodiment thereof, it will be understood by those skilled in the art that various changes in form and details of the device and the method may be made therein without departing from the spirit and scope of the invention.

What is claimed is:

1. A computer implemented method for generating, displaying and presenting an electronic map from digital mapping data for a surface area having geographical and cultural features, said method comprising the steps of:

organizing the mapping data into a hierarchy of a plurality of successive magnitudes or levels for presentation of said mapping data with variable degrees of mapping resolution, each magnitude for presentation of said mapping data with a different degree of mapping resolution from a first or highest magnitude with lowest resolution to a last or lowest magnitude with highest resolution;

structuring said mapping data at each magnitude into a plurality of windows, frames or files representing subdivisions or partitions of said surface area, said windows of a respective magnitude including mapping data which are appropriate to a degree of mapping resolution being afforded at said magnitude while excluding mapping data which are not appropriate to said degree of mapping resolution, and at least a portion of said windows of each magnitude being structured to receive substantially a same predetermined amount or quantity of mapping data for segmented presentation of the mapping data by window;

organizing said mapping data into records of geographical or cultural features for presentation within said windows, and coding said features;

managing said mapping data for each window by excluding or including coded features appropriate to the degree of mapping resolution and density being afforded by said window, such that a quantity of mapping data entered in each window is no greater than said predetermined amount;

relating windows of a same magnitude by window position coordinates or names and structuring said windows with overlap or mapping data between adjacent or neighboring windows of a magnitude or achieve display continuity during generation, display and presentation of an electronic map;

relating windows of different magnitude by vertical lineage through successive magnitudes, each window of a higher magnitude and lower resolution being an ancestor window being related to a plurality of descendant windows of lower magnitude and higher resolution in a next lower magnitude;

accessing and displaying or presenting mapping data for different positions of a selected magnitude by

scrolling between adjacent or neighboring windows of a same magnitude in predetermined north, south, each and west directions;

and accessing and displaying or presenting mapping data for different selected magnitudes having different resolutions by zooming between windows of different magnitudes in a same vertical lineage.

2. The method of claim 1 further comprising:

organizing said mapping data of said surface area by degrees of latitude and longitude;

structuring each said window of mapping data to represent a substantially rectangular surface area configuration encompassing defined degrees of latitude and longitude for each magnitude, and storing the mapping data for each magnitude in a vertical Mercator projection format;

accessing and presenting said windows of mapping data in a corrected or compensated projection format departing from said Mercator projection format according to a real configuration of said surface area, by varying an aspect ratio of latitude to longitudinal dimensions of each window according to a coordinate position of said window with respect to a coordinate layout of said surface area.

3. The method of claim 2 wherein said surface area comprises a spherical or spheroidal globe having an equator and poles, said method comprising the further steps of:

accessing and presenting mapping data in a corrected projection format by aspecting or narrowing, in a direction from an equator to pole, the width or latitudinal dimension of windows, of a same magnitude, which encompass the same number of degrees of latitude and longitude;

and periodically increasing a number of degrees of longitude encompassed by said windows in said direction from equator to pole to compensate for compound curvature of said globe.

4. The method of claim 1 wherein said surface area comprises a generally spherical or spheroidal globe with 360 degrees of longitudinal, 180 degrees of latitude and an equator and poles, said method comprising the further steps of:

relating windows of different magnitudes by vertical lineage in a hierarchical quadtree database structure, by successively partitioning or subdividing ancestor windows of a vertical lineage into four descent windows or quadrants at a next lower magnitude or level, and incorporating additional records of features in said descendant windows to incorporate mapping data for a next higher resolution.

5. The method of claim 4 wherein said hierarchical quadtree database structure comprises at least sixteen degrees of magnitudes or levels.

6. The method of claim 4 comprising the further steps of:

mapping and storing mapping data for said globe in a virtual Mercator projection format representing an imaginary surface having 512 degrees of longitude and latitude comprising a zero magnitude or root node of said hierarchical quadtree database structure;

mapping and storing a first degree or highest magnitude of mapping data in four windows or quadrants each comprising 256 degrees of longitude and latitude, each window of said first degree of magnitude comprising mapping data for one quarter of said globe thereby mapping 180 degrees of surface area longitude and 90 degrees of surface area latitude in said imaginary surface of 256 degrees of longitude and latitude and leaving excess imaginary space;

mapping and storing a second degree of magnitude of mapping data in sixteen windows each comprising 128 degrees of longitude and latitude of said imaginary surface, each window of said second degree of magnitude comprising mapping data for a further subdivision or partition of said globe;

and mapping and storing third through twelfth degrees of magnitude thereby forming additional levels of a hierarchical quadtree database structure so that an eleventh magnitude comprises windows encompassing 15 seconds of latitude and a twelfth magnitude comprises windows encompassing seven and a half seconds of latitude;

whereby, as a result of the foregoing, windows of said electronic map at respective magnitudes or levels are consistent with conventional mapping scale divisions.

7. The method of claim 6 wherein said hierarchical quadtree database structure comprises sixteen degree of magnitudes or levels including a sixteenth magnitude comprising over 1.4 billion windows, each encompassing approximately a fraction of a minute of a degree of latitude.

8. The method of claim 6 wherein each said window corresponds to a trapezoidal surface area configuration.

9. The method of claim 6 comprising the step of floating mapping data records of selected features from a window of one magnitude to a window of the same vertical lineage in another magnitude.

10. The method of claim 6 comprising the further steps of: generating analog mapping data, structuring said analog mapping data according to a same format as digital mapping data, and overlaying and presenting said digital mapping data and analog mapping data during generation, display and presentation of an electronic map.

11. The method of claim 6 comprising the further step of selectively filling said windows with mapping data so that some windows contain a full complement of mapping data appropriate to a degree of mapping resolution being afforded at said magnitude, and other windows, each of which correspond to a subdivision of surface area containing few or no geographical or cultural features, contain less than a full complement of mapping data.

12. The method of claim 6 comprising the further steps of:

accessing and presenting mapping data in a corrected projection format by aspecting or narrowing, in a direction from an equator to pole, a width or latitudinal dimension of windows, of a same magnitude, which encompass the same number of degrees of latitude and longitude;

and periodically increasing a number of degrees of longitude encompassed by said windows in said direction from equator to pole to compensate for a compound curvature of said globe.

13. The method of claim 12 comprising the further steps of accessing and presenting mapping data in corrected projection format, with each window having a width substantially equal to a center latitude width of said window throughout said window, so that said window is of rectangular configuration.

14. An electronic map generating system including a digital computer, a mass storage device, a display monitor, graphics controller, and system software for structuring, managing, controlling and displaying digital mapping data for a surface area having cultural and geographical features, said system comprising:

a database structure comprising a hierarchical database structure programmed and arranged for organizing said digital mapping data into a hierarchy of a plurality of successive magnitudes or levels for presentation of mapping data with variable resolution, each magnitude for presentation of said mapping data with a different degree of mapping resolution from a first or highest magnitude of lowest resolution to a last or lowest magnitude of lowest resolution to a last or lowest magnitude of highest resolution, and for structuring said digital mapping data at each magnitude into a plurality of windows, frames or files representing subdivisions or partitions of said surface area, said windows of a respective magnitude including mapping data which are appropriate to a degree of mapping resolution being afforded at said magnitude while excluding mapping data which are not appropriate to said degree of mapping resolution, at least a portion of said windows of all magnitudes being structured to receive substantially a same predetermined amount of mapping data for segmented presentation of said mapping data by window, said mapping data being organized into coded records of geographical and cultural features within each window;

a database manager or controller programmed and arranged for managing said mapping data by magnitude or level by excluding or including coded records of features in each window of a particular magnitude according to a resolution and density of mapping data appropriate to the particular magnitude of said each window, and maintaining a quantity of mapping data entered in each window to no greater than a specified full complement whatever the magnitude of the window;

said database structure being programmed to relate windows of a same magnitude by position coordinates or names, and to structure windows of a same magnitude with overlap of mapping data between adjacent or neighboring windows of a magnitude to achieve display continuity during generation, display and presentation of an electronic map, and to relate windows of different magnitude by vertical lineage through successive magnitudes, each window of a higher magnitude and lower resolution being an ancestor window of a plurality of descendant windows of lower magnitude and higher resolution in a next lower magnitude;

said database manager being programmed to access and display or present mapping data for different positions of a selected magnitude by scrolling between adjacent or neighboring windows of a same magnitude in predetermined north, south, east and west directions, and being programmed to access and display or present mapping data for different magnitudes having different resolutions by zooming between windows of different magnitudes in a same vertical lineage.

15. The system of claim 14 wherein said hierarchical database structure is programmed to organize said mapping data by degrees of latitude and longitude and to structure each window of mapping data to represent a substantially rectangular surface area configuration encompassing predetermined degrees of latitude and longitude, said windows for each magnitude being stored in virtual Mercator projection format, said database manager being programmed to access and present windows of mapping data in a corrected or compensated projection format departing from Mercator projection format according to a real configuration of said surface area by varying an aspect ratio of latitude and longitude dimensions of each window according to a coordinate position of said each window with respect to a coordinate layout of said surface area.

16. The system of claim 15 wherein said surface area comprises a spherical or spheroidal globe having an equator and poles, and wherein said database manager is programmed to access and present mapping data in a corrected projection format by aspecting or narrowing, in a direction from an equator to pole, the width or latitudinal dimension of windows, of a same magnitude, which encompass the same number of degrees of longitude, said database manager being further programmed to periodically increase a number of degrees of longitude encompassed by said windows in said direction from equator to pole to compensate for compound curvature of said globe.

17. The system of claim 16 wherein said hierarchical database structure comprises a hierarchical quadtree database structure successively partitioning or subdividing ancestor windows of a vertical lineage into four descendant windows or quadrants at a next lower magnitude or level, and incorporating additional coded records of features in said descendant windows to incorporate mapping data for a next higher resolution.

18. The system of claim 17 wherein said database structure is programmed and arranged to store the mapping data in a virtual Mercator projection representing an imaginary surface having 512 degrees of longitude and latitude comprising a zero magnitude or root node of said hierarchical quadtree database structure, wherein a first degree or first magnitude of mapping data comprises four windows, each window of said first magnitude comprising mapping data for one quarter of said globe on an imaginary surface area of 256 degrees of longitude and latitude, said hierarchical quadtree database structure comprising, in addition to first through tenth magnitudes each having windows which are predetermined subdivisions of said imaginary surface having 512 degrees of longitude and latitude, at least an eleventh magnitude having windows encompassing 15 minutes of latitude, and a twelfth magnitude having windows encompassing 7.5 minutes of latitude, so that windows of a resultant electronic map at respective said eleventh and twelfth magnitudes or levels are consistent with conventional mapping scale divisions.

19. The system of claim 18 wherein said hierarchical quadtree database structure comprises at least 16 degrees of magnitudes or levels, said sixteenth magnitude comprising over 1.4 billion windows, each encompassing degrees of latitude of approximately a fraction of a second of a degree.

20. The system of claim 19 further comprising a database of digital mapping data selectively entered in said database structure, such that some of said windows contain a full complement of mapping data appropriate to a degree of mapping resolution being afforded at said magnitude, and other windows, each of which correspond to a subdivision of surface area containing few or

no geographical or cultural features, contain less than a full complement of mapping data.

21. The system of claim **19** further comprising a database of analog data structured according to a same format as said digital data, and means for overlaying said digital and analog data for electronic map presentation.

22. An electronic map generating system for generating reproductions of a map with selectable degrees of mapping resolution, said map generating system comprising:

database means storing a plurality of computer files containing mapping data corresponding to respective surface areas of a mapping surface, wherein said plurality of computer files is organized into a plurality of successive magnitudes, each magnitude for presentation of said mapping data with a different degree of mapping resolution from a first or highest magnitude with lowest resolution to a last or lowest magnitude with highest resolution, files of a respective magnitude including mapping data which are appropriate to a degree of mapping resolution being afforded at said respective magnitude while excluding mapping data which are not appropriate to said degree of mapping resolution, and wherein a predetermined file naming procedure is utilized to assign, to each respective computer file, a unique filename which:

relates said respective computer file to all other computer files having mapping data corresponding to a same magnitude or degree of mapping resolution; and

relates said respective computer file to any computer file comprising mapping data corresponding to a same surface area of a mapping surface as said respective computer file; and

database manager means for accessing said plurality of computer files using said predetermined file naming procedure, to generate a reproduction of a selected area of a map at a selected degree of mapping resolution.

23. An electronic map generating system as claimed in claim **22**,

wherein each said unique filename is represented by a value contained in a plurality of bits, and

wherein said predetermined file naming procedure:

utilizes a first predetermined subset of said plurality of bits to relate said respective files having mapping data corresponding to a same magnitude or degree of mapping resolution; and

utilizes a second predetermined subset of said plurality of bits to relate said respective computer file to any computer file comprising mapping data corresponding to a same surface area of a mapping surface as said respective computer file.

24. An electronic map generating system as claimed in claim **23**, wherein said unique filename also includes geographical information which can be used to relate a geographical coordinate position of a respective computer file with respect to a coordinate layout of surface areas of said mapping surface.

25. An electronic map generating system as claimed in claim **22**,

wherein an assignment of said unique filenames using said predetermined file naming procedure results in said respective computer files of said plurality to be related in a quadtree database structure.

26. An electronic map generating system as claimed in claim **25**, wherein the respective area of a mapping surface covered within the computer files of consecutive magnitudes or degrees of mapping resolution changes at a predetermined rate in that, when a computer file at a reference magnitude or degree of mapping resolution contains mapping data corresponding to an $N \times N$ area of a mapping surface (where N is a real number, and is associated with one of the conventional degree °, minute ′, or second ″ mapping scale divisions), then a computer file at a next consecutive magnitude having a higher degree of mapping resolution contains mapping data corresponding to an $(N/2) \times (N/2)$ area of said mapping surface.

27. An electronic map generating system as claimed in claim **26**, wherein the value of N at said reference magnitude or degree of mapping resolution, corresponds to one of the following values: 512°, 256°, 128°, 64°, 32°, 16°, 8°, 4°, 2°, 1°, 30′, 15′, 7.5′, 3.75′, 1.875′, 0.9375′ and 0.46875′.

28. A method for providing an electronic map generating system for generating reproductions of a map with selectable degrees of mapping resolution, said method comprising the steps of:

storing a plurality of computer files containing mapping data corresponding to respective surface areas of a mapping surface, wherein said plurality of computer files is organized into a plurality of successive magnitudes, each magnitude for presentation of said mapping data with a different degree of mapping resolution from a first or highest magnitude with lowest resolution to a last or lowest magnitude with highest resolution, files of a respective magnitude including mapping data which are appropriate to a degree of mapping resolution being afforded at said respective magnitude while excluding mapping data which are not appropriate to said degree of mapping resolution, and wherein a predetermined file naming procedure is utilized to assign, to each respective computer file, a unique filename which:

relates said respective computer file to all other computer files having mapping data corresponding to a same magnitude or degree of mapping resolution; and

relates said respective computer file to any computer file comprising mapping data corresponding to a same surface area of a mapping surface as said respective computer file; and

accessing said plurality of computer files using said predetermined file naming procedure, to generate a reproduction of a selected area of a map at a selected degree of mapping resolution.

29. A method as claimed in claim **28**,

wherein each said unique filename is represented by a value contained in a plurality of bits, and

wherein said predetermined file naming procedure;

utilizes a first predetermined subset of said plurality of bits to relate said respective computer file to all other computer files having mapping data corresponding to a same magnitude or degree of mapping resolution; and

utilizes a second predetermined subset of said plurality of bits to relate said respective computer file to any computer file comprising mapping data corresponding to a same surface area of a mapping surface as said respective computer file.

30. A method as claimed in claim 29, wherein said unique filename also includes geographical information which can be used to relate a geographical coordinate position of a respective computer file with respect to a coordinate layout of surface areas of said mapping surface.

31. A method as claimed in claim 28,

wherein an assignment of said unique filenames using said predetermined file naming procedure results in said respective computer files of said plurality to be related in a quadtree database structure.

32. A method as claimed in claim 31, wherein the respective area of a mapping surface covered within the computer files of consecutive magnitudes or degrees of mapping resolution changes at a predetermined rate in that, when a computer file at a reference magnitude or degree of mapping resolution contains mapping data corresponding to an N×N area of a mapping surface (where N is a real number, and is associated with one of the conventional degree °, minute ′, or second ″ mapping scale divisions), then a computer file at a next consecutive magnitude having a higher degree of mapping resolution contains mapping data corresponding to an (N/2)×(N/2) area of said mapping surface.

33. A method as claimed in claim 32, wherein the value of N at said reference magnitude or degree of mapping resolution, corresponds to one of the following values: 512°, 256°, 128°, 64°, 32°, 16°, 8°, 4°, 2°, 1°, 30′, 15′, 7.5′, 3.75′, 1.875′, 0.9375′ and 0.46875′.

* * * * *

APPENDIX F

INTERNATIONAL TELECOMMUNICATION UNION

CCITT

THE INTERNATIONAL
TELEGRAPH AND TELEPHONE
CONSULTATIVE COMMITTEE

T.81

(09/92)

TERMINAL EQUIPMENT AND PROTOCOLS FOR TELEMATIC SERVICES

INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES

Recommendation T.81

Foreword

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The CCITT (the International Telegraph and Telephone Consultative Committee) is a permanent organ of the ITU. Some 166 member countries, 68 telecom operating entities, 163 scientific and industrial organizations and 39 international organizations participate in CCITT which is the body which sets world telecommunications standards (Recommendations).

The approval of Recommendations by the members of CCITT is covered by the procedure laid down in CCITT Resolution No. 2 (Melbourne, 1988). In addition, the Plenary Assembly of CCITT, which meets every four years, approves Recommendations submitted to it and establishes the study programme for the following period.

In some areas of information technology, which fall within CCITT's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC. The text of CCITT Recommendation T.81 was approved on 18th September 1992. The identical text is also published as ISO/IEC International Standard 10918-1.

CCITT NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized private operating agency.

Contents

		Page
Introduction		iii
1	Scope	1
2	Normative references	1
3	Definitions, abbreviations and symbols	1
4	General	12
5	Interchange format requirements	23
6	Encoder requirements	23
7	Decoder requirements	23
Annex A – Mathematical definitions		24
Annex B – Compressed data formats		31
Annex C – Huffman table specification		50
Annex D – Arithmetic coding		54
Annex E – Encoder and decoder control procedures		77
Annex F – Sequential DCT-based mode of operation		87
Annex G – Progressive DCT-based mode of operation		119
Annex H – Lossless mode of operation		132
Annex J – Hierarchical mode of operation		137
Annex K – Examples and guidelines		143
Annex L – Patents		179
Annex M – Bibliography		181

APPENDIX F

Introduction

This CCITT Recommendation | ISO/IEC International Standard was prepared by CCITT Study Group VIII and the Joint Photographic Experts Group (JPEG) of ISO/IEC JTC 1/SC 29/WG 10. This Experts Group was formed in 1986 to establish a standard for the sequential progressive encoding of continuous tone grayscale and colour images.

Digital Compression and Coding of Continuous-tone Still images, is published in two parts:

- Requirements and guidelines;
- Compliance testing.

This part, Part 1, sets out requirements and implementation guidelines for continuous-tone still image encoding and decoding processes, and for the coded representation of compressed image data for interchange between applications. These processes and representations are intended to be generic, that is, to be applicable to a broad range of applications for colour and grayscale still images within communications and computer systems. Part 2, sets out tests for determining whether implementations comply with the requirments for the various encoding and decoding processes specified in Part 1.

The user's attention is called to the possibility that – for some of the coding processes specified herein – compliance with this Recommendation | International Standard may require use of an invention covered by patent rights. See Annex L for further information.

The requirements which these processes must satisfy to be useful for specific image communications applications such as facsimile, Videotex and audiographic conferencing are defined in CCITT Recommendation T.80. The intent is that the generic processes of Recommendation T.80 will be incorporated into the various CCITT Recommendations for terminal equipment for these applications.

In addition to the applications addressed by the CCITT and ISO/IEC, the JPEG committee has developped a compression standard to meet the needs of other applications as well, including desktop publishing, graphic arts, medical imaging and scientific imaging.

Annexes A, B, C, D, E, F, G, H and J are normative, and thus form an integral part of this Specification. Annexes K, L and M are informative and thus do not form an integral part of this Specification.

This Specification aims to follow the guidelines of CCITT and ISO/IEC JTC 1 on *Rules for presentation of CCITT | ISO/IEC common text*.

INTERNATIONAL STANDARD

CCITT RECOMMENDATION

INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES

1 Scope

This CCITT Recommendation | International Standard is applicable to continuous-tone – grayscale or colour – digital still image data. It is applicable to a wide range of applications which require use of compressed images. It is not applicable to bi-level image data.

This Specification

- specifies processes for converting source image data to compressed image data;
- specifies processes for converting compressed image data to reconstructed image data;
- gives guidance on how to implement these processes in practice;
- specifies coded representations for compressed image data.

NOTE – This Specification does not specify a complete coded image representation. Such representations may include certain parameters, such as aspect ratio, component sample registration, and colour space designation, which are application-dependent.

2 Normative references

The following CCITT Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this CCITT Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this CCITT Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The CCITT Secretariat maintains a list of currently valid CCITT Recommendations.

- *CCITT Recommendation T.80 (1992), Common components for image compression and communication – Basic principles.*

3 Definitions, abbreviations and symbols

3.1 Definitions and abbreviations

For the purposes of this Specification, the following definitions apply.

3.1.1 abbreviated format: A representation of compressed image data which is missing some or all of the table specifications required for decoding, or a representation of table-specification data without frame headers, scan headers, and entropy-coded segments.

3.1.2 AC coefficient: Any DCT coefficient for which the frequency is not zero in at least one dimension.

3.1.3 (adaptive) (binary) arithmetic decoding: An entropy decoding procedure which recovers the sequence of symbols from the sequence of bits produced by the arithmetic encoder.

3.1.4 (adaptive) (binary) arithmetic encoding: An entropy encoding procedure which codes by means of a recursive subdivision of the probability of the sequence of symbols coded up to that point.

3.1.5 application environment: The standards for data representation, communication, or storage which have been established for a particular application.

3.1.6 arithmetic decoder: An embodiment of arithmetic decoding procedure.

3.1.7 arithmetic encoder: An embodiment of arithmetic encoding procedure.

3.1.8 baseline (sequential): A particular sequential DCT-based encoding and decoding process specified in this Specification, and which is required for all DCT-based decoding processes.

3.1.9 binary decision: Choice between two alternatives.

3.1.10 bit stream: Partially encoded or decoded sequence of bits comprising an entropy-coded segment.

3.1.11 block: An 8×8 array of samples or an 8×8 array of DCT coefficient values of one component.

3.1.12 block-row: A sequence of eight contiguous component lines which are partitioned into 8×8 blocks.

3.1.13 byte: A group of 8 bits.

3.1.14 byte stuffing: A procedure in which either the Huffman coder or the arithmetic coder inserts a zero byte into the entropy-coded segment following the generation of an encoded hexadecimal X'FF' byte.

3.1.15 carry bit: A bit in the arithmetic encoder code register which is set if a carry-over in the code register overflows the eight bits reserved for the output byte.

3.1.16 ceiling function: The mathematical procedure in which the greatest integer value of a real number is obtained by selecting the smallest integer value which is greater than or equal to the real number.

3.1.17 class (of coding process): Lossy or lossless coding processes.

3.1.18 code register: The arithmetic encoder register containing the least significant bits of the partially completed entropy-coded segment. Alternatively, the arithmetic decoder register containing the most significant bits of a partially decoded entropy-coded segment.

3.1.19 coder: An embodiment of a coding process.

3.1.20 coding: Encoding or decoding.

3.1.21 coding model: A procedure used to convert input data into symbols to be coded.

3.1.22 (coding) process: A general term for referring to an encoding process, a decoding process, or both.

3.1.23 colour image: A continuous-tone image that has more than one component.

3.1.24 columns: Samples per line in a component.

3.1.25 component: One of the two-dimensional arrays which comprise an image.

3.1.26 compressed data: Either compressed image data or table specification data or both.

3.1.27 compressed image data: A coded representation of an image, as specified in this Specification.

3.1.28 compression: Reduction in the number of bits used to represent source image data.

3.1.29 conditional exchange: The interchange of MPS and LPS probability intervals whenever the size of the LPS interval is greater than the size of the MPS interval (in arithmetic coding).

3.1.30 (conditional) probability estimate: The probability value assigned to the LPS by the probability estimation state machine (in arithmetic coding).

3.1.31 conditioning table: The set of parameters which select one of the defined relationships between prior coding decisions and the conditional probability estimates used in arithmetic coding.

3.1.32 context: The set of previously coded binary decisions which is used to create the index to the probability estimation state machine (in arithmetic coding).

3.1.33 continuous-tone image: An image whose components have more than one bit per sample.

3.1.34 data unit: An 8×8 block of samples of one component in DCT-based processes; a sample in lossless processes.

3.1.35 DC coefficient: The DCT coefficient for which the frequency is zero in both dimensions.

3.1.36 DC prediction: The procedure used by DCT-based encoders whereby the quantized DC coefficient from the previously encoded 8×8 block of the same component is subtracted from the current quantized DC coefficient.

3.1.37 (DCT) coefficient: The amplitude of a specific cosine basis function – may refer to an original DCT coefficient, to a quantized DCT coefficient, or to a dequantized DCT coefficient.

3.1.38 decoder: An embodiment of a decoding process.

3.1.39 decoding process: A process which takes as its input compressed image data and outputs a continuous-tone image.

3.1.40 default conditioning: The values defined for the arithmetic coding conditioning tables at the beginning of coding of an image.

3.1.41 dequantization: The inverse procedure to quantization by which the decoder recovers a representation of the DCT coefficients.

3.1.42 differential component: The difference between an input component derived from the source image and the corresponding reference component derived from the preceding frame for that component (in hierarchical mode coding).

3.1.43 differential frame: A frame in a hierarchical process in which differential components are either encoded or decoded.

3.1.44 (digital) reconstructed image (data): A continuous-tone image which is the output of any decoder defined in this Specification.

3.1.45 (digital) source image (data): A continuous-tone image used as input to any encoder defined in this Specification.

3.1.46 (digital) (still) image: A set of two-dimensional arrays of integer data.

3.1.47 discrete cosine transform; DCT: Either the forward discrete cosine transform or the inverse discrete cosine transform.

3.1.48 downsampling (filter): A procedure by which the spatial resolution of an image is reduced (in hierarchical mode coding).

3.1.49 encoder: An embodiment of an encoding process.

3.1.50 encoding process: A process which takes as its input a continuous-tone image and outputs compressed image data.

3.1.51 entropy-coded (data) segment: An independently decodable sequence of entropy encoded bytes of compressed image data.

3.1.52 (entropy-coded segment) pointer: The variable which points to the most recently placed (or fetched) byte in the entropy encoded segment.

3.1.53 entropy decoder: An embodiment of an entropy decoding procedure.

3.1.54 entropy decoding: A lossless procedure which recovers the sequence of symbols from the sequence of bits produced by the entropy encoder.

3.1.55 entropy encoder: An embodiment of an entropy encoding procedure.

3.1.56 entropy encoding: A lossless procedure which converts a sequence of input symbols into a sequence of bits such that the average number of bits per symbol approaches the entropy of the input symbols.

3.1.57 extended (DCT-based) process: A descriptive term for DCT-based encoding and decoding processes in which additional capabilities are added to the baseline sequential process.

3.1.58 forward discrete cosine transform; FDCT: A mathematical transformation using cosine basis functions which converts a block of samples into a corresponding block of original DCT coefficients.

3.1.59 frame: A group of one or more scans (all using the same DCT-based or lossless process) through the data of one or more of the components in an image.

3.1.60 frame header: A marker segment that contains a start-of-frame marker and associated frame parameters that are coded at the beginning of a frame.

3.1.61 frequency: A two-dimensional index into the two-dimensional array of DCT coefficients.

3.1.62 (frequency) band: A contiguous group of coefficients from the zig-zag sequence (in progressive mode coding).

3.1.63 full progression: A process which uses both spectral selection and successive approximation (in progressive mode coding).

3.1.64 grayscale image: A continuous-tone image that has only one component.

3.1.65 hierarchical: A mode of operation for coding an image in which the first frame for a given component is followed by frames which code the differences between the source data and the reconstructed data from the previous frame for that component. Resolution changes are allowed between frames.

3.1.66 hierarchical decoder: A sequence of decoder processes in which the first frame for each component is followed by frames which decode an array of differences for each component and adds it to the reconstructed data from the preceding frame for that component.

3.1.67 hierarchical encoder: The mode of operation in which the first frame for each component is followed by frames which encode the array of differences between the source data and the reconstructed data from the preceding frame for that component.

3.1.68 horizontal sampling factor: The relative number of horizontal data units of a particular component with respect to the number of horizontal data units in the other components.

3.1.69 Huffman decoder: An embodiment of a Huffman decoding procedure.

3.1.70 Huffman decoding: An entropy decoding procedure which recovers the symbol from each variable length code produced by the Huffman encoder.

3.1.71 Huffman encoder: An embodiment of a Huffman encoding procedure.

3.1.72 Huffman encoding: An entropy encoding procedure which assigns a variable length code to each input symbol.

3.1.73 Huffman table: The set of variable length codes required in a Huffman encoder and Huffman decoder.

3.1.74 image data: Either source image data or reconstructed image data.

3.1.75 interchange format: The representation of compressed image data for exchange between application environments.

3.1.76 interleaved: The descriptive term applied to the repetitive multiplexing of small groups of data units from each component in a scan in a specific order.

3.1.77 inverse discrete cosine transform; IDCT: A mathematical transformation using cosine basis functions which converts a block of dequantized DCT coefficients into a corresponding block of samples.

3.1.78 Joint Photographic Experts Group; JPEG: The informal name of the committee which created this Specification. The “joint” comes from the CCITT and ISO/IEC collaboration.

3.1.79 latent output: Output of the arithmetic encoder which is held, pending resolution of carry-over (in arithmetic coding).

3.1.80 less probable symbol; LPS: For a binary decision, the decision value which has the smaller probability.

3.1.81 level shift: A procedure used by DCT-based encoders and decoders whereby each input sample is either converted from an unsigned representation to a two’s complement representation or from a two’s complement representation to an unsigned representation.

3.1.82 lossless: A descriptive term for encoding and decoding processes and procedures in which the output of the decoding procedure(s) is identical to the input to the encoding procedure(s).

3.1.83 lossless coding: The mode of operation which refers to any one of the coding processes defined in this Specification in which all of the procedures are lossless (see Annex H).

3.1.84 lossy: A descriptive term for encoding and decoding processes which are not lossless.

3.1.85 marker: A two-byte code in which the first byte is hexadecimal FF (X'FF') and the second byte is a value between 1 and hexadecimal FE (X'FE').

3.1.86 marker segment: A marker and associated set of parameters.

3.1.87 MCU-row: The smallest sequence of MCU which contains at least one line of samples or one block-row from every component in the scan.

3.1.88 minimum coded unit; MCU: The smallest group of data units that is coded.

3.1.89 modes (of operation): The four main categories of image coding processes defined in this Specification.

3.1.90 more probable symbol; MPS: For a binary decision, the decision value which has the larger probability.

3.1.91 non-differential frame: The first frame for any components in a hierarchical encoder or decoder. The components are encoded or decoded without subtraction from reference components. The term refers also to any frame in modes other than the hierarchical mode.

3.1.92 non-interleaved: The descriptive term applied to the data unit processing sequence when the scan has only one component.

3.1.93 parameters: Fixed length integers 4, 8 or 16 bits in length, used in the compressed data formats.

3.1.94 point transform: Scaling of a sample or DCT coefficient.

3.1.95 precision: Number of bits allocated to a particular sample or DCT coefficient.

3.1.96 predictor: A linear combination of previously reconstructed values (in lossless mode coding).

3.1.97 probability estimation state machine: An interlinked table of probability values and indices which is used to estimate the probability of the LPS (in arithmetic coding).

3.1.98 probability interval: The probability of a particular sequence of binary decisions within the ordered set of all possible sequences (in arithmetic coding).

3.1.99 (probability) sub-interval: A portion of a probability interval allocated to either of the two possible binary decision values (in arithmetic coding).

3.1.100 procedure: A set of steps which accomplishes one of the tasks which comprise an encoding or decoding process.

3.1.101 process: See coding process.

3.1.102 progressive (coding): One of the DCT-based processes defined in this Specification in which each scan typically improves the quality of the reconstructed image.

3.1.103 progressive DCT-based: The mode of operation which refers to any one of the processes defined in Annex G.

3.1.104 quantization table: The set of 64 quantization values used to quantize the DCT coefficients.

3.1.105 quantization value: An integer value used in the quantization procedure.

3.1.106 quantize: The act of performing the quantization procedure for a DCT coefficient.

3.1.107 reference (reconstructed) component: Reconstructed component data which is used in a subsequent frame of a hierarchical encoder or decoder process (in hierarchical mode coding).

3.1.108 renormalization: The doubling of the probability interval and the code register value until the probability interval exceeds a fixed minimum value (in arithmetic coding).

3.1.109 restart interval: The integer number of MCUs processed as an independent sequence within a scan.

3.1.110 restart marker: The marker that separates two restart intervals in a scan.

3.1.111 run (length): Number of consecutive symbols of the same value.

3.1.112 sample: One element in the two-dimensional array which comprises a component.

3.1.113 sample-interleaved: The descriptive term applied to the repetitive multiplexing of small groups of samples from each component in a scan in a specific order.

3.1.114 scan: A single pass through the data for one or more of the components in an image.

3.1.115 scan header: A marker segment that contains a start-of-scan marker and associated scan parameters that are coded at the beginning of a scan.

3.1.116 sequential (coding): One of the lossless or DCT-based coding processes defined in this Specification in which each component of the image is encoded within a single scan.

3.1.117 sequential DCT-based: The mode of operation which refers to any one of the processes defined in Annex F.

3.1.118 spectral selection: A progressive coding process in which the zig-zag sequence is divided into bands of one or more contiguous coefficients, and each band is coded in one scan.

3.1.119 stack counter: The count of X'FF' bytes which are held, pending resolution of carry-over in the arithmetic encoder.

3.1.120 statistical conditioning: The selection, based on prior coding decisions, of one estimate out of a set of conditional probability estimates (in arithmetic coding).

3.1.121 statistical model: The assignment of a particular conditional probability estimate to each of the binary arithmetic coding decisions.

3.1.122 statistics area: The array of statistics bins required for a coding process which uses arithmetic coding.

3.1.123 statistics bin: The storage location where an index is stored which identifies the value of the conditional probability estimate used for a particular arithmetic coding binary decision.

3.1.124 successive approximation: A progressive coding process in which the coefficients are coded with reduced precision in the first scan, and precision is increased by one bit with each succeeding scan.

3.1.125 table specification data: The coded representation from which the tables used in the encoder and decoder are generated and their destinations specified.

3.1.126 transcoder: A procedure for converting compressed image data of one encoder process to compressed image data of another encoder process.

3.1.127 (uniform) quantization: The procedure by which DCT coefficients are linearly scaled in order to achieve compression.

3.1.128 upsampling (filter): A procedure by which the spatial resolution of an image is increased (in hierarchical mode coding).

3.1.129 vertical sampling factor: The relative number of vertical data units of a particular component with respect to the number of vertical data units in the other components in the frame.

3.1.130 zero byte: The X'00' byte.

3.1.131 zig-zag sequence: A specific sequential ordering of the DCT coefficients from (approximately) lowest spatial frequency to highest.

3.1.132 3-sample predictor: A linear combination of the three nearest neighbor reconstructed samples to the left and above (in lossless mode coding).

3.2 Symbols

The symbols used in this Specification are listed below.

A	probability interval
AC	AC DCT coefficient
AC_{ji}	AC coefficient predicted from DC values
Ah	successive approximation bit position, high
Al	successive approximation bit position, low
Ap_i	*i*th 8-bit parameter in APP_n segment
APP_n	marker reserved for application segments
B	current byte in compressed data
B2	next byte in compressed data when B = X'FF'
BE	counter for buffered correction bits for Huffman coding in the successive approximation process
BITS	16-byte list containing number of Huffman codes of each length
BP	pointer to compressed data
BPST	pointer to byte before start of entropy-coded segment
BR	counter for buffered correction bits for Huffman coding in the successive approximation process
Bx	byte modified by a carry-over
C	value of bit stream in code register
C_i	component identifier for frame
C_u	horizontal frequency dependent scaling factor in DCT
C_v	vertical frequency dependent scaling factor in DCT
CE	conditional exchange
C-low	low order 16 bits of the arithmetic decoder code register
Cm_i	*i*th 8-bit parameter in COM segment
CNT	bit counter in NEXTBYTE procedure
CODE	Huffman code value
CODESIZE(V)	code size for symbol V
COM	comment marker
Cs	conditioning table value
Cs_i	component identifier for scan
CT	renormalization shift counter
Cx	high order 16 bits of arithmetic decoder code register
CX	conditional exchange
d_{ji}	data unit from horizontal position i, vertical position j
d_{ji}^k	d_{ji} for component k
D	decision decoded

Da	in DC coding, the DC difference coded for the previous block from the same component; in lossless coding, the difference coded for the sample immediately to the left
DAC	define-arithmetic-coding-conditioning marker
Db	the difference coded for the sample immediately above
DC	DC DCT coefficient
DC_i	DC coefficient for *i*th block in component
DC_k	*k*th DC value used in prediction of AC coefficients
DHP	define hierarchical progression marker
DHT	define-Huffman-tables marker
DIFF	difference between quantized DC and prediction
DNL	define-number-of-lines marker
DQT	define-quantization-tables marker
DRI	define restart interval marker
E	exponent in magnitude category upper bound
EC	event counter
ECS	entropy-coded segment
ECS_i	*i*th entropy-coded segment
Eh	horizontal expansion parameter in EXP segment
EHUFCO	Huffman code table for encoder
EHUFSI	encoder table of Huffman code sizes
EOB	end-of-block for sequential; end-of-band for progressive
EOBn	run length category for EOB runs
EOBx	position of EOB in previous successive approximation scan
EOB0, EOB1, ..., EOB14	run length categories for EOB runs
EOI	end-of-image marker
Ev	vertical expansion parameter in EXP segment
EXP	expand reference components marker
FREQ(V)	frequency of occurrence of symbol V
H_i	horizontal sampling factor for *i*th component
H_{max}	largest horizontal sampling factor
HUFFCODE	list of Huffman codes corresponding to lengths in HUFFSIZE
HUFFSIZE	list of code lengths
HUFFVAL	list of values assigned to each Huffman code
i	subscript index
I	integer variable
Index(S)	index to probability estimation state machine table for context index S
j	subscript index
J	integer variable

JPG	marker reserved for JPEG extensions
JPG_n	marker reserved for JPEG extensions
k	subscript index
K	integer variable
Kmin	index of 1st AC coefficient in band (1 for sequential DCT)
Kx	conditioning parameter for AC arithmetic coding model
L	DC and lossless coding conditioning lower bound parameter
L_i	element in BITS list in DHT segment
$L_i(t)$	element in BITS list in the DHT segment for Huffman table t
La	length of parameters in APP_n segment
LASTK	largest value of K
Lc	length of parameters in COM segment
Ld	length of parameters in DNL segment
Le	length of parameters in EXP segment
Lf	length of frame header parameters
Lh	length of parameters in DHT segment
Lp	length of parameters in DAC segment
LPS	less probable symbol (in arithmetic coding)
Lq	length of parameters in DQT segment
Lr	length of parameters in DRI segment
Ls	length of scan header parameters
LSB	least significant bit
m	modulo 8 counter for RST_m marker
m_t	number of $V_{i,j}$ parameters for Huffman table t
M	bit mask used in coding magnitude of V
Mn	*n*th statistics bin for coding magnitude bit pattern category
MAXCODE	table with maximum value of Huffman code for each code length
MCU	minimum coded unit
MCU_i	*i*th MCU
MCUR	number of MCU required to make up one MCU-row
MINCODE	table with minimum value of Huffman code for each code length
MPS	more probable symbol (in arithmetic coding)
MPS(S)	more probable symbol for context-index S
MSB	most significant bit
M2, M3, M4, ... , M15	designation of context-indices for coding of magnitude bits in the arithmetic coding models
n	integer variable
N	data unit counter for MCU coding
N/A	not applicable

Nb	number of data units in MCU
Next_Index_LPS	new value of Index(S) after a LPS renormalization
Next_Index_MPS	new value of Index(S) after a MPS renormalization
Nf	number of components in frame
NL	number of lines defined in DNL segment
Ns	number of components in scan
OTHERS(V)	index to next symbol in chain
P	sample precision
Pq	quantizer precision parameter in DQT segment
Pq(t)	quantizer precision parameter in DQT segment for quantization table t
PRED	quantized DC coefficient from the most recently coded block of the component
Pt	point transform parameter
Px	calculated value of sample
Q_{ji}	quantizer value for coefficient AC_{ji}
Q_{vu}	quantization value for DCT coefficient S_{vu}
Q_{00}	quantizer value for DC coefficient
QAC_{ji}	quantized AC coefficient predicted from DC values
QDC_k	*k*th quantized DC value used in prediction of AC coefficients
Qe	LPS probability estimate
Qe(S)	LPS probability estimate for context index S
Qk	*k*th element of 64 quantization elements in DQT segment
r_{vu}	reconstructed image sample
R	length of run of zero amplitude AC coefficients
R_{vu}	dequantized DCT coefficient
Ra	reconstructed sample value
Rb	reconstructed sample value
Rc	reconstructed sample value
Rd	rounding in prediction calculation
RES	reserved markers
Ri	restart interval in DRI segment
RRRR	4-bit value of run length of zero AC coefficients
RS	composite value used in Huffman coding of AC coefficients
RST_m	restart marker number m
s_{yx}	reconstructed value from IDCT
S	context index
S_{vu}	DCT coefficient at horizontal frequency u, vertical frequency v

SC	context-index for coding of correction bit in successive approximation coding
Se	end of spectral selection band in zig-zag sequence
SE	context-index for coding of end-of-block or end-of-band
SI	Huffman code size
SIGN	1 if decoded sense of sign is negative and 0 if decoded sense of sign is positive
SIZE	length of a Huffman code
SLL	shift left logical operation
SLL α β	logical shift left of α by β bits
SN	context-index for coding of first magnitude category when V is negative
SOF_0	baseline DCT process frame marker
SOF_1	extended sequential DCT frame marker, Huffman coding
SOF_2	progressive DCT frame marker, Huffman coding
SOF_3	lossless process frame marker, Huffman coding
SOF_5	differential sequential DCT frame marker, Huffman coding
SOF_6	differential progressive DCT frame marker, Huffman coding
SOF_7	differential lossless process frame marker, Huffman coding
SOF_9	sequential DCT frame marker, arithmetic coding
SOF_{10}	progressive DCT frame marker, arithmetic coding
SOF_{11}	lossless process frame marker, arithmetic coding
SOF_{13}	differential sequential DCT frame marker, arithmetic coding
SOF_{14}	differential progressive DCT frame marker, arithmetic coding
SOF_{15}	differential lossless process frame marker, arithmetic coding
SOI	start-of-image marker
SOS	start-of-scan marker
SP	context-index for coding of first magnitude category when V is positive
Sq_{vu}	quantized DCT coefficient
SRL	shift right logical operation
SRL α β	logical shift right of α by β bits
Ss	start of spectral selection band in zig-zag sequence
SS	context-index for coding of sign decision
SSSS	4-bit size category of DC difference or AC coefficient amplitude
ST	stack counter
Switch_MPS	parameter controlling inversion of sense of MPS
Sz	parameter used in coding magnitude of V
S0	context-index for coding of V = 0 decision
t	summation index for parameter limits computation
T	temporary variable

Ta_j	AC entropy table destination selector for *j*th component in scan
Tb	arithmetic conditioning table destination identifier
Tc	Huffman coding or arithmetic coding table class
Td_j	DC entropy table destination selector for *j*th component in scan
TEM	temporary marker
Th	Huffman table destination identifier in DHT segment
Tq	quantization table destination identifier in DQT segment
Tq_i	quantization table destination selector for *i*th component in frame
U	DC and lossless coding conditioning upper bound parameter
V	symbol or value being either encoded or decoded
V_i	vertical sampling factor for *i*th component
$V_{i,j}$	*j*th value for length *i* in HUFFVAL
V_{max}	largest vertical sampling factor
V_t	temporary variable
VALPTR	list of indices for first value in HUFFVAL for each code length
V1	symbol value
V2	symbol value
x_i	number of columns in *i*th component
X	number of samples per line in component with largest horizontal dimension
X_i	*i*th statistics bin for coding magnitude category decision
X1, X2, X3, ... , X15	designation of context-indices for coding of magnitude categories in the arithmetic coding models
XHUFCO	extended Huffman code table
XHUFSI	table of sizes of extended Huffman codes
X'values'	values within the quotes are hexadecimal
y_i	number of lines in *i*th component
Y	number of lines in component with largest vertical dimension
ZRL	value in HUFFVAL assigned to run of 16 zero coefficients
ZZ(K)	*K*th element in zig-zag sequence of quantized DCT coefficients
ZZ(0)	quantized DC coefficient in zig-zag sequence order

4 General

The purpose of this clause is to give an informative overview of the elements specified in this Specification. Another purpose is to introduce many of the terms which are defined in clause 3. These terms are printed in *italics* upon first usage in this clause.

4.1 Elements specified in this Specification

There are three elements specified in this Specification:

a) An *encoder* is an embodiment of an *encoding process*. As shown in Figure 1, an encoder takes as input *digital source image data* and *table specifications*, and by means of a specified set of *procedures* generates as output *compressed image data*.

b) A *decoder* is an embodiment of a *decoding process*. As shown in Figure 2, a decoder takes as input compressed image data and table specifications, and by means of a specified set of procedures generates as output *digital reconstructed image data*.

c) The *interchange format*, shown in Figure 3, is a compressed image data representation which includes all table specifications used in the encoding process. The interchange format is for exchange between *application environments*.

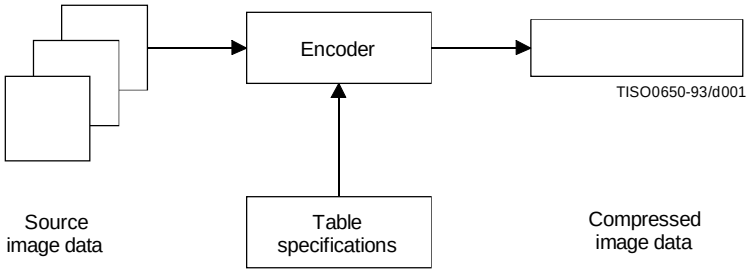


Figure 1 – Encoder

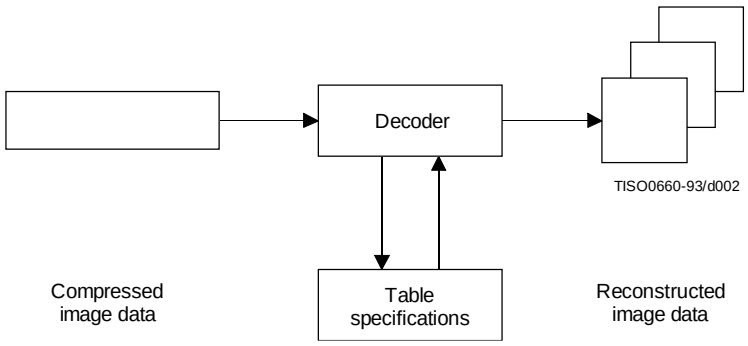


Figure 2 – Decoder

Figures 1 and 2 illustrate the general case for which the *continuous-tone* source and reconstructed image data consist of multiple *components*. (A *colour* image consists of multiple components; a *grayscale* image consists only of a single component.) A significant portion of this Specification is concerned with how to handle multiple-component images in a flexible, application-independent way.

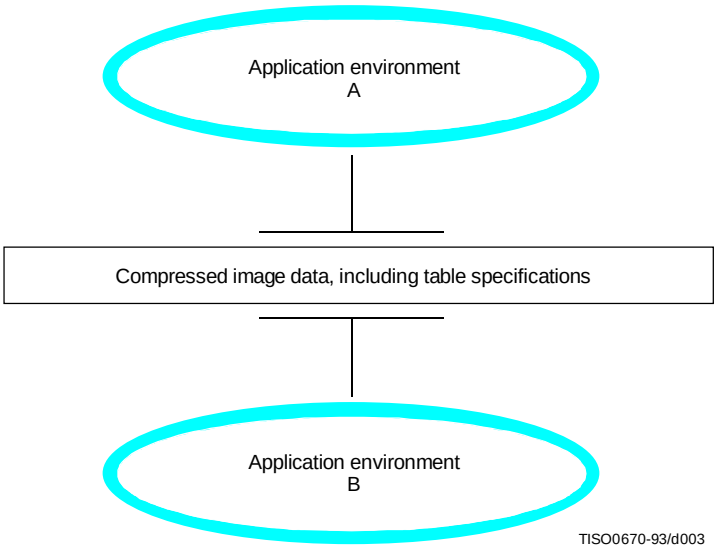


Figure 3 – Interchange format for compressed image data

These figures are also meant to show that the same tables specified for an encoder to use to compress a particular image must be provided to a decoder to reconstruct that image. However, this Specification does not specify how applications should associate tables with compressed image data, nor how they should represent source image data generally within their specific environments.

Consequently, this Specification also specifies the interchange format shown in Figure 3, in which table specifications are included within compressed image data. An image compressed with a specified encoding process within one application environment, A, is passed to a different environment, B, by means of the interchange format. The interchange format does not specify a complete coded image representation. Application-dependent information, e.g. colour space, is outside the scope of this Specification.

4.2 Lossy and lossless compression

This Specification specifies two *classes* of encoding and decoding processes, *lossy* and *lossless* processes. Those based on the *discrete cosine transform* (DCT) are lossy, thereby allowing substantial *compression* to be achieved while producing a reconstructed image with high visual fidelity to the encoder's source image.

The simplest DCT-based *coding process* is referred to as the *baseline sequential* process. It provides a capability which is sufficient for many applications. There are additional DCT-based processes which extend the baseline sequential process to a broader range of applications. In any decoder using *extended DCT-based decoding processes*, the baseline decoding process is required to be present in order to provide a default decoding capability.

The second class of coding processes is not based upon the DCT and is provided to meet the needs of applications requiring lossless compression. These lossless encoding and decoding processes are used independently of any of the DCT-based processes.

A table summarizing the relationship among these lossy and lossless coding processes is included in 4.11.

The amount of compression provided by any of the various processes is dependent on the characteristics of the particular image being compressed, as well as on the picture quality desired by the application and the desired speed of compression and decompression.

4.3 DCT-based coding

Figure 4 shows the main procedures for all encoding processes based on the DCT. It illustrates the special case of a single-component image; this is an appropriate simplification for overview purposes, because all processes specified in this Specification operate on each image component independently.

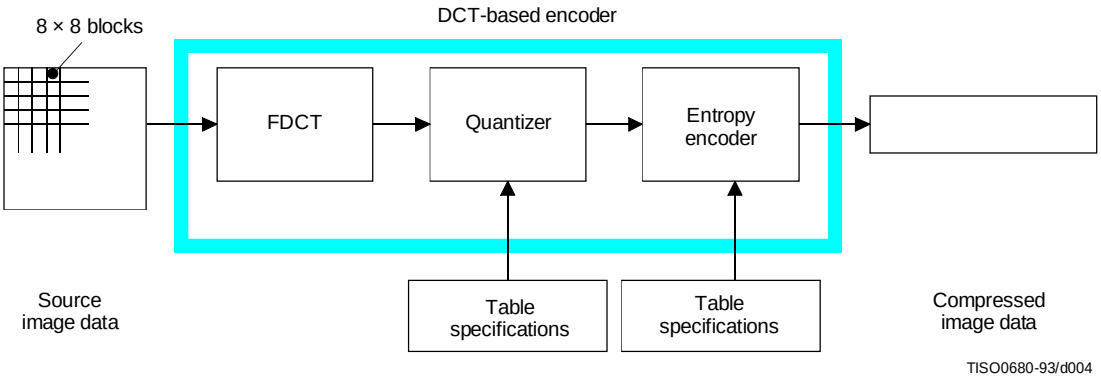


Figure 4 – DCT-based encoder simplified diagram

In the encoding process the input component's *samples* are grouped into 8 × 8 *blocks*, and each block is transformed by the *forward DCT* (FDCT) into a set of 64 values referred to as *DCT coefficients*. One of these values is referred to as the *DC coefficient* and the other 63 as the *AC coefficients*.

Each of the 64 coefficients is then *quantized* using one of 64 corresponding values from a *quantization table* (determined by one of the table specifications shown in Figure 4). No default values for quantization tables are specified in this Specification; applications may specify values which customize picture quality for their particular image characteristics, display devices, and viewing conditions.

After quantization, the DC coefficient and the 63 AC coefficients are prepared for *entropy encoding*, as shown in Figure 5. The previous quantized DC coefficient is used to predict the current quantized DC coefficient, and the difference is encoded. The 63 quantized AC coefficients undergo no such differential encoding, but are converted into a one-dimensional *zig-zag sequence*, as shown in Figure 5.

The quantized coefficients are then passed to an entropy encoding procedure which compresses the data further. One of two entropy coding procedures can be used, as described in 4.6. If *Huffman encoding* is used, *Huffman table* specifications must be provided to the encoder. If *arithmetic encoding* is used, arithmetic coding *conditioning table* specifications may be provided, otherwise the default conditioning table specifications shall be used.

Figure 6 shows the main procedures for all DCT-based decoding processes. Each step shown performs essentially the inverse of its corresponding main procedure within the encoder. The entropy decoder decodes the zig-zag sequence of quantized DCT coefficients. After *dequantization* the DCT coefficients are transformed to an 8 × 8 block of samples by the *inverse DCT* (IDCT).

4.4 Lossless coding

Figure 7 shows the main procedures for the lossless encoding processes. A *predictor* combines the reconstructed values of up to three neighbourhood samples at positions a, b, and c to form a prediction of the sample at position x as shown in Figure 8. This prediction is then subtracted from the actual value of the sample at position x, and the difference is losslessly entropy-coded by either Huffman or arithmetic coding.

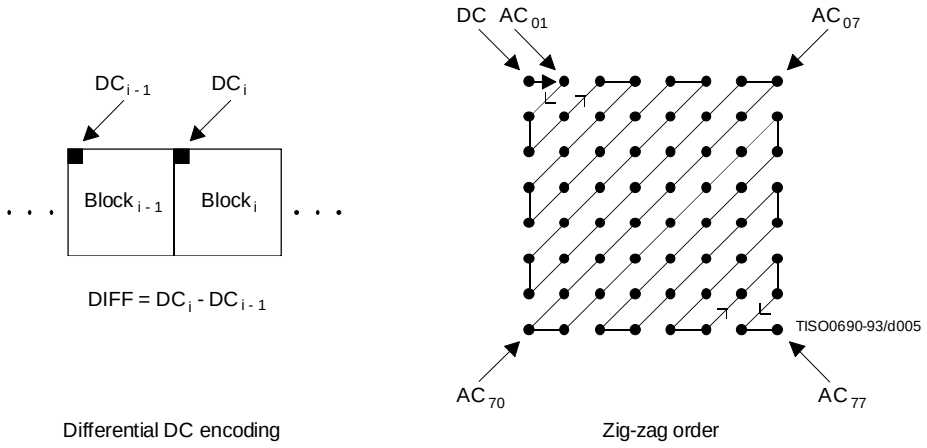


Figure 5 – Preparation of quantized coefficients for entropy encoding

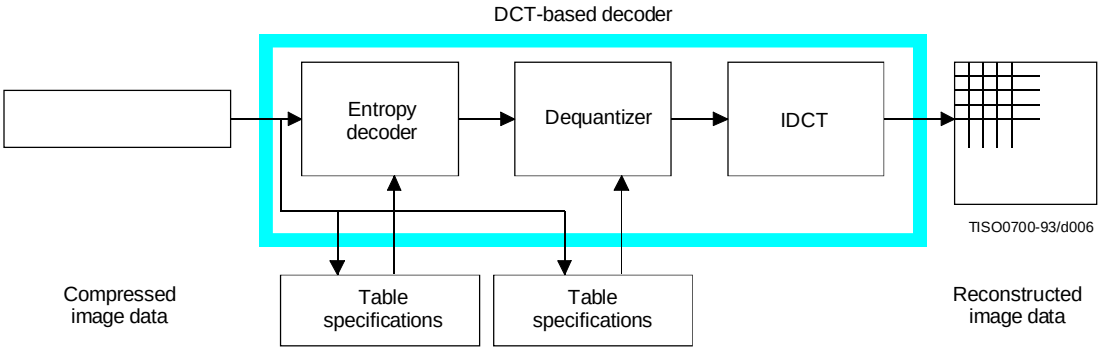


Figure 6 – DCT-based decoder simplified diagram

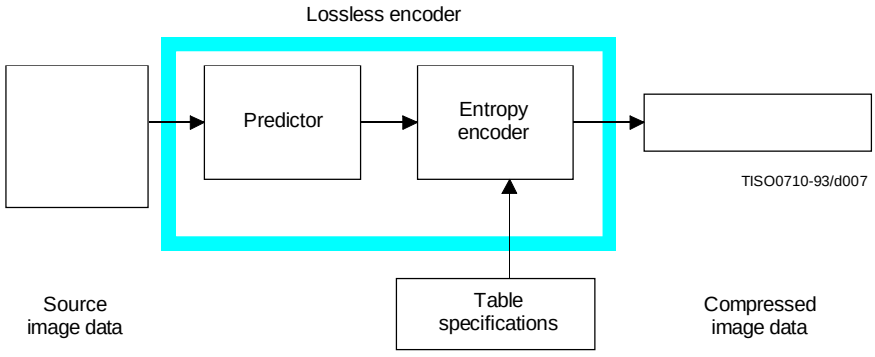


Figure 7 – Lossless encoder simplified diagram

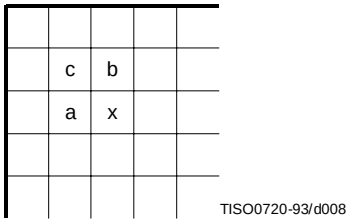


Figure 8 – 3-sample prediction neighbourhood

This encoding process may also be used in a slightly modified way, whereby the *precision* of the input samples is reduced by one or more bits prior to the lossless coding. This achieves higher compression than the lossless process (but lower compression than the DCT-based processes for equivalent visual fidelity), and limits the reconstructed image's worst-case sample error to the amount of input precision reduction.

4.5 Modes of operation

There are four distinct *modes of operation* under which the various coding processes are defined: *sequential DCT-based*, *progressive DCT-based*, lossless, and *hierarchical*. (Implementations are not required to provide all of these.) The lossless mode of operation was described in 4.4. The other modes of operation are compared as follows.

For the sequential DCT-based mode, 8×8 sample blocks are typically input block by block from left to right, and block-row by block-row from top to bottom. After a block has been transformed by the forward DCT, quantized and prepared for entropy encoding, all 64 of its quantized DCT coefficients can be immediately entropy encoded and output as part of the compressed image data (as was described in 4.3), thereby minimizing coefficient storage requirements.

For the progressive DCT-based mode, 8×8 blocks are also typically encoded in the same order, but in multiple *scans* through the image. This is accomplished by adding an image-sized coefficient memory buffer (not shown in Figure 4) between the quantizer and the entropy encoder. As each block is transformed by the forward DCT and quantized, its coefficients are stored in the buffer. The DCT coefficients in the buffer are then partially encoded in each of multiple scans. The typical sequence of image presentation at the output of the decoder for sequential versus progressive modes of operation is shown in Figure 9.

There are two procedures by which the quantized coefficients in the buffer may be partially encoded within a scan. First, only a specified *band* of coefficients from the zig-zag sequence need be encoded. This procedure is called *spectral selection*, because each band typically contains coefficients which occupy a lower or higher part of the *frequency* spectrum for that 8×8 block. Secondly, the coefficients within the current band need not be encoded to their full (quantized) accuracy within each scan. Upon a coefficient's first encoding, a specified number of most significant bits is encoded first. In subsequent scans, the less significant bits are then encoded. This procedure is called *successive approximation*. Either procedure may be used separately, or they may be mixed in flexible combinations.

In hierarchical mode, an image is encoded as a sequence of *frames*. These frames provide *reference reconstructed components* which are usually needed for prediction in subsequent frames. Except for the first frame for a given component, *differential frames* encode the difference between source components and reference reconstructed components. The coding of the differences may be done using only DCT-based processes, only lossless processes, or DCT-based processes with a final lossless process for each component. *Downsampling* and *upsampling filters* may be used to provide a pyramid of spatial resolutions as shown in Figure 10. Alternatively, the hierarchical mode can be used to improve the quality of the reconstructed components at a given spatial resolution.

Hierarchical mode offers a progressive presentation similar to the progressive DCT-based mode but is useful in environments which have multi-resolution requirements. Hierarchical mode also offers the capability of progressive coding to a final lossless stage.

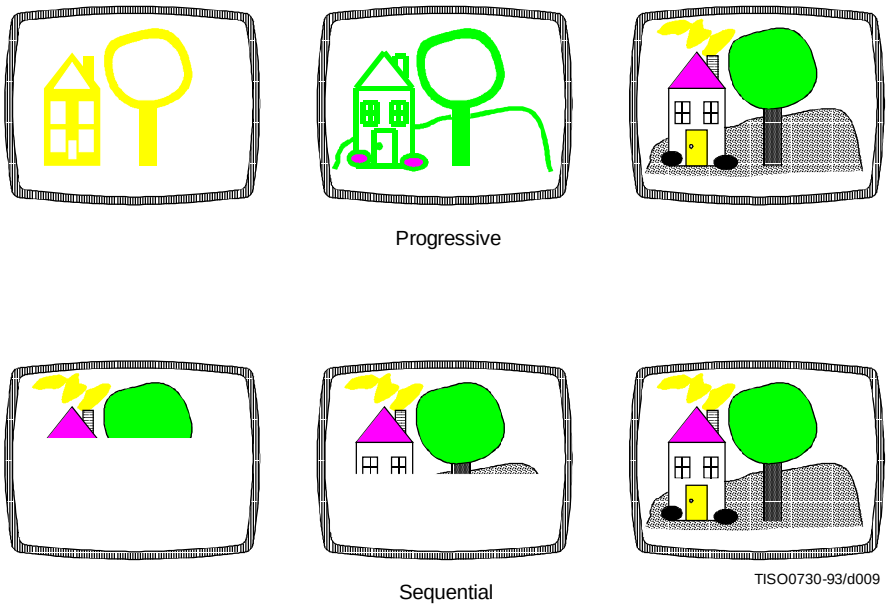


Figure 9 – Progressive versus sequential presentation

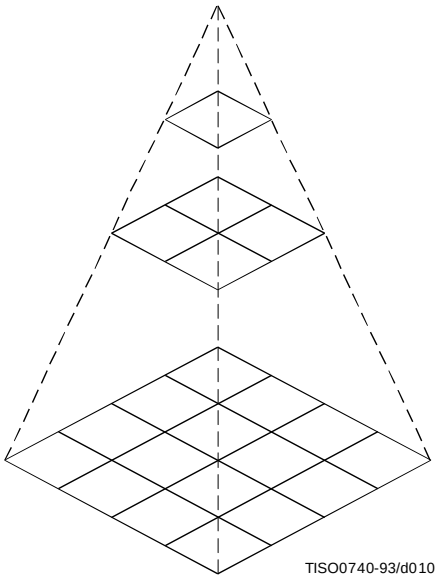


Figure 10 – Hierarchical multi-resolution encoding

4.6 Entropy coding alternatives

Two alternative entropy coding procedures are specified: Huffman coding and arithmetic coding. Huffman coding procedures use Huffman tables, determined by one of the table specifications shown in Figures 1 and 2. Arithmetic coding procedures use arithmetic coding conditioning tables, which may also be determined by a table specification. No default values for Huffman tables are specified, so that applications may choose tables appropriate for their own environments. Default tables are defined for the arithmetic coding conditioning.

The baseline sequential process uses Huffman coding, while the extended DCT-based and lossless processes may use either Huffman or arithmetic coding.

4.7 Sample precision

For DCT-based processes, two alternative sample precisions are specified: either 8 bits or 12 bits per sample. Applications which use samples with other precisions can use either 8-bit or 12-bit precision by shifting their source image samples appropriately. The baseline process uses only 8-bit precision. DCT-based implementations which handle 12-bit source image samples are likely to need greater computational resources than those which handle only 8-bit source images. Consequently in this Specification separate normative requirements are defined for 8-bit and 12-bit DCT-based processes.

For lossless processes the sample precision is specified to be from 2 to 16 bits.

4.8 Multiple-component control

Subclauses 4.3 and 4.4 give an overview of one major part of the encoding and decoding processes – those which operate on the sample values in order to achieve compression. There is another major part as well – the procedures which control the order in which the image data from multiple components are processed to create the compressed data, and which ensure that the proper set of table data is applied to the proper *data units* in the image. (A data unit is a sample for lossless processes and an 8×8 block of samples for DCT-based processes.)

4.8.1 Interleaving multiple components

Figure 11 shows an example of how an encoding process selects between multiple source image components as well as multiple sets of table data, when performing its encoding procedures. The source image in this example consists of the three components A, B and C, and there are two sets of table specifications. (This simplified view does not distinguish between the quantization tables and entropy coding tables.)

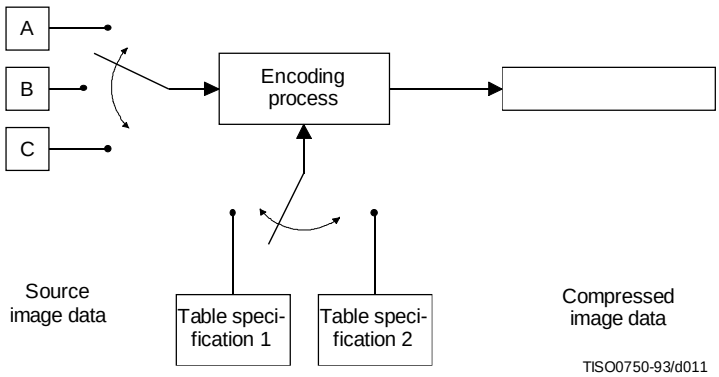


Figure 11 – Component-interleave and table-switching control

In sequential mode, encoding is *non-interleaved* if the encoder compresses all image data units in component A before beginning component B, and then in turn all of B before C. Encoding is *interleaved* if the encoder compresses a data unit from A, a data unit from B, a data unit from C, then back to A, etc. These alternatives are illustrated in Figure 12, which shows a case in which all three image components have identical dimensions: X *columns* by Y lines, for a total of n data units each.

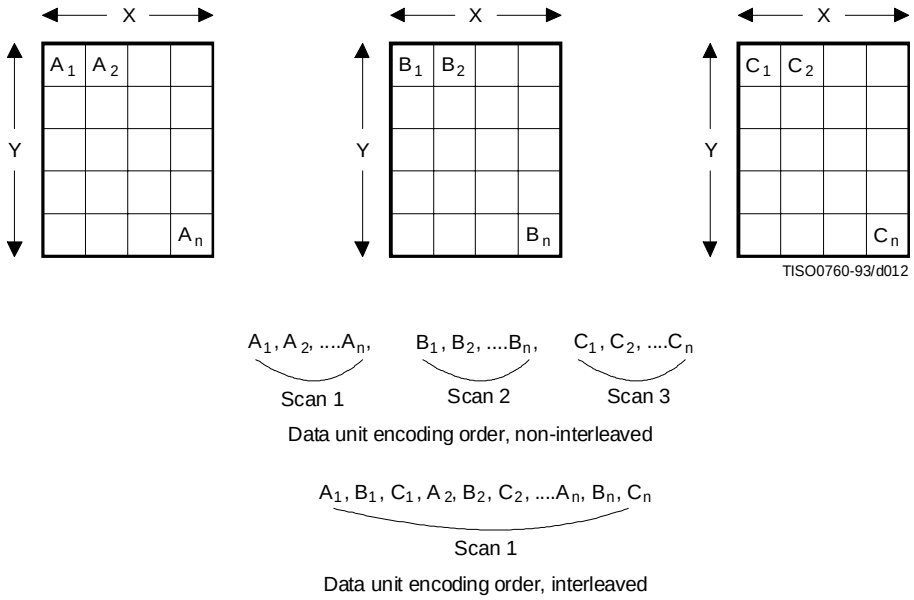


Figure 12 – Interleaved versus non-interleaved encoding order

These control procedures are also able to handle cases in which the source image components have different dimensions. Figure 13 shows a case in which two of the components, B and C, have half the number of horizontal samples relative to component A. In this case, two data units from A are interleaved with one each from B and C. Cases in which components of an image have more complex relationships, such as different horizontal and vertical dimensions, can be handled as well. (See Annex A.)

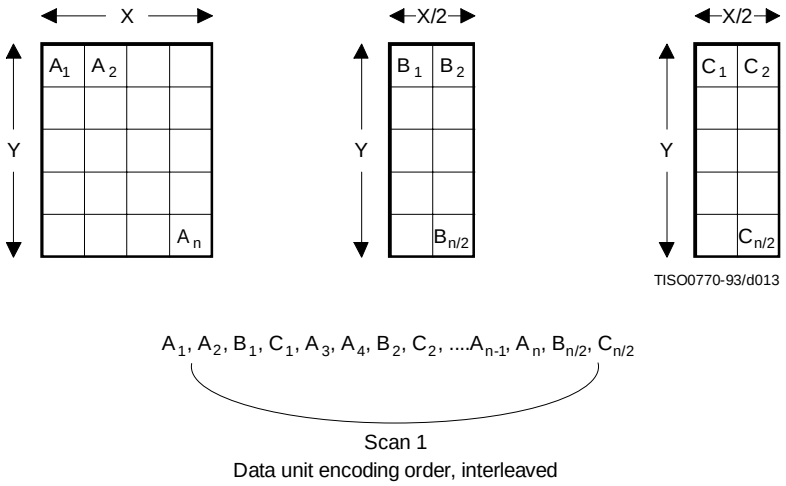


Figure 13 – Interleaved order for components with different dimensions

4.8.2 Minimum coded unit

Related to the concepts of multiple-component interleave is the *minimum coded unit* (MCU). If the compressed image data is non-interleaved, the MCU is defined to be one data unit. For example, in Figure 12 the MCU for the non-interleaved case is a single data unit. If the compressed data is interleaved, the MCU contains one or more data units from each component. For the interleaved case in Figure 12, the (first) MCU consists of the three interleaved data units A_1, B_1, C_1. In the example of Figure 13, the (first) MCU consists of the four data units A_1, A_2 , B_1, C_1.

4.9 Structure of compressed data

Figures 1, 2, and 3 all illustrate slightly different views of compressed image data. Figure 1 shows this data as the output of an encoding process, Figure 2 shows it as the input to a decoding process, and Figure 3 shows compressed image data in the interchange format, at the interface between applications.

Compressed image data are described by a uniform structure and set of *parameters* for both classes of encoding processes (lossy or lossless), and for all modes of operation (sequential, progressive, lossless, and hierarchical). The various parts of the compressed image data are identified by special two-byte codes called *markers*. Some markers are followed by particular sequences of parameters, as in the case of table specifications, *frame header*, or *scan header*. Others are used without parameters for functions such as marking the start-of-image and end-of-image. When a marker is associated with a particular sequence of parameters, the marker and its parameters comprise a *marker segment*.

The data created by the entropy encoder are also segmented, and one particular marker – *the restart marker* – is used to isolate *entropy-coded data segments*. The encoder outputs the restart markers, intermixed with the entropy-coded data, at regular *restart intervals* of the source image data. Restart markers can be identified without having to decode the compressed data to find them. Because they can be independently decoded, they have application-specific uses, such as parallel encoding or decoding, isolation of data corruptions, and semi-random access of entropy-coded segments.

There are three compressed data formats:

a) the interchange format;

b) the *abbreviated format* for compressed image data;

c) the abbreviated format for table-specification data.

4.9.1 Interchange format

In addition to certain required marker segments and the entropy-coded segments, the interchange format shall include the marker segments for all quantization and entropy-coding table specifications needed by the decoding process. This guarantees that a compressed image can cross the boundary between application environments, regardless of how each environment internally associates tables with compressed image data.

4.9.2 Abbreviated format for compressed image data

The abbreviated format for compressed image data is identical to the interchange format, except that it does not include all tables required for decoding. (It may include some of them.) This format is intended for use within applications where alternative mechanisms are available for supplying some or all of the table-specification data needed for decoding.

4.9.3 Abbreviated format for table-specification data

This format contains only table-specification data. It is a means by which the application may install in the decoder the tables required to subsequently reconstruct one or more images.

4.10 Image, frame, and scan

Compressed image data consists of only one image. An image contains only one frame in the cases of sequential and progressive coding processes; an image contains multiple frames for the hierarchical mode.

A frame contains one or more scans. For sequential processes, a scan contains a complete encoding of one or more image components. In Figures 12 and 13, the frame consists of three scans when non-interleaved, and one scan if all three components are interleaved together. The frame could also consist of two scans: one with a non-interleaved component, the other with two components interleaved.

For progressive processes, a scan contains a partial encoding of all data units from one or more image components. Components shall not be interleaved in progressive mode, except for the DC coefficients in the first scan for each component of a progressive frame.

4.11 Summary of coding processes

Table 1 provides a summary of the essential characteristics of the various coding processes specified in this Specification. The full specification of these processes is contained in Annexes F, G, H, and J.

Table 1 – Summary: Essential characteristics of coding processes

Baseline process (required for all DCT-based decoders)
• DCT-based process • Source image: 8-bit samples within each component • Sequential • Huffman coding: 2 AC and 2 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Extended DCT-based processes
• DCT-based process • Source image: 8-bit or 12-bit samples • Sequential or progressive • Huffman or arithmetic coding: 4 AC and 4 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Lossless processes
• Predictive process (not DCT-based) • Source image: P-bit samples ($2 \leq P \leq 16$) • Sequential • Huffman or arithmetic coding: 4 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Hierarchical processes
• Multiple frames (non-differential and differential) • Uses extended DCT-based or lossless processes • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

5 Interchange format requirements

The interchange format is the coded representation of compressed image data for exchange between application environments.

The interchange format requirements are that any compressed image data represented in interchange format shall comply with the syntax and code assignments appropriate for the decoding process selected, as specified in Annex B.

Tests for whether compressed image data comply with these requirements are specified in Part 2 of this Specification.

6 Encoder requirements

An encoding process converts source image data to compressed image data. Each of Annexes F, G, H, and J specifies a number of distinct encoding processes for its particular mode of operation.

An encoder is an embodiment of one (or more) of the encoding processes specified in Annexes F, G, H, or J. In order to comply with this Specification, an encoder shall satisfy at least one of the following two requirements.

An encoder shall

a) with appropriate accuracy, convert source image data to compressed image data which comply with the interchange format syntax specified in Annex B for the encoding process(es) embodied by the encoder;

b) with appropriate accuracy, convert source image data to compressed image data which comply with the abbreviated format for compressed image data syntax specified in Annex B for the encoding process(es) embodied by the encoder.

For each of the encoding processes specified in Annexes F, G, H, and J, the compliance tests for the above requirements are specified in Part 2 of this Specification.

NOTE – There is **no requirement** in this Specification that any encoder which embodies one of the encoding processes specified in Annexes F, G, H, or J shall be able to operate for all ranges of the parameters which are allowed for that process. An encoder is only required to meet the compliance tests specified in Part 2, and to generate the compressed data format according to Annex B for those parameter values which it does use.

7 Decoder requirements

A decoding process converts compressed image data to reconstructed image data. Each of Annexes F, G, H, and J specifies a number of distinct decoding processes for its particular mode of operation.

A decoder is an embodiment of one (or more) of the decoding processes specified in Annexes F, G, H, or J. In order to comply with this Specification, a decoder shall satisfy all three of the following requirements.

A decoder shall

a) with appropriate accuracy, convert to reconstructed image data any compressed image data with parameters within the range supported by the application, and which comply with the interchange format syntax specified in Annex B for the decoding process(es) embodied by the decoder;

b) accept and properly store any table-specification data which comply with the abbreviated format for table-specification data syntax specified in Annex B for the decoding process(es) embodied by the decoder;

c) with appropriate accuracy, convert to reconstructed image data any compressed image data which comply with the abbreviated format for compressed image data syntax specified in Annex B for the decoding process(es) embodied by the decoder, provided that the table-specification data required for decoding the compressed image data has previously been installed into the decoder.

Additionally, any DCT-based decoder, if it embodies any DCT-based decoding process other than baseline sequential, shall also embody the baseline sequential decoding process.

For each of the decoding processes specified in Annexes F, G, H, and J, the compliance tests for the above requirements are specified in Part 2 of this Specification.

Annex A

Mathematical definitions

(This annex forms an integral part of this Recommendation | International Standard)

A.1 Source image

Source images to which the encoding processes specified in this Specification can be applied are defined in this annex.

A.1.1 Dimensions and sampling factors

As shown in Figure A.1, a source image is defined to consist of Nf components. Each component, with unique identifier C_i, is defined to consist of a rectangular array of samples of x_i columns by y_i lines. The component dimensions are derived from two parameters, X and Y, where X is the maximum of the x_i values and Y is the maximum of the y_i values for all components in the frame. For each component, sampling factors H_i and V_i are defined relating component dimensions x_i and y_i to maximum dimensions X and Y, according to the following expressions:

$$x_i = \left\lceil X \times \frac{H_i}{H_{max}} \right\rceil \text{ and } y_i \left\lceil Y \times \frac{V_i}{V_{max}} \right\rceil,$$

where H_{max} and V_{max} are the maximum sampling factors for all components in the frame, and $\lceil\ \rceil$ is the ceiling function.

As an example, consider an image having 3 components with maximum dimensions of 512 lines and 512 samples per line, and with the following sampling factors:

Component 0	$H_0 = 4, \quad V_0 = 1$
Component 1	$H_1 = 2, \quad V_1 = 2$
Component 2	$H_2 = 1, \quad V_2 = 1$

Then $X = 512$, $Y = 512$, $H_{max} = 4$, $V_{max} = 2$, and x_i and y_i for each component are

Component 0	$x_0 = 512, \quad y_0 = 256$
Component 1	$x_1 = 256, \quad y_1 = 512$
Component 2	$x_2 = 128, \quad y_2 = 256$

NOTE – The X, Y, H_i, and V_i parameters are contained in the frame header of the compressed image data (see B.2.2), whereas the individual component dimensions x_i and y_i are derived by the decoder. Source images with x_i and y_i dimensions which do not satisfy the expressions above cannot be properly reconstructed.

A.1.2 Sample precision

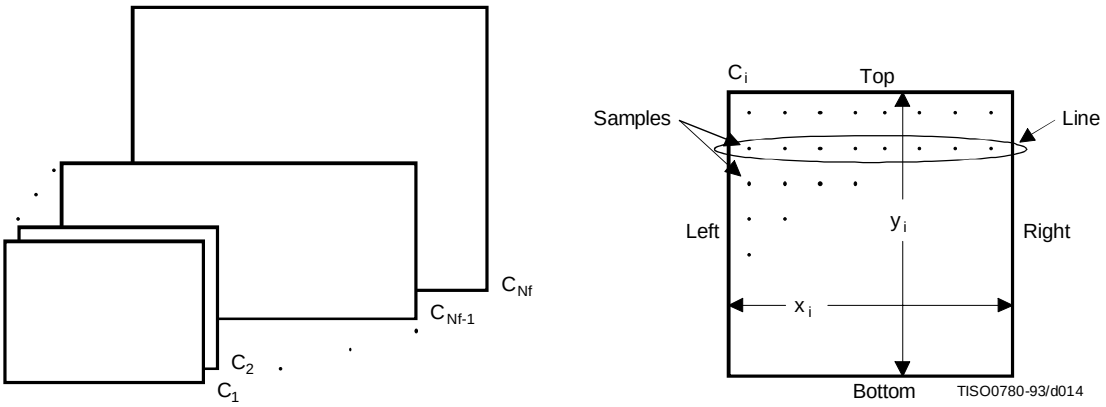
A sample is an integer with precision P bits, with any value in the range 0 through 2^{P-1}. All samples of all components within an image shall have the same precision P. Restrictions on the value of P depend on the mode of operation, as specified in B.2 to B.7.

A.1.3 Data unit

A data unit is a sample in lossless processes and an 8×8 block of contiguous samples in DCT-based processes. The left-most 8 samples of each of the top-most 8 rows in the component shall always be the top-left-most block. With this top-left-most block as the reference, the component is partitioned into contiguous data units to the right and to the bottom (as shown in Figure A.4).

A.1.4 Orientation

Figure A.1 indicates the orientation of an image component by the terms top, bottom, left, and right. The order by which the data units of an image component are input to the compression encoding procedures is defined to be left-to-right and top-to-bottom within the component. (This ordering is precisely defined in A.2.) Applications determine which edges of a source image are defined as top, bottom, left, and right.



a) Source image with multiple components

b) Characteristics of an image component

Figure A.1 – Source image characteristics

A.2 Order of source image data encoding

The scan header (see B.2.3) specifies the order by which source image data units shall be encoded and placed within the compressed image data. For a given scan, if the scan header parameter Ns = 1, then data from only one source component – the component specified by parameter Cs_1 – shall be present within the scan. This data is non-interleaved by definition. If Ns > 1, then data from the Ns components Cs_1 through Cs_{Ns} shall be present within the scan. This data shall always be interleaved. The order of components in a scan shall be according to the order specified in the frame header.

The ordering of data units and the construction of minimum coded units (MCU) is defined as follows.

A.2.1 Minimum coded unit (MCU)

For non-interleaved data the MCU is one data unit. For interleaved data the MCU is the sequence of data units defined by the sampling factors of the components in the scan.

A.2.2 Non-interleaved order (Ns = 1)

When Ns = 1 (where Ns is the number of components in a scan), the order of data units within a scan shall be left-to-right and top-to-bottom, as shown in Figure A.2. This ordering applies whenever Ns = 1, regardless of the values of H_1 and V_1.

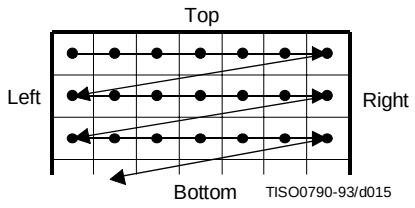
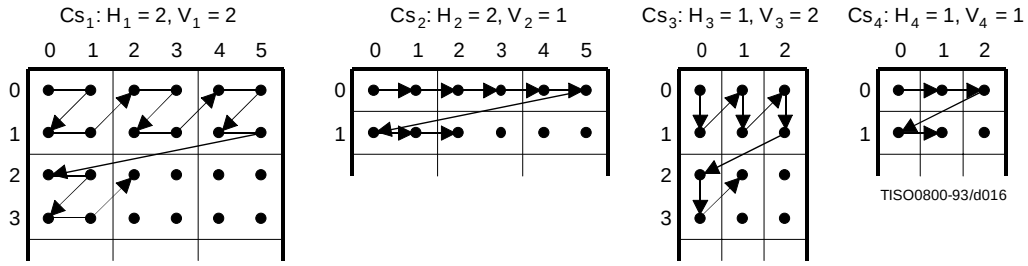


Figure A.2 – Non-interleaved data ordering

A.2.3 Interleaved order (Ns > 1)

When Ns > 1, each scan component Cs_i is partitioned into small rectangular arrays of H_k horizontal data units by V_k vertical data units. The subscripts k indicate that H_k and V_k are from the position in the frame header component-specification for which $C_k = Cs_i$. Within each H_k by V_k array, data units are ordered from left-to-right and top-to-bottom. The arrays in turn are ordered from left-to-right and top-to-bottom within each component.

As shown in the example of Figure A.3, Ns = 4, and MCU_1 consists of data units taken first from the top-left-most region of Cs_1, followed by data units from the corresponding region of Cs_2, then from Cs_3 and then from Cs_4. MCU_2 follows the same ordering for data taken from the next region to the right for the four components.



$$MCU_1 = d^1_{00}\ d^1_{01}\ d^1_{10}\ d^1_{11}\ d^2_{00}\ d^2_{01}\ d^3_{00}\ d^3_{10}\ d^4_{00},$$
$$MCU_2 = d^1_{02}\ d^1_{03}\ d^1_{12}\ d^1_{13}\ d^2_{02}\ d^2_{03}\ d^3_{01}\ d^3_{11}\ d^4_{01},$$
$$MCU_3 = d^1_{04}\ d^1_{05}\ d^1_{14}\ d^1_{15}\ d^2_{04}\ d^2_{05}\ d^3_{02}\ d^3_{12}\ d^4_{02},$$
$$MCU_4 = d^1_{20}\ d^1_{21}\ d^1_{30}\ d^1_{31}\ d^2_{10}\ d^2_{11}\ d^3_{20}\ d^3_{30}\ d^4_{10},$$

Cs_1 data units Cs_2 Cs_3 Cs_4

Figure A.3 – Interleaved data ordering example

A.2.4 Completion of partial MCU

For DCT-based processes the data unit is a block. If x_i is not a multiple of 8, the encoding process shall extend the number of columns to complete the right-most sample blocks. If the component is to be interleaved, the encoding process shall also extend the number of samples by one or more additional blocks, if necessary, so that the number of blocks is an integer multiple of H_i. Similarly, if y_i is not a multiple of 8, the encoding process shall extend the number of lines to complete the bottom-most block-row. If the component is to be interleaved, the encoding process shall also extend the number of lines by one or more additional block-rows, if necessary, so that the number of block-rows is an integer multiple of V_i.

NOTE – It is recommended that any incomplete MCUs be completed by replication of the right-most column and the bottom line of each component.

For lossless processes the data unit is a sample. If the component is to be interleaved, the encoding process shall extend the number of samples, if necessary, so that the number is a multiple of H_i. Similarly, the encoding process shall extend the number of lines, if necessary, so that the number of lines is a multiple of V_i.

Any sample added by an encoding process to complete partial MCUs shall be removed by the decoding process.

A.3 DCT compression

A.3.1 Level shift

Before a non-differential frame encoding process computes the FDCT for a block of source image samples, the samples shall be level shifted to a signed representation by subtracting 2^{P-1}, where P is the precision parameter specified in B.2.2. Thus, when P = 8, the level shift is by 128; when P = 12, the level shift is by 2048.

After a non-differential frame decoding process computes the IDCT and produces a block of reconstructed image samples, an inverse level shift shall restore the samples to the unsigned representation by adding 2^{P-1} and clamping the results to the range 0 to 2^{P-1}.

A.3.2 Orientation of samples for FDCT computation

Figure A.4 shows an image component which has been partitioned into 8×8 blocks for the FDCT computations. Figure A.4 also defines the orientation of the samples within a block by showing the indices used in the FDCT equation of A.3.3.

The definitions of block partitioning and sample orientation also apply to any DCT decoding process and the output reconstructed image. Any sample added by an encoding process to complete partial MCUs shall be removed by the decoding process.

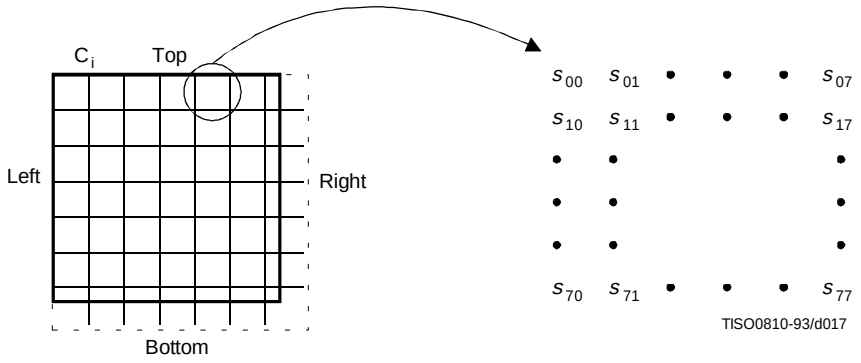


Figure A.4 – Partition and orientation of 8 x 8 sample blocks

A.3.3 FDCT and IDCT (informative)

The following equations specify the ideal functional definition of the FDCT and the IDCT.

NOTE – These equations contain terms which cannot be represented with perfect accuracy by any real implementation. The accuracy requirements for the combined FDCT and quantization procedures are specified in Part 2 of this Specification. The accuracy requirements for the combined dequantization and IDCT procedures are also specified in Part 2 of this Specification.

FDCT:
$$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

IDCT:
$$s_{yx} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v S_{vu} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

where

$$C_u, C_v = 1/\sqrt{2} \text{ for } u, v = 0$$

$$C_u, C_v = 1 \text{ otherwise}$$

otherwise.

A.3.4 DCT coefficient quantization (informative) and dequantization (normative)

After the FDCT is computed for a block, each of the 64 resulting DCT coefficients is quantized by a uniform quantizer. The quantizer step size for each coefficient S_{vu} is the value of the corresponding element Q_{vu} from the quantization table specified by the frame parameter Tq_i (see B.2.2).

The uniform quantizer is defined by the following equation. Rounding is to the nearest integer:

$$Sq_{vu} = round\left(\frac{S_{vu}}{Q_{vu}}\right)$$

Sq_{vu} is the quantized DCT coefficient, normalized by the quantizer step size.

NOTE – This equation contains a term which may not be represented with perfect accuracy by any real implementation. The accuracy requirements for the combined FDCT and quantization procedures are specified in Part 2 of this Specification.

At the decoder, this normalization is removed by the following equation, which defines dequantization:

$$R_{vu} = Sq_{vu} \times Q_{vu}$$

NOTE – Depending on the rounding used in quantization, it is possible that the dequantized coefficient may be outside the expected range.

The relationship among samples, DCT coefficients, and quantization is illustrated in Figure A.5.

A.3.5 Differential DC encoding

After quantization, and in preparation for entropy encoding, the quantized DC coefficient Sq_{00} is treated separately from the 63 quantized AC coefficients. The value that shall be encoded is the difference (DIFF) between the quantized DC coefficient of the current block (DC_i which is also designated as Sq_{00}) and that of the previous block of the same component (PRED):

$$DIFF = DC_i - PRED$$

A.3.6 Zig-zag sequence

After quantization, and in preparation for entropy encoding, the quantized AC coefficients are converted to the zig-zag sequence. The quantized DC coefficient (coefficient zero in the array) is treated separately, as defined in A.3.5. The zig-zag sequence is specified in Figure A.6.

A.4 Point transform

For various procedures data may be optionally divided by a power of 2 by a point transform prior to coding. There are three processes which require a point transform: lossless coding, lossless differential frame coding in the hierarchical mode, and successive approximation coding in the progressive DCT mode.

In the lossless mode of operation the point transform is applied to the input samples. In the difference coding of the hierarchical mode of operation the point transform is applied to the difference between the input component samples and the reference component samples. In both cases the point transform is an integer divide by 2^{Pt}, where Pt is the value of the point transform parameter (see B.2.3).

In successive approximation coding the point transform for the AC coefficients is an integer divide by 2^{Al}, where Al is the successive approximation bit position, low (see B.2.3). The point transform for the DC coefficients is an arithmetic-shift-right by Al bits. This is equivalent to dividing by 2^{Pt} before the level shift (see A.3.1).

The output of the decoder is rescaled by multiplying by 2^{Pt}. An example of the point transform is given in K.10.

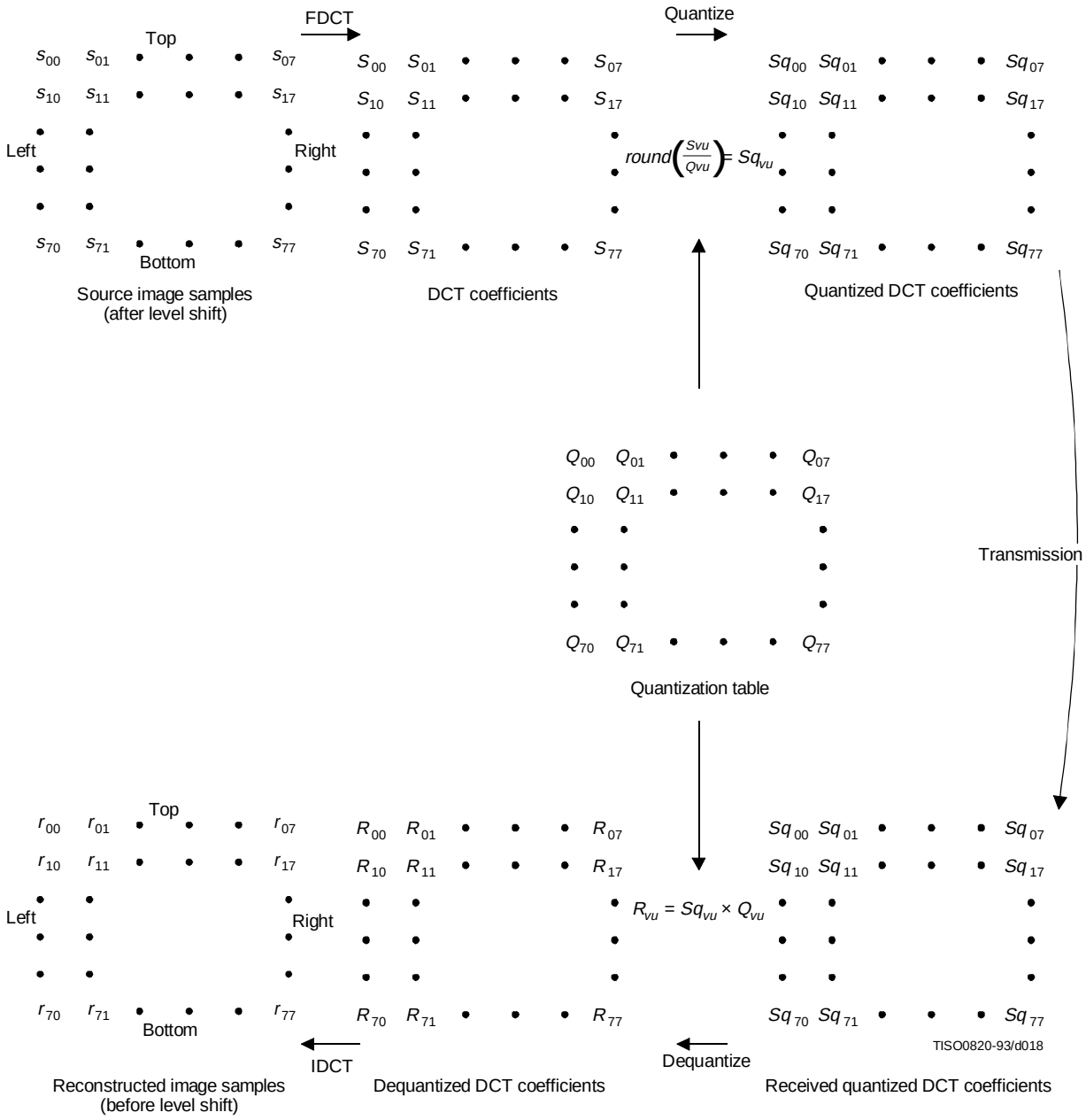


Figure A.5 – Relationship between 8 × 8-block samples and DCT coefficients

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Figure A.6 – Zig-zag sequence of quantized DCT coefficients

A.5 Arithmetic procedures in lossless and hierarchical modes of operation

In the lossless mode of operation predictions are calculated with full precision and without clamping of either overflow or underflow beyond the range of values allowed by the precision of the input. However, the division by two which is part of some of the prediction calculations shall be approximated by an arithmetic-shift-right by one bit.

The two's complement differences which are coded in either the lossless mode of operation or the differential frame coding in the hierarchical mode of operation are calculated modulo 65 536, thereby restricting the precision of these differences to a maximum of 16 bits. The modulo values are calculated by performing the logical AND operation of the two's complement difference with X'FFFF'. For purposes of coding, the result is still interpreted as a 16 bit two's complement difference. Modulo 65 536 arithmetic is also used in the decoder in calculating the output from the sum of the prediction and this two's complement difference.

Annex B

Compressed data formats

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies three compressed data formats:

a) the interchange format, specified in B.2 and B.3;
b) the abbreviated format for compressed image data, specified in B.4;
c) the abbreviated format for table-specification data, specified in B.5.

B.1 describes the constituent parts of these formats. B.1.3 and B.1.4 give the conventions for symbols and figures used in the format specifications.

B.1 General aspects of the compressed data format specifications

Structurally, the compressed data formats consist of an ordered collection of parameters, markers, and entropy-coded data segments. Parameters and markers in turn are often organized into marker segments. Because all of these constituent parts are represented with byte-aligned codes, each compressed data format consists of an ordered sequence of 8-bit bytes. For each byte, a most significant bit (MSB) and a least significant bit (LSB) are defined.

B.1.1 Constituent parts

This subclause gives a general description of each of the constituent parts of the compressed data format.

B.1.1.1 Parameters

Parameters are integers, with values specific to the encoding process, source image characteristics, and other features selectable by the application. Parameters are assigned either 4-bit, 1-byte, or 2-byte codes. Except for certain optional groups of parameters, parameters encode critical information without which the decoding process cannot properly reconstruct the image.

The code assignment for a parameter shall be an unsigned integer of the specified length in bits with the particular value of the parameter.

For parameters which are 2 bytes (16 bits) in length, the most significant byte shall come first in the compressed data's ordered sequence of bytes. Parameters which are 4 bits in length always come in pairs, and the pair shall always be encoded in a single byte. The first 4-bit parameter of the pair shall occupy the most significant 4 bits of the byte. Within any 16-, 8-, or 4-bit parameter, the MSB shall come first and LSB shall come last.

B.1.1.2 Markers

Markers serve to identify the various structural parts of the compressed data formats. Most markers start marker segments containing a related group of parameters; some markers stand alone. All markers are assigned two-byte codes: an X'FF' byte followed by a byte which is not equal to 0 or X'FF' (see Table B.1). Any marker may optionally be preceded by any number of fill bytes, which are bytes assigned code X'FF'.

NOTE – Because of this special code-assignment structure, markers make it possible for a decoder to parse the compressed data and locate its various parts without having to decode other segments of image data.

B.1.1.3 Marker assignments

All markers shall be assigned two-byte codes: a X'FF' byte followed by a second byte which is not equal to 0 or X'FF'. The second byte is specified in Table B.1 for each defined marker. An asterisk (*) indicates a marker which stands alone, that is, which is not the start of a marker segment.

Table B.1 – Marker code assignments

Code Assignment	Symbol	Description
Start Of Frame markers, non-differential, Huffman coding		
X'FFC0' X'FFC1' X'FFC2' X'FFC3'	SOF_0 SOF_1 SOF_2 SOF_3	Baseline DCT Extended sequential DCT Progressive DCT Lossless (sequential)
Start Of Frame markers, differential, Huffman coding		
X'FFC5' X'FFC6' X'FFC7'	SOF_5 SOF_6 SOF_7	Differential sequential DCT Differential progressive DCT Differential lossless (sequential)
Start Of Frame markers, non-differential, arithmetic coding		
X'FFC8' X'FFC9' X'FFCA' X'FFCB'	JPG SOF_9 SOF_{10} SOF_{11}	Reserved for JPEG extensions Extended sequential DCT Progressive DCT Lossless (sequential)
Start Of Frame markers, differential, arithmetic coding		
X'FFCD' X'FFCE' X'FFCF'	SOF_{13} SOF_{14} SOF_{15}	Differential sequential DCT Differential progressive DCT Differential lossless (sequential)
Huffman table specification		
X'FFC4'	DHT	Define Huffman table(s)
Arithmetic coding conditioning specification		
X'FFCC'	DAC	Define arithmetic coding conditioning(s)
Restart interval termination		
X'FFD0' through X'FFD7'	RST_m*	Restart with modulo 8 count "m"
Other markers		
X'FFD8' X'FFD9' X'FFDA' X'FFDB' X'FFDC' X'FFDD' X'FFDE' X'FFDF' X'FFE0' through X'FFEF' X'FFF0' through X'FFFD' X'FFFE'	SOI* EOI* SOS DQT DNL DRI DHP EXP APP_n JPG_n COM	Start of image End of image Start of scan Define quantization table(s) Define number of lines Define restart interval Define hierarchical progression Expand reference component(s) Reserved for application segments Reserved for JPEG extensions Comment
Reserved markers		
X'FF01' X'FF02' through X'FFBF'	TEM* RES	For temporary private use in arithmetic coding Reserved

B.1.1.4 Marker segments

A marker segment consists of a marker followed by a sequence of related parameters. The first parameter in a marker segment is the two-byte length parameter. This length parameter encodes the number of bytes in the marker segment, including the length parameter and excluding the two-byte marker. The marker segments identified by the SOF and SOS marker codes are referred to as headers: the frame header and the scan header respectively.

B.1.1.5 Entropy-coded data segments

An entropy-coded data segment contains the output of an entropy-coding procedure. It consists of an integer number of bytes, whether the entropy-coding procedure used is Huffman or arithmetic.

NOTES

1 Making entropy-coded segments an integer number of bytes is performed as follows: for Huffman coding, 1-bits are used, if necessary, to pad the end of the compressed data to complete the final byte of a segment. For arithmetic coding, byte alignment is performed in the procedure which terminates the entropy-coded segment (see D.1.8).

2 In order to ensure that a marker does not occur within an entropy-coded segment, any X'FF' byte generated by either a Huffman or arithmetic encoder, or an X'FF' byte that was generated by the padding of 1-bits described in NOTE 1 above, is followed by a "stuffed" zero byte (see D.1.6 and F.1.2.3).

B.1.2 Syntax

In B.2 and B.3 the interchange format syntax is specified. For the purposes of this Specification, the syntax specification consists of:

- the required ordering of markers, parameters, and entropy-coded segments;
- identification of optional or conditional constituent parts;
- the name, symbol, and definition of each marker and parameter;
- the allowed values of each parameter;
- any restrictions on the above which are specific to the various coding processes.

The ordering of constituent parts and the identification of which are optional or conditional is specified by the syntax figures in B.2 and B.3. Names, symbols, definitions, allowed values, conditions, and restrictions are specified immediately below each syntax figure.

B.1.3 Conventions for syntax figures

The syntax figures in B.2 and B.3 are a part of the interchange format specification. The following conventions, illustrated in Figure B.1, apply to these figures:

- **parameter/marker indicator:** A thin-lined box encloses either a marker or a single parameter;
- **segment indicator:** A thick-lined box encloses either a marker segment, an entropy-coded data segment, or combinations of these;
- **parameter length indicator:** The width of a thin-lined box is proportional to the parameter length (4, 8, or 16 bits, shown as E, B, and D respectively in Figure B.1) of the marker or parameter it encloses; the width of thick-lined boxes is not meaningful;
- **optional/conditional indicator:** Square brackets indicate that a marker or marker segment is only optionally or conditionally present in the compressed image data;
- **ordering:** In the interchange format a parameter or marker shown in a figure precedes all of those shown to its right, and follows all of those shown to its left;
- **entropy-coded data indicator:** Angled brackets indicate that the entity enclosed has been entropy encoded.

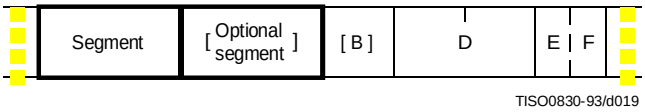


Figure B.1 – Syntax notation conventions

B.1.4 Conventions for symbols, code lengths, and values

Following each syntax figure in B.2 and B.3, the symbol, name, and definition for each marker and parameter shown in the figure are specified. For each parameter, the length and allowed values are also specified in tabular form.

The following conventions apply to symbols for markers and parameters:

- all marker symbols have three upper-case letters, and some also have a subscript. Examples: SOI, SOF_n;
- all parameter symbols have one upper-case letter; some also have one lower-case letter and some have subscripts. Examples: Y, Nf, H_i, Tq_i.

B.2 General sequential and progressive syntax

This clause specifies the interchange format syntax which applies to all coding processes for sequential DCT-based, progressive DCT-based, and lossless modes of operation.

B.2.1 High-level syntax

Figure B.2 specifies the order of the high-level constituent parts of the interchange format for all non-hierarchical encoding processes specified in this Specification.

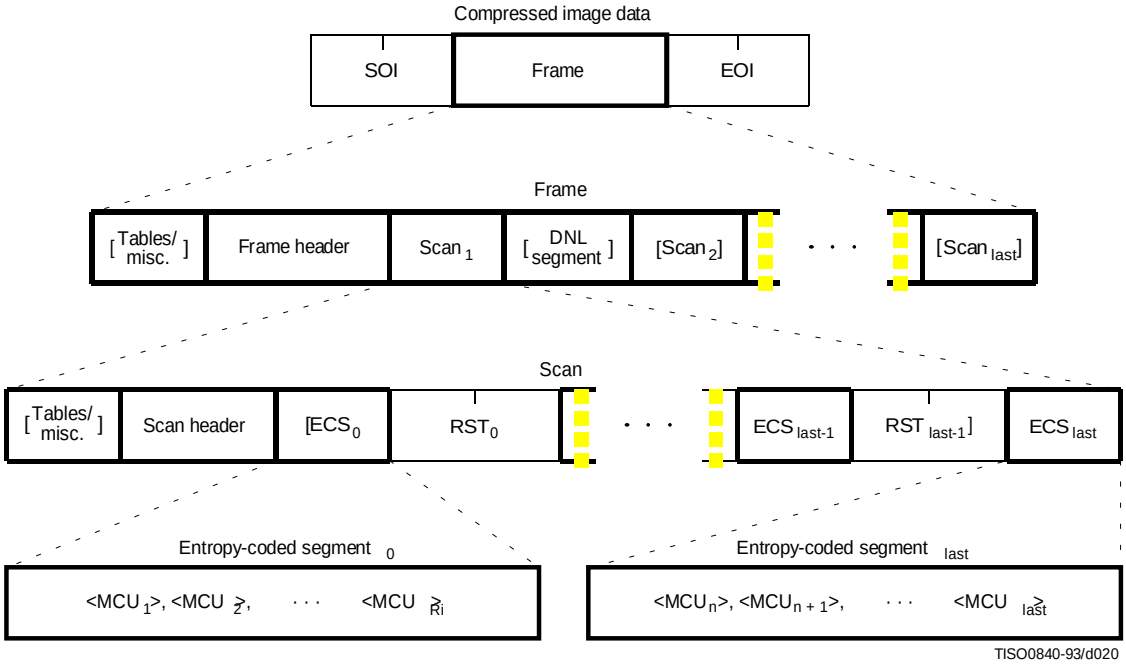


Figure B.2 – Syntax for sequential DCT-based, progressive DCT-based, and lossless modes of operation

The three markers shown in Figure B.2 are defined as follows:

SOI: Start of image marker – Marks the start of a compressed image represented in the interchange format or abbreviated format.

EOI: End of image marker – Marks the end of a compressed image represented in the interchange format or abbreviated format.

RST_m: Restart marker – A conditional marker which is placed between entropy-coded segments only if restart is enabled. There are 8 unique restart markers (m = 0 - 7) which repeat in sequence from 0 to 7, starting with zero for each scan, to provide a modulo 8 restart interval count.

The top level of Figure B.2 specifies that the non-hierarchical interchange format shall begin with an SOI marker, shall contain one frame, and shall end with an EOI marker.

The second level of Figure B.2 specifies that a frame shall begin with a frame header and shall contain one or more scans. A frame header may be preceded by one or more table-specification or miscellaneous marker segments as specified in B.2.4. If a DNL segment (see B.2.5) is present, it shall immediately follow the first scan.

For sequential DCT-based and lossless processes each scan shall contain from one to four image components. If two to four components are contained within a scan, they shall be interleaved within the scan. For progressive DCT-based processes each image component is only partially contained within any one scan. Only the first scan(s) for the components (which contain only DC coefficient data) may be interleaved.

The third level of Figure B.2 specifies that a scan shall begin with a scan header and shall contain one or more entropy-coded data segments. Each scan header may be preceded by one or more table-specification or miscellaneous marker segments. If restart is not enabled, there shall be only one entropy-coded segment (the one labeled “last”), and no restart markers shall be present. If restart is enabled, the number of entropy-coded segments is defined by the size of the image and the defined restart interval. In this case, a restart marker shall follow each entropy-coded segment except the last one.

The fourth level of Figure B.2 specifies that each entropy-coded segment is comprised of a sequence of entropy-coded MCUs. If restart is enabled and the restart interval is defined to be Ri, each entropy-coded segment except the last one shall contain Ri MCUs. The last one shall contain whatever number of MCUs completes the scan.

Figure B.2 specifies the locations where table-specification segments **may** be present. However, this Specification hereby specifies that the interchange format **shall** contain all table-specification data necessary for decoding the compressed image. Consequently, the required table-specification data **shall** be present at one or more of the allowed locations.

B.2.2 Frame header syntax

Figure B.3 specifies the frame header which shall be present at the start of a frame. This header specifies the source image characteristics (see A.1), the components in the frame, and the sampling factors for each component, and specifies the destinations from which the quantized tables to be used with each component are retrieved.

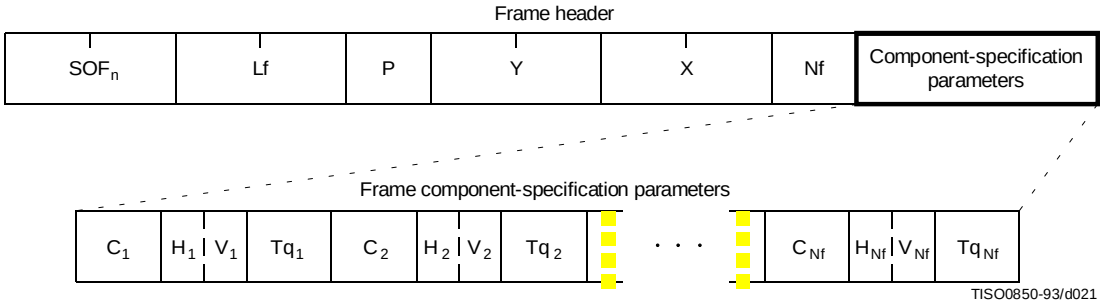


Figure B.3 – Frame header syntax

The markers and parameters shown in Figure B.3 are defined below. The size and allowed values of each parameter are given in Table B.2. In Table B.2 (and similar tables which follow), value choices are separated by commas (e.g. 8, 12) and inclusive bounds are separated by dashes (e.g. 0 - 3).

SOF_n: Start of frame marker – Marks the beginning of the frame parameters. The subscript n identifies whether the encoding process is baseline sequential, extended sequential, progressive, or lossless, as well as which entropy encoding procedure is used.

SOF_0: Baseline DCT

SOF_1: Extended sequential DCT, Huffman coding

SOF_2: Progressive DCT, Huffman coding

SOF_3: Lossless (sequential), Huffman coding

SOF_9: Extended sequential DCT, arithmetic coding

SOF_{10}: Progressive DCT, arithmetic coding

SOF_{11}: Lossless (sequential), arithmetic coding

Lf: Frame header length – Specifies the length of the frame header shown in Figure B.3 (see B.1.1.4).

P: Sample precision – Specifies the precision in bits for the samples of the components in the frame.

Y: Number of lines – Specifies the maximum number of lines in the source image. This shall be equal to the number of lines in the component with the maximum number of vertical samples (see A.1.1). Value 0 indicates that the number of lines shall be defined by the DNL marker and parameters at the end of the first scan (see B.2.5).

X: Number of samples per line – Specifies the maximum number of samples per line in the source image. This shall be equal to the number of samples per line in the component with the maximum number of horizontal samples (see A.1.1).

Nf: Number of image components in frame – Specifies the number of source image components in the frame. The value of Nf shall be equal to the number of sets of frame component specification parameters (C_i, H_i, V_i, and Tq_i) present in the frame header.

C_i: Component identifier – Assigns a unique label to the *i*th component in the sequence of frame component specification parameters. These values shall be used in the scan headers to identify the components in the scan. The value of C_i shall be different from the values of C_1 through C_{i-1}.

H_i: Horizontal sampling factor – Specifies the relationship between the component horizontal dimension and maximum image dimension X (see A.1.1); also specifies the number of horizontal data units of component C_i in each MCU, when more than one component is encoded in a scan.

V_i: Vertical sampling factor – Specifies the relationship between the component vertical dimension and maximum image dimension Y (see A.1.1); also specifies the number of vertical data units of component C_i in each MCU, when more than one component is encoded in a scan.

Tq_i: Quantization table destination selector – Specifies one of four possible quantization table destinations from which the quantization table to use for dequantization of DCT coefficients of component C_i is retrieved. If the decoding process uses the dequantization procedure, this table shall have been installed in this destination by the time the decoder is ready to decode the scan(s) containing component C_i. The destination shall not be re-specified, or its contents changed, until all scans containing C_i have been completed.

Table B.2 – Frame header parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lf	16	$8 + 3 \times Nf$			
P	8	8	8, 12	8, 12	2-16
Y	16	0-65 535			
X	16	1-65 535			
Nf	8	1-255	1-255	1-4	1-255
C_i	8	0-255			
H_i	4	1-4			
V_i	4	1-4			
Tq_i	8	0-3	0-3	0-3	0

B.2.3 Scan header syntax

Figure B.4 specifies the scan header which shall be present at the start of a scan. This header specifies which component(s) are contained in the scan, specifies the destinations from which the entropy tables to be used with each component are retrieved, and (for the progressive DCT) which part of the DCT quantized coefficient data is contained in the scan. For lossless processes the scan parameters specify the predictor and the point transform.

NOTE – If there is only one image component present in a scan, that component is, by definition, non-interleaved. If there is more than one image component present in a scan, the components present are, by definition, interleaved.

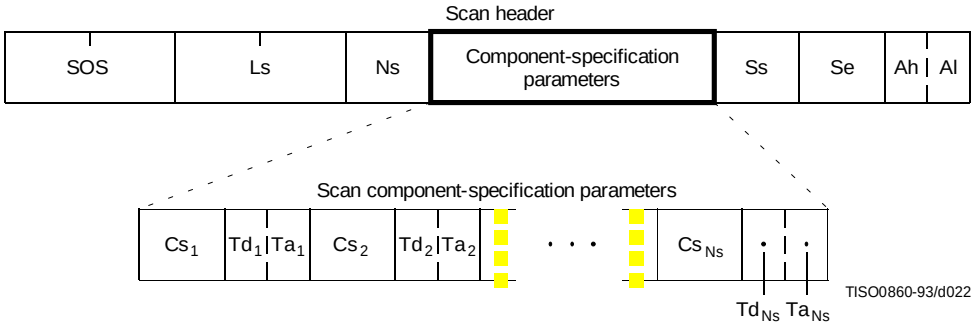


Figure B.4 – Scan header syntax

The marker and parameters shown in Figure B.4 are defined below. The size and allowed values of each parameter are given in Table B.3.

SOS: Start of scan marker – Marks the beginning of the scan parameters.

Ls: Scan header length – Specifies the length of the scan header shown in Figure B.4 (see B.1.1.4).

Ns: Number of image components in scan – Specifies the number of source image components in the scan. The value of Ns shall be equal to the number of sets of scan component specification parameters (Cs_j, Td_j, and Ta_j) present in the scan header.

Cs_j: Scan component selector – Selects which of the Nf image components specified in the frame parameters shall be the *j*th component in the scan. Each Cs_j shall match one of the C_i values specified in the frame header, and the ordering in the scan header shall follow the ordering in the frame header. If Ns > 1, the order of interleaved components in the MCU is Cs_1 first, Cs_2 second, etc. If Ns > 1, the following restriction shall be placed on the image components contained in the scan:

$$\sum_{j=1}^{N_s} H_j \times V_j \leq 10,$$

where H_j and V_j are the horizontal and vertical sampling factors for scan component j. These sampling factors are specified in the frame header for component i, where i is the frame component specification index for which frame component identifier C_i matches scan component selector Cs_j.

As an example, consider an image having 3 components with maximum dimensions of 512 lines and 512 samples per line, and with the following sampling factors:

Component 0	$H_0 = 4,$	$V_0 = 1$
Component 1	$H_1 = 1,$	$V_1 = 2$
Component 2	$H_2 = 2$	$V_2 = 2$

Then the summation of $H_j \times V_j$ is $(4 \times 1) + (1 \times 2) + (2 \times 2) = 10$.

The value of Cs_j shall be different from the values of Cs_1 to Cs_{j-1}.

Td_j: DC entropy coding table destination selector – Specifies one of four possible DC entropy coding table destinations from which the entropy table needed for decoding of the DC coefficients of component Cs_j is retrieved. The DC entropy table shall have been installed in this destination (see B.2.4.2 and B.2.4.3) by the time the decoder is ready to decode the current scan. This parameter specifies the entropy coding table destination for the lossless processes.

Ta_j: AC entropy coding table destination selector – Specifies one of four possible AC entropy coding table destinations from which the entropy table needed for decoding of the AC coefficients of component Cs_j is retrieved. The AC entropy table selected shall have been installed in this destination (see B.2.4.2 and B.2.4.3) by the time the decoder is ready to decode the current scan. This parameter is zero for the lossless processes.

Ss: Start of spectral or predictor selection – In the DCT modes of operation, this parameter specifies the first DCT coefficient in each block in zig-zag order which shall be coded in the scan. This parameter shall be set to zero for the sequential DCT processes. In the lossless mode of operations this parameter is used to select the predictor.

Se: End of spectral selection – Specifies the last DCT coefficient in each block in zig-zag order which shall be coded in the scan. This parameter shall be set to 63 for the sequential DCT processes. In the lossless mode of operations this parameter has no meaning. It shall be set to zero.

Ah: Successive approximation bit position high – This parameter specifies the point transform used in the preceding scan (i.e. successive approximation bit position low in the preceding scan) for the band of coefficients specified by Ss and Se. This parameter shall be set to zero for the first scan of each band of coefficients. In the lossless mode of operations this parameter has no meaning. It shall be set to zero.

Al: Successive approximation bit position low or point transform – In the DCT modes of operation this parameter specifies the point transform, i.e. bit position low, used before coding the band of coefficients specified by Ss and Se. This parameter shall be set to zero for the sequential DCT processes. In the lossless mode of operations, this parameter specifies the point transform, Pt.

The entropy coding table destination selectors, Td_j and Ta_j, specify either Huffman tables (in frames using Huffman coding) or arithmetic coding tables (in frames using arithmetic coding). In the latter case the entropy coding table destination selector specifies both an arithmetic coding conditioning table destination and an associated statistics area.

Table B.3 – Scan header parameter size and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Ls	16	$6 + 2 \times Ns$			
Ns	8	1-4			
Cs_j	8	0-255[a)]			
Td_j	4	0-1	0-3	0-3	0-3
Ta_j	4	0-1	0-3	0-3	0
Ss	8	0	0	0-63	1-7[b)]
Se	8	63	63	Ss-63[c)]	0
Ah	4	0	0	0-13	0
Al	4	0	0	0-13	0-15

a) Cs_j shall be a member of the set of C_i specified in the frame header.

b) 0 for lossless differential frames in the hierarchical mode (see B.3).

c) 0 if Ss equals zero.

B.2.4 Table-specification and miscellaneous marker segment syntax

Figure B.5 specifies that, at the places indicated in Figure B.2, any of the table-specification segments or miscellaneous marker segments specified in B.2.4.1 through B.2.4.6 may be present in any order and with no limit on the number of segments.

If any table specification for a particular destination occurs in the compressed image data, it shall replace any previous table specified for this destination, and shall be used whenever this destination is specified in the remaining scans in the frame or subsequent images represented in the abbreviated format for compressed image data. If a table specification for a given destination occurs more than once in the compressed image data, each specification shall replace the previous specification. The quantization table specification shall not be altered between progressive DCT scans of a given component.

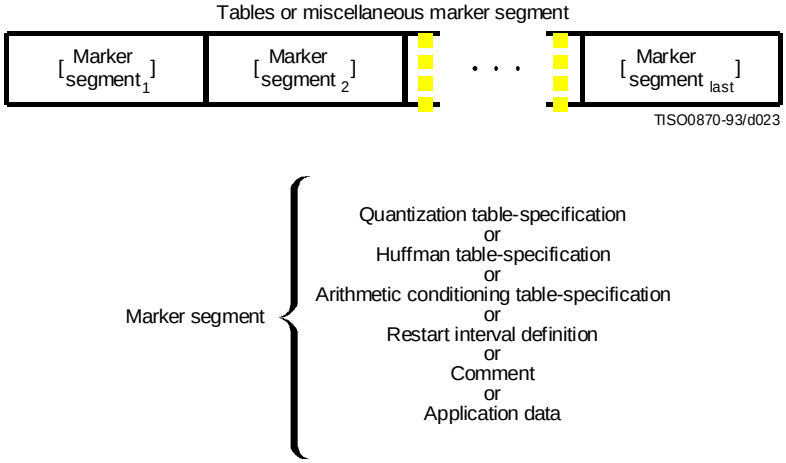


Figure B.5 – Tables/miscellaneous marker segment syntax

B.2.4.1 Quantization table-specification syntax

Figure B.6 specifies the marker segment which defines one or more quantization tables.

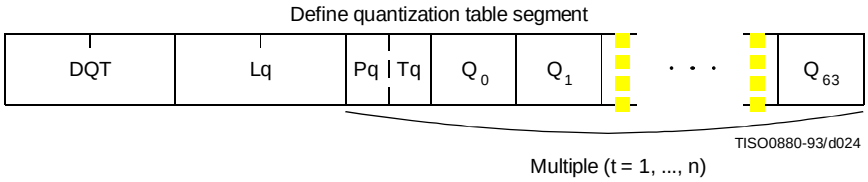


Figure B.6 – Quantization table syntax

The marker and parameters shown in Figure B.6 are defined below. The size and allowed values of each parameter are given in Table B.4.

DQT: Define quantization table marker – Marks the beginning of quantization table-specification parameters.

Lq: Quantization table definition length – Specifies the length of all quantization table parameters shown in Figure B.6 (see B.1.1.4).

Pq: Quantization table element precision – Specifies the precision of the Q_k values. Value 0 indicates 8-bit Q_k values; value 1 indicates 16-bit Q_k values. Pq shall be zero for 8 bit sample precision P (see B.2.2).

Tq: Quantization table destination identifier – Specifies one of four possible destinations at the decoder into which the quantization table shall be installed.

Q_k: Quantization table element – Specifies the *k*th element out of 64 elements, where *k* is the index in the zig-zag ordering of the DCT coefficients. The quantization elements shall be specified in zig-zag scan order.

Table B.4 – Quantization table-specification parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lq	16	$2 + \sum_{t=1}^{n} (65 + 64 \times Pq(t))$			Undefined
Pq	4	0	0, 1	0, 1	Undefined
Tq	4	0-3			Undefined
Q_k	8, 16	1-255, 1-65 535			Undefined

The value n in Table B.4 is the number of quantization tables specified in the DQT marker segment.

Once a quantization table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used, when referenced, in the remaining scans of the current image and in subsequent images represented in the abbreviated format for compressed image data. If a table has never been defined for a particular destination, then when this destination is specified in a frame header, the results are unpredictable.

An 8-bit DCT-based process shall not use a 16-bit precision quantization table.

B.2.4.2 Huffman table-specification syntax

Figure B.7 specifies the marker segment which defines one or more Huffman table specifications.

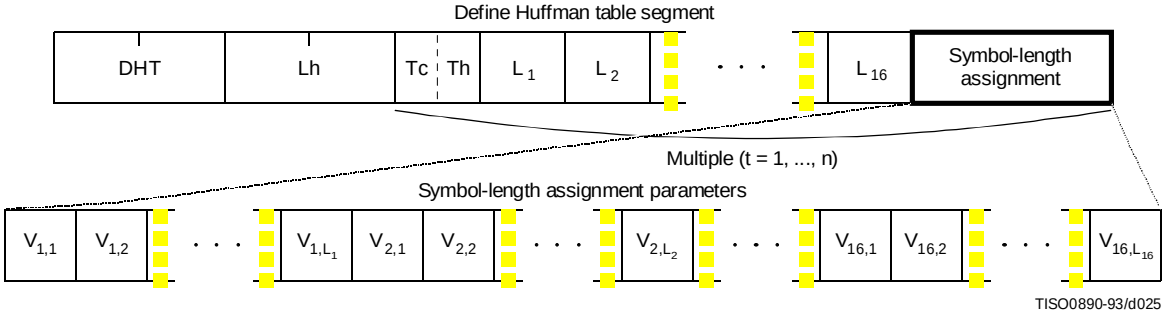


Figure B.7 – Huffman table syntax

The marker and parameters shown in Figure B.7 are defined below. The size and allowed values of each parameter are given in Table B.5.

DHT: Define Huffman table marker – Marks the beginning of Huffman table definition parameters.

Lh: Huffman table definition length – Specifies the length of all Huffman table parameters shown in Figure B.7 (see B.1.1.4).

Tc: Table class – 0 = DC table or lossless table, 1 = AC table.

Th: Huffman table destination identifier – Specifies one of four possible destinations at the decoder into which the Huffman table shall be installed.

L_i: Number of Huffman codes of length i – Specifies the number of Huffman codes for each of the 16 possible lengths allowed by this Specification. L_i's are the elements of the list BITS.

$V_{i,j}$: Value associated with each Huffman code – Specifies, for each i, the value associated with each Huffman code of length i. The meaning of each value is determined by the Huffman coding model. The $V_{i,j}$'s are the elements of the list HUFFVAL.

Table B.5 – Huffman table specification parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lh	16	$2 + \sum_{t=1}^{n} (17 + m_t)$			
Tc	4	0, 1			0
Th	4	0, 1	0-3		
L_i	8	0-255			
$V_{i,j}$	8	0-255			

The value n in Table B.5 is the number of Huffman tables specified in the DHT marker segment. The value m_t is the number of parameters which follow the 16 $L_i(t)$ parameters for Huffman table t, and is given by:

$$m_t = \sum_{i=1}^{16} L_i$$

In general, m_t is different for each table.

Once a Huffman table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used when referenced, in the remaining scans of the current image and in subsequent images represented in the abbreviated format for compressed image data. If a table has never been defined for a particular destination, then when this destination is specified in a scan header, the results are unpredictable.

B.2.4.3 Arithmetic conditioning table-specification syntax

Figure B.8 specifies the marker segment which defines one or more arithmetic coding conditioning table specifications. These replace the default arithmetic coding conditioning tables established by the SOI marker for arithmetic coding processes. (See F.1.4.4.1.4 and F.1.4.4.2.1.)

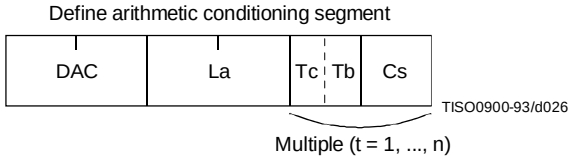


Figure B.8 – Arithmetic conditioning table-specification syntax

The marker and parameters shown in Figure B.8 are defined below. The size and allowed values of each parameter are given in Table B.6.

DAC: Define arithmetic coding conditioning marker – Marks the beginning of the definition of arithmetic coding conditioning parameters.

La: Arithmetic coding conditioning definition length – Specifies the length of all arithmetic coding conditioning parameters shown in Figure B.8 (see B.1.1.4).

Tc: Table class – 0 = DC table or lossless table, 1 = AC table.

Tb: Arithmetic coding conditioning table destination identifier – Specifies one of four possible destinations at the decoder into which the arithmetic coding conditioning table shall be installed.

Cs: Conditioning table value – Value in either the AC or the DC (and lossless) conditioning table. A single value of Cs shall follow each value of Tb. For AC conditioning tables Tc shall be one and Cs shall contain a value of Kx in the range $1 \le Kx \le 63$. For DC (and lossless) conditioning tables Tc shall be zero and Cs shall contain two 4-bit parameters, U and L. U and L shall be in the range $0 \le L \le U \le 15$ and the value of Cs shall be $L + 16 \times U$.

The value n in Table B.6 is the number of arithmetic coding conditioning tables specified in the DAC marker segment. The parameters L and U are the lower and upper conditioning bounds used in the arithmetic coding procedures defined for DC coefficient coding and lossless coding. The separate value range 1-63 listed for DCT coding is the Kx conditioning used in AC coefficient coding.

Table B.6 – Arithmetic coding conditioning table-specification parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
La	16	Undefined	$2 + 2 \times n$		
Tc	4	Undefined	0, 1		0
Tb	4	Undefined	0-3		
Cs	8	Undefined	0-255 (Tc = 0), 1-63 (Tc = 1)		0-255

B.2.4.4 Restart interval definition syntax

Figure B.9 specifies the marker segment which defines the restart interval.

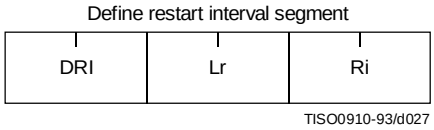


Figure B.9 – Restart interval definition syntax

The marker and parameters shown in Figure B.9 are defined below. The size and allowed values of each parameter are given in Table B.7.

DRI: Define restart interval marker – Marks the beginning of the parameters which define the restart interval.

Lr: Define restart interval segment length – Specifies the length of the parameters in the DRI segment shown in Figure B.9 (see B.1.1.4).

Ri: Restart interval – Specifies the number of MCU in the restart interval.

In Table B.7 the value n is the number of rows of MCU in the restart interval. The value MCUR is the number of MCU required to make up one line of samples of each component in the scan. The SOI marker disables the restart intervals. A DRI marker segment with Ri nonzero shall be present to enable restart interval processing for the following scans. A DRI marker segment with Ri equal to zero shall disable restart intervals for the following scans.

Table B.7 – Define restart interval segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lr	16	4			
Ri	16	0-65 535			n × MCUR

B.2.4.5 Comment syntax

Figure B.10 specifies the marker segment structure for a comment segment.

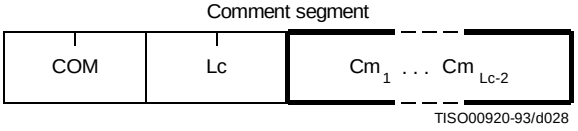


Figure B.10 – Comment segment syntax

The marker and parameters shown in Figure B.10 are defined below. The size and allowed values of each parameter are given in Table B.8.

COM: Comment marker – Marks the beginning of a comment.

Lc: Comment segment length – Specifies the length of the comment segment shown in Figure B.10 (see B.1.1.4).

Cm_i: Comment byte – The interpretation is left to the application.

Table B.8 – Comment segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lc	16	2-65 535			
Cm_i	8	0-255			

B.2.4.6 Application data syntax

Figure B.11 specifies the marker segment structure for an application data segment.

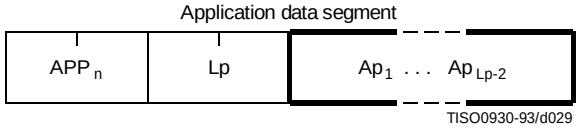


Figure B.11 – Application data syntax

The marker and parameters shown in Figure B.11 are defined below. The size and allowed values of each parameter are given in Table B.9.

APP_n: Application data marker – Marks the beginning of an application data segment.

Lp: Application data segment length – Specifies the length of the application data segment shown in Figure B.11 (see B.1.1.4).

Ap_i: Application data byte – The interpretation is left to the application.

The APP_n (Application) segments are reserved for application use. Since these segments may be defined differently for different applications, they should be removed when the data are exchanged between application environments.

Table B.9 – Application data segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lp	16	2-65 535			
Ap_i	8	0-255			

B.2.5 Define number of lines syntax

Figure B.12 specifies the marker segment for defining the number of lines. The DNL (Define Number of Lines) segment provides a mechanism for defining or redefining the number of lines in the frame (the Y parameter in the frame header) at the end of the first scan. The value specified shall be consistent with the number of MCU-rows encoded in the first scan. This segment, if used, shall only occur at the end of the first scan, and only after coding of an integer number of MCU-rows. This marker segment is mandatory if the number of lines (Y) specified in the frame header has the value zero.

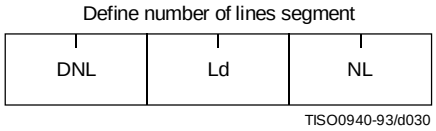


Figure B.12 – Define number of lines syntax

The marker and parameters shown in Figure B.12 are defined below. The size and allowed values of each parameter are given in Table B.10.

DNL: Define number of lines marker – Marks the beginning of the define number of lines segment.

Ld: Define number of lines segment length – Specifies the length of the define number of lines segment shown in Figure B.12 (see B.1.1.4).

NL: Number of lines – Specifies the number of lines in the frame (see definition of Y in B.2.2).

Table B.10 – Define number of lines segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Ld	16	4			
NL	16	1-65 535[a)]			

a) The value specified shall be consistent with the number of lines coded at the point where the DNL segment terminates the compressed data segment.

B.3 Hierarchical syntax

B.3.1 High level hierarchical mode syntax

Figure B.13 specifies the order of the high level constituent parts of the interchange format for hierarchical encoding processes.

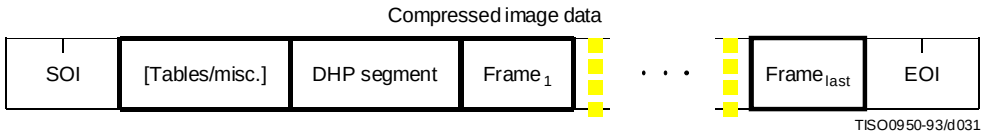


Figure B.13 – Syntax for the hierarchical mode of operation

Hierarchical mode syntax requires a DHP marker segment that appears before the non-differential frame or frames. The hierarchical mode compressed image data may include EXP marker segments and differential frames which shall follow the initial non-differential frame. The frame structure in hierarchical mode is identical to the frame structure in non-hierarchical mode.

The non-differential frames in the hierarchical sequence shall use one of the coding processes specified for SOF_n markers: SOF_0, SOF_1, SOF_2, SOF_3, SOF_9, SOF_{10} and SOF_{11}. The differential frames shall use one of the processes specified for SOF_5, SOF_6, SOF_7, SOF_{13}, SOF_{14} and SOF_{15}. The allowed combinations of SOF markers within one hierarchical sequence are specified in Annex J.

The sample precision (P) shall be constant for all frames and have the identical value as that coded in the DHP marker segment. The number of samples per line (X) for all frames shall not exceed the value coded in the DHP marker segment. If the number of lines (Y) is non-zero in the DHP marker segment, then the number of lines for all frames shall not exceed the value in the DHP marker segment.

B.3.2 DHP segment syntax

The DHP segment defines the image components, size, and sampling factors for the completed hierarchical sequence of frames. The DHP segment shall precede the first frame; a single DHP segment shall occur in the compressed image data.

The DHP segment structure is identical to the frame header syntax, except that the DHP marker is used instead of the SOF_n marker. The figures and description of B.2.2 then apply, except that the quantization table destination selector parameter shall be set to zero in the DHP segment.

B.3.3 EXP segment syntax

Figure B.14 specifies the marker segment structure for the EXP segment. The EXP segment shall be present if (and only if) expansion of the reference components is required either horizontally or vertically. The EXP segment parameters apply only to the next frame (which shall be a differential frame) in the image. If required, the EXP segment shall be one of the table-specification segments or miscellaneous marker segments preceding the frame header; the EXP segment shall not be one of the table-specification segments or miscellaneous marker segments preceding a scan header or a DHP marker segment.

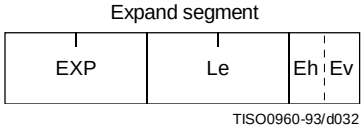


Figure B.14 – Syntax of the expand segment

The marker and parameters shown in Figure B.14 are defined below. The size and allowed values of each parameter are given in Table B.11.

EXP: Expand reference components marker – Marks the beginning of the expand reference components segment.

Le: Expand reference components segment length – Specifies the length of the expand reference components segment (see B.1.1.4).

Eh: Expand horizontally – If one, the reference components shall be expanded horizontally by a factor of two. If horizontal expansion is not required, the value shall be zero.

Ev: Expand vertically – If one, the reference components shall be expanded vertically by a factor of two. If vertical expansion is not required, the value shall be zero.

Both Eh and Ev shall be one if expansion is required both horizontally and vertically.

Table B.11 – Expand segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Le	16	3			
Eh	4	0, 1			
Ev	4	0, 1			

B.4 Abbreviated format for compressed image data

Figure B.2 shows the high-level constituent parts of the interchange format. This format includes all table specifications required for decoding. If an application environment provides methods for table specification other than by means of the compressed image data, some or all of the table specifications may be omitted. Compressed image data which is missing any table specification data required for decoding has the abbreviated format.

B.5 Abbreviated format for table-specification data

Figure B.2 shows the high-level constituent parts of the interchange format. If no frames are present in the compressed image data, the only purpose of the compressed image data is to convey table specifications or miscellaneous marker segments defined in B.2.4.1, B.2.4.2, B.2.4.5, and B.2.4.6. In this case the compressed image data has the abbreviated format for table specification data (see Figure B.15).

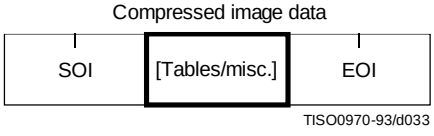


Figure B.15 – Abbreviated format for table-specification data syntax

B.6 Summary

The order of the constituent parts of interchange format and all marker segment structures is summarized in Figures B.16 and B.17. Note that in Figure B.16 double-lined boxes enclose marker segments. In Figures B.16 and B.17 thick-lined boxes enclose only markers.

The EXP segment can be mixed with the other tables/miscellaneous marker segments preceding the frame header but not with the tables/miscellaneous marker segments preceding the DHP segment or the scan header.

APPENDIX F

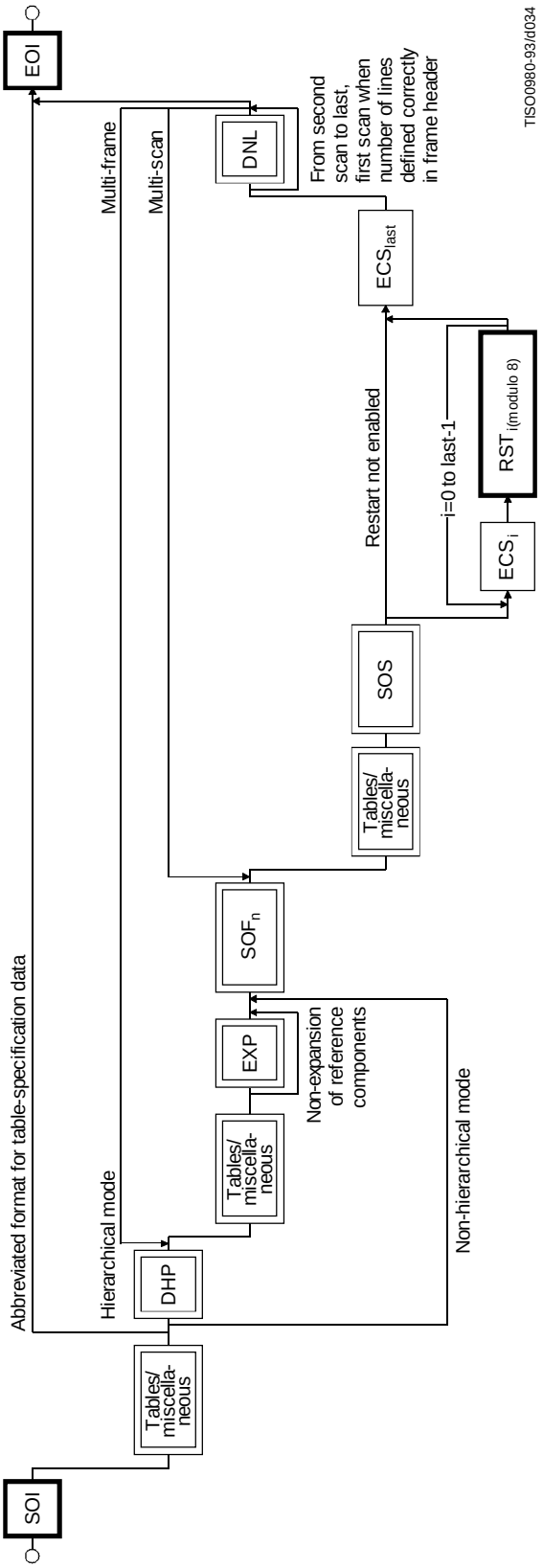


Figure B.16 – Flow of compressed data syntax

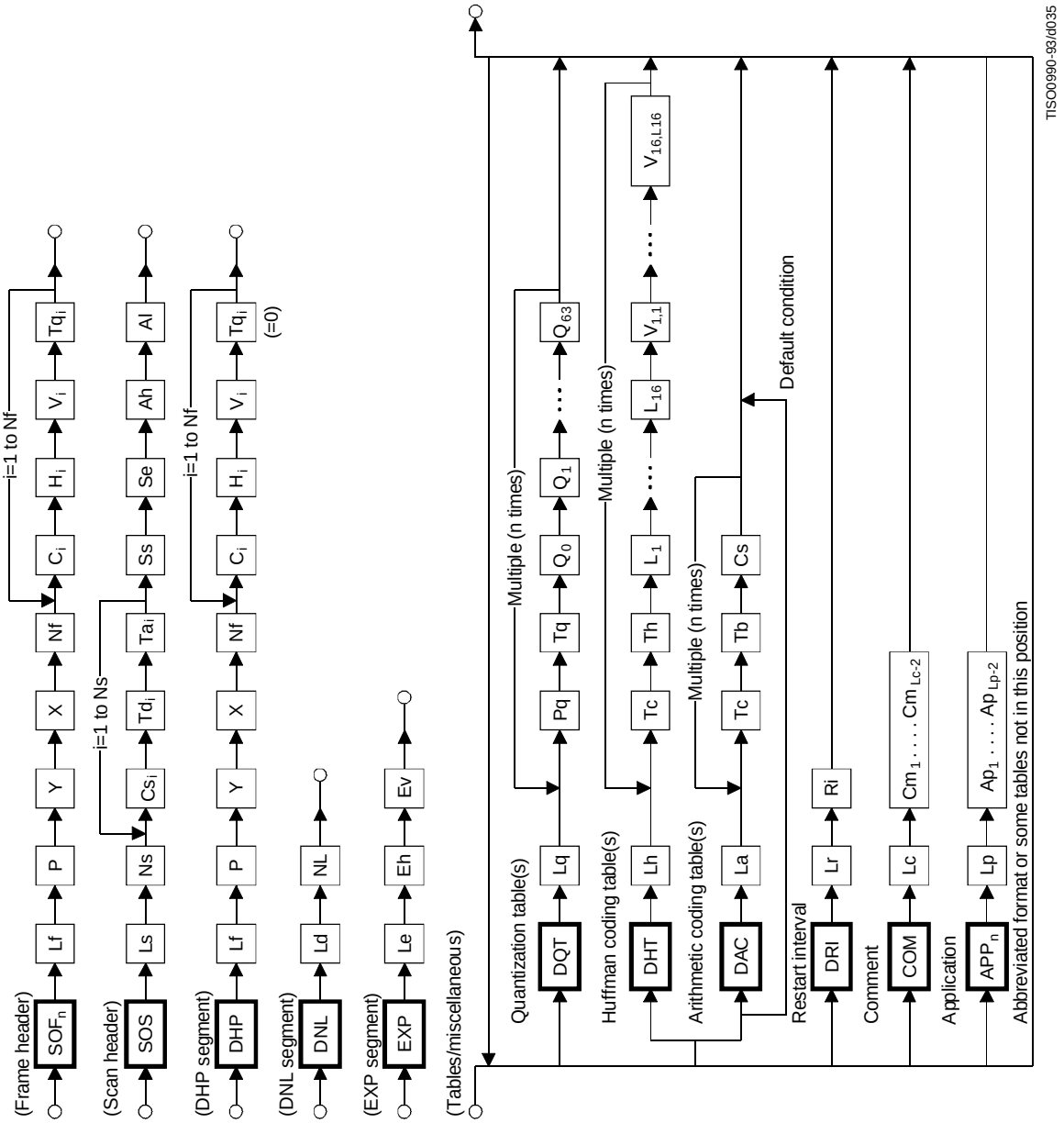


Figure B.17 – Flow of marker segment

APPENDIX F

Annex C

Huffman table specification

(This annex forms an integral part of this Recommendation | International Standard)

A Huffman coding procedure may be used for entropy coding in any of the coding processes. Coding models for Huffman encoding are defined in Annexes F, G, and H. In this Annex, the Huffman table specification is defined.

Huffman tables are specified in terms of a 16-byte list (BITS) giving the number of codes for each code length from 1 to 16. This is followed by a list of the 8-bit symbol values (HUFFVAL), each of which is assigned a Huffman code. The symbol values are placed in the list in order of increasing code length. Code lengths greater than 16 bits are not allowed. In addition, the codes shall be generated such that the all-1-bits code word of any length is reserved as a prefix for longer code words.

NOTE – The order of the symbol values within HUFFVAL is determined only by code length. Within a given code length the ordering of the symbol values is arbitrary.

This annex specifies the procedure by which the Huffman tables (of Huffman code words and their corresponding 8-bit symbol values) are derived from the two lists (BITS and HUFFVAL) in the interchange format. However, the way in which these lists are generated is not specified. The lists should be generated in a manner which is consistent with the rules for Huffman coding, and it shall observe the constraints discussed in the previous paragraph. Annex K contains an example of a procedure for generating lists of Huffman code lengths and values which are in accord with these rules.

NOTE – There is **no requirement** in this Specification that any encoder or decoder shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

C.1 Marker segments for Huffman table specification

The DHT marker identifies the start of Huffman table definitions within the compressed image data. B.2.4.2 specifies the syntax for Huffman table specification.

C.2 Conversion of Huffman table specifications to tables of codes and code lengths

Conversion of Huffman table specifications to tables of codes and code lengths uses three procedures. The first procedure (Figure C.1) generates a table of Huffman code sizes. The second procedure (Figure C.2) generates the Huffman codes from the table built in Figure C.1. The third procedure (Figure C.3) generates the Huffman codes in symbol value order.

Given a list BITS (1 to 16) containing the number of codes of each size, and a list HUFFVAL containing the symbol values to be associated with those codes as described above, two tables are generated. The HUFFSIZE table contains a list of code lengths; the HUFFCODE table contains the Huffman codes corresponding to those lengths.

Note that the variable LASTK is set to the index of the last entry in the table.

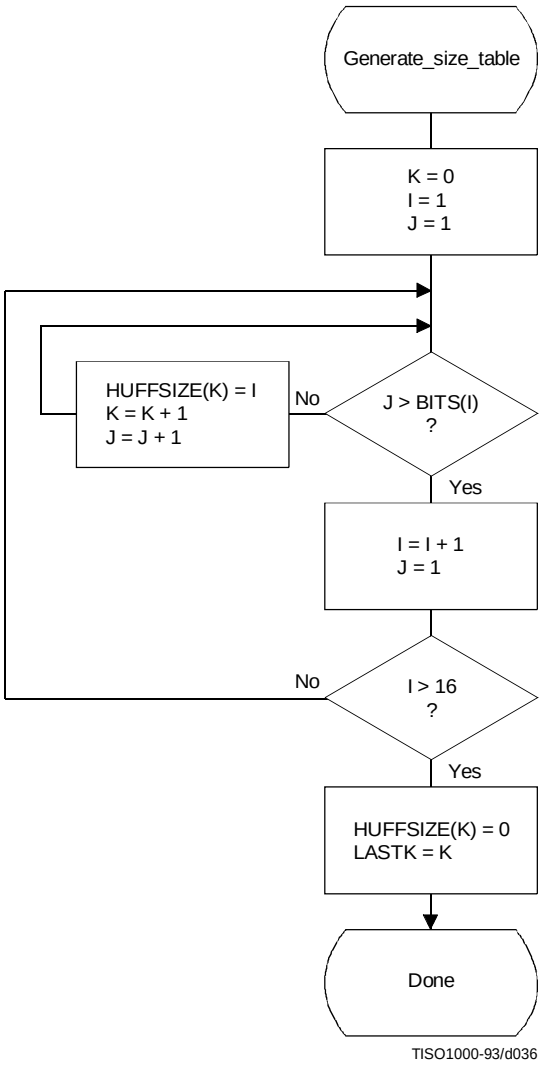


Figure C.1 – Generation of table of Huffman code sizes

A Huffman code table, HUFFCODE, containing a code for each size in HUFFSIZE is generated by the procedure in Figure C.2. The notation “SLL CODE 1” in Figure C.2 indicates a shift-left-logical of CODE by one bit position.

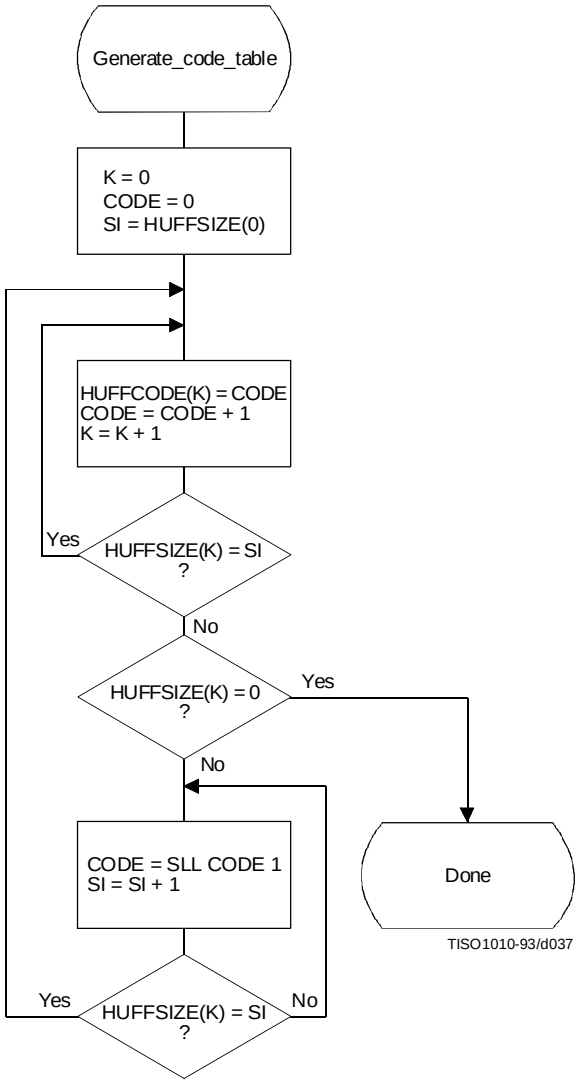


Figure C.2 – Generation of table of Huffman codes

Two tables, HUFFCODE and HUFFSIZE, have now been generated. The entries in the tables are ordered according to increasing Huffman code numeric value and length.

The encoding procedure code tables, EHUFCO and EHUFSI, are created by reordering the codes specified by HUFFCODE and HUFFSIZE according to the symbol values assigned to each code in HUFFVAL.

Figure C.3 illustrates this ordering procedure.

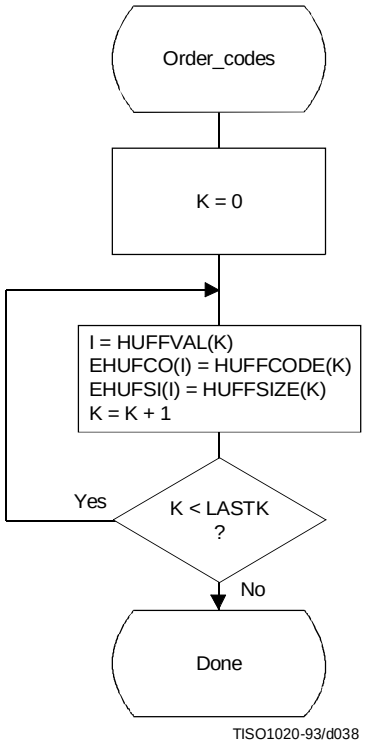


Figure C.3 – Ordering procedure for encoding procedure code tables

C.3 Bit ordering within bytes

The root of a Huffman code is placed toward the MSB (most-significant-bit) of the byte, and successive bits are placed in the direction MSB to LSB (least-significant-bit) of the byte. Remaining bits, if any, go into the next byte following the same rules.

Integers associated with Huffman codes are appended with the MSB adjacent to the LSB of the preceding Huffman code.

APPENDIX F

Annex D

Arithmetic coding

(This annex forms an integral part of this Recommendation | International Standard)

An adaptive binary arithmetic coding procedure may be used for entropy coding in any of the coding processes except the baseline sequential process. Coding models for adaptive binary arithmetic coding are defined in Annexes F, G, and H. In this annex the arithmetic encoding and decoding procedures used in those models are defined.

In K.4 a simple test example is given which should be helpful in determining if a given implementation is correct.

NOTE – There is **no requirement** in this Specification that any encoder or decoder shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

D.1 Arithmetic encoding procedures

Four arithmetic encoding procedures are required in a system with arithmetic coding (see Table D.1).

Table D.1 – Procedures for binary arithmetic encoding

Procedure	Purpose
Code_0(S)	Code a “0” binary decision with context-index S
Code_1(S)	Code a “1” binary decision with context-index S
Initenc	Initialize the encoder
Flush	Terminate entropy-coded segment

The “Code_0(S)”and “Code_1(S)” procedures code the 0-decision and 1-decision respectively; S is a context-index which identifies a particular conditional probability estimate used in coding the binary decision. The “Initenc” procedure initializes the arithmetic coding entropy encoder. The “Flush” procedure terminates the entropy-coded segment in preparation for the marker which follows.

D.1.1 Binary arithmetic encoding principles

The arithmetic coder encodes a series of binary symbols, zeros and ones, each symbol representing one possible result of a binary decision.

Each “binary decision” provides a choice between two alternatives. The binary decision might be between positive and negative signs, a magnitude being zero or nonzero, or a particular bit in a sequence of binary digits being zero or one.

The output bit stream (entropy-coded data segment) represents a binary fraction which increases in precision as bytes are appended by the encoding process.

D.1.1.1 Recursive interval subdivision

Recursive probability interval subdivision is the basis for the binary arithmetic encoding procedures. With each binary decision the current probability interval is subdivided into two sub-intervals, and the bit stream is modified (if necessary) so that it points to the base (the lower bound) of the probability sub-interval assigned to the symbol which occurred.

In the partitioning of the current probability interval into two sub-intervals, the sub-interval for the less probable symbol (LPS) and the sub-interval for the more probable symbol (MPS) are ordered such that usually the MPS sub-interval is closer to zero. Therefore, when the LPS is coded, the MPS sub-interval size is added to the bit stream. This coding convention requires that symbols be recognized as either MPS or LPS rather than 0 or 1. Consequently, the size of the LPS sub-interval and the sense of the MPS for each decision must be known in order to encode that decision.

The subdivision of the current probability interval would ideally require a multiplication of the interval by the probability estimate for the LPS. Because this subdivision is done approximately, it is possible for the LPS sub-interval to be larger than the MPS sub-interval. When that happens a "conditional exchange" interchanges the assignment of the sub-intervals such that the MPS is given the larger sub-interval.

Since the encoding procedure involves addition of binary fractions rather than concatenation of integer code words, the more probable binary decisions can sometimes be coded at a cost of much less than one bit per decision.

D.1.1.2 Conditioning of probability estimates

An adaptive binary arithmetic coder requires a statistical model – a model for selecting conditional probability estimates to be used in the coding of each binary decision. When a given binary decision probability estimate is dependent on a particular feature or features (the context) already coded, it is "conditioned" on that feature. The conditioning of probability estimates on previously coded decisions must be identical in encoder and decoder, and therefore can use only information known to both.

Each conditional probability estimate required by the statistical model is kept in a separate storage location or "bin" identified by a unique context-index S. The arithmetic coder is adaptive, which means that the probability estimates at each context-index are developed and maintained by the arithmetic coding system on the basis of prior coding decisions for that context-index.

D.1.2 Encoding conventions and approximations

The encoding procedures use fixed precision integer arithmetic and an integer representation of fractional values in which X'8000' can be regarded as the decimal value 0.75. The probability interval, A, is kept in the integer range X'8000' ≤ A < X'10000' by doubling it whenever its integer value falls below X'8000'. This is equivalent to keeping A in the decimal range $0.75 \le A < 1.5$. This doubling procedure is called renormalization.

The code register, C, contains the trailing bits of the bit stream. C is also doubled each time A is doubled. Periodically – to keep C from overflowing – a byte of data is removed from the high order bits of the C-register and placed in the entropy-coded segment.

Carry-over into the entropy-coded segment is limited by delaying X'FF' output bytes until the carry-over is resolved. Zero bytes are stuffed after each X'FF' byte in the entropy-coded segment in order to avoid the accidental generation of markers in the entropy-coded segment.

Keeping A in the range $0.75 \le A < 1.5$ allows a simple arithmetic approximation to be used in the probability interval subdivision. Normally, if the current estimate of the LPS probability for context-index S is Qe(S), precise calculation of the sub-intervals would require:

$Qe(S) \times A$	Probability sub-interval for the LPS;
$A - (Qe(S) \times A)$	Probability sub-interval for the MPS.

Because the decimal value of A is of order unity, these can be approximated by

Qe(S)	Probability sub-interval for the LPS;
A – Qe(S)	Probability sub-interval for the MPS.

Whenever the LPS is coded, the value of A – Qe(S) is added to the code register and the probability interval is reduced to Qe(S). Whenever the MPS is coded, the code register is left unchanged and the interval is reduced to A – Qe(S). The precision range required for A is then restored, if necessary, by renormalization of both A and C.

With the procedure described above, the approximations in the probability interval subdivision process can sometimes make the LPS sub-interval larger than the MPS sub-interval. If, for example, the value of Qe(S) is 0.5 and A is at the minimum allowed value of 0.75, the approximate scaling gives one-third of the probability interval to the MPS and two-thirds to the LPS. To avoid this size inversion, conditional exchange is used. The probability interval is subdivided using the simple approximation, but the MPS and LPS sub-interval assignments are exchanged whenever the LPS sub-interval is larger than the MPS sub-interval. This MPS/LPS conditional exchange can only occur when a renormalization will be needed.

Each binary decision uses a context. A context is the set of prior coding decisions which determine the context-index, S, identifying the probability estimate used in coding the decision.

Whenever a renormalization occurs, a probability estimation procedure is invoked which determines a new probability estimate for the context currently being coded. No explicit symbol counts are needed for the estimation. The relative probabilities of renormalization after coding of LPS and MPS provide, by means of a table-based probability estimation state machine, a direct estimate of the probabilities.

D.1.3 Encoder code register conventions

The flow charts in this annex assume the register structures for the encoder as shown in Table D.2.

Table D.2 – Encoder register connections

	MSB			LSB
C-register	0000cbbb,	bbbbbsss,	xxxxxxxx,	xxxxxxxx
A-register	00000000,	00000000,	aaaaaaaa,	aaaaaaaa

The “a” bits are the fractional bits in the A-register (the current probability interval value) and the “x” bits are the fractional bits in the code register. The “s” bits are optional spacer bits which provide useful constraints on carry-over, and the “b” bits indicate the bit positions from which the completed bytes of data are removed from the C-register. The “c” bit is a carry bit. Except at the time of initialization, bit 15 of the A-register is always set and bit 16 is always clear (the LSB is bit 0).

These register conventions illustrate one possible implementation. However, any register conventions which allow resolution of carry-over in the encoder and which produce the same entropy-coded segment may be used. The handling of carry-over and the byte stuffing following X’FF’ will be described in a later part of this annex.

D.1.4 Code_1(S) and Code_0(S) procedures

When a given binary decision is coded, one of two possibilities occurs – either a 1-decision or a 0-decision is coded. Code_1(S) and Code_0(S) are shown in Figures D.1 and D.2. The Code_1(S) and Code_0(S) procedures use probability estimates with a context-index S. The context-index S is determined by the statistical model and is, in general, a function of the previous coding decisions; each value of S identifies a particular conditional probability estimate which is used in encoding the binary decision.

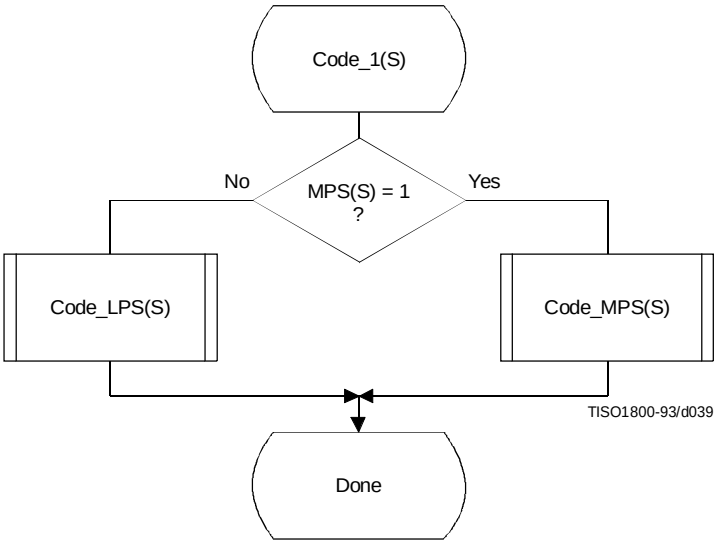


Figure D.1 – Code_1(S) procedure

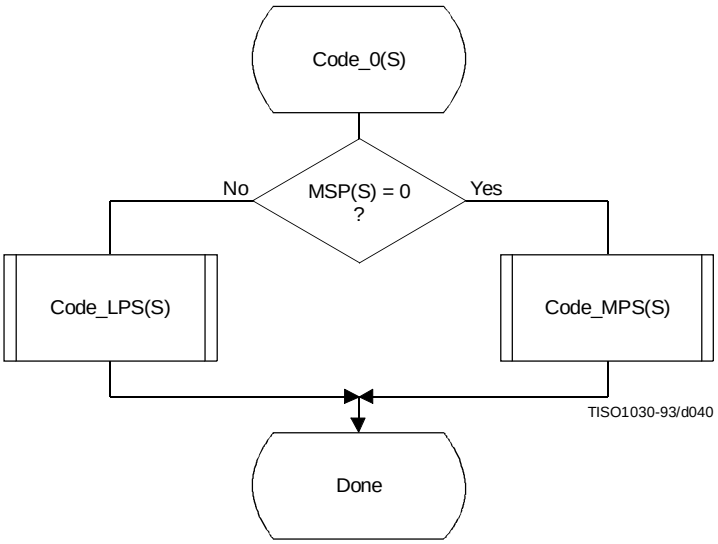


Figure D.2 – Code_0(S) procedure

The context-index S selects a storage location which contains Index(S), an index to the tables which make up the probability estimation state machine. When coding a binary decision, the symbol being coded is either the more probable symbol or the less probable symbol. Therefore, additional information is stored at each context-index identifying the sense of the more probable symbol, MPS(S).

For simplicity, the flow charts in this subclause assume that the context storage for each context-index S has an additional storage field for Qe(S) containing the value of Qe(Index(S)). If only the value of Index(S) and MPS(S) are stored, all references to Qe(S) should be replaced by Qe(Index(S)).

The Code_LPS(S) procedure normally consists of the addition of the MPS sub-interval A – Qe(S) to the bit stream and a scaling of the interval to the sub-interval, Qe(S). It is always followed by the procedures for obtaining a new LPS probability estimate (Estimate_Qe(S)_after_LPS) and renormalization (Renorm_e) (see Figure D.3).

However, in the event that the LPS sub-interval is larger than the MPS sub-interval, the conditional MPS/LPS exchange occurs and the MPS sub-interval is coded.

The Code_MPS(S) procedure normally reduces the size of the probability interval to the MPS sub-interval. However, if the LPS sub-interval is larger than the MPS sub-interval, the conditional exchange occurs and the LPS sub-interval is coded instead. Note that conditional exchange cannot occur unless the procedures for obtaining a new LPS probability estimate (Estimate_Qe(S)_after_MPS) and renormalization (Renorm_e) are required after the coding of the symbol (see Figure D.4).

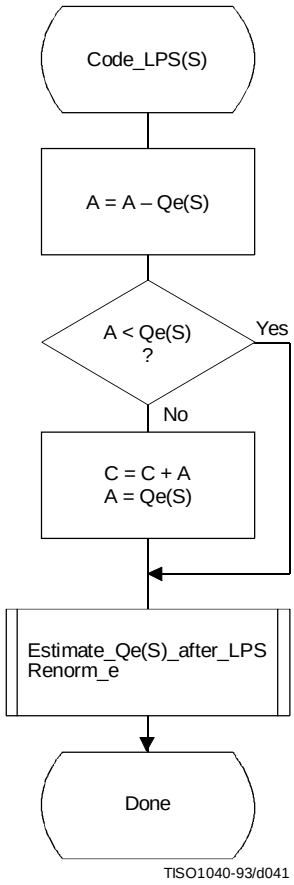


Figure D.3 – Code_LPS(S) procedure with conditional MPS/LPS exchange

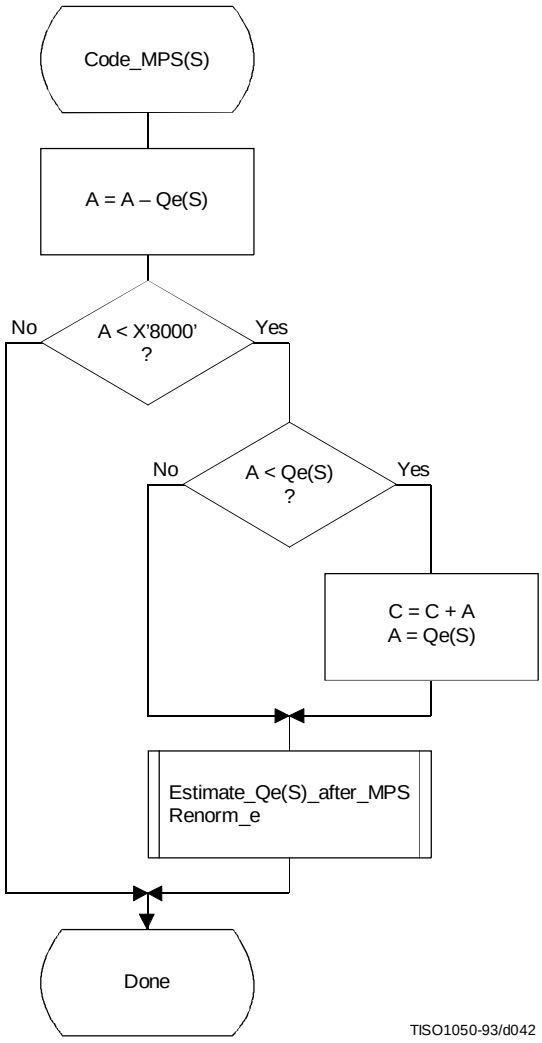


Figure D.4 – Code_MPS(S) procedure with conditional MPS/LPS exchange

D.1.5 Probability estimation in the encoder

D.1.5.1 Probability estimation state machine

The probability estimation state machine consists of a number of sequences of probability estimates. These sequences are interlinked in a manner which provides probability estimates based on approximate symbol counts derived from the arithmetic coder renormalization. Some of these sequences are used during the initial "learning" stages of probability estimation; the rest are used for "steady state" estimation.

Each entry in the probability estimation state machine is assigned an index, and each index has associated with it a Qe value and two Next_Index values. The Next_Index_MPS gives the index to the new probability estimate after an MPS renormalization; the Next_Index_LPS gives the index to the new probability estimate after an LPS renormalization. Note that both the index to the estimation state machine and the sense of the MPS are kept for each context-index S. The sense of the MPS is changed whenever the entry in the Switch_MPS is one.

The probability estimation state machine is given in Table D.3. Initialization of the arithmetic coder is always with an MPS sense of zero and a Qe index of zero in Table D.3.

The Qe values listed in Table D.3 are expressed as hexadecimal integers. To approximately convert the 15-bit integer representation of Qe to a decimal probability, divide the Qe values by $(4/3) \times (X'8000')$.

Table D.3 – Qe values and probability estimation state machine

Index	Qe _Value	Next_ Index _LPS	Next_ Index _MPS	Switch _MPS	Index	Qe _Value	Next_ Index _LPS	Next_ Index _MPS	Switch _MPS
0	X'5A1D'	1	1	1	57	X'01A4'	55	58	0
1	X'2586'	14	2	0	58	X'0160'	56	59	0
2	X'1114'	16	3	0	59	X'0125'	57	60	0
3	X'080B'	18	4	0	60	X'00F6'	58	61	0
4	X'03D8'	20	5	0	61	X'00CB'	59	62	0
5	X'01DA'	23	6	0	62	X'00AB'	61	63	0
6	X'00E5'	25	7	0	63	X'008F'	61	32	0
7	X'006F'	28	8	0	64	X'5B12'	65	65	1
8	X'0036'	30	9	0	65	X'4D04'	80	66	0
9	X'001A'	33	10	0	66	X'412C'	81	67	0
10	X'000D'	35	11	0	67	X'37D8'	82	68	0
11	X'0006'	9	12	0	68	X'2FE8'	83	69	0
12	X'0003'	10	13	0	69	X'293C'	84	70	0
13	X'0001'	12	13	0	70	X'2379'	86	71	0
14	X'5A7F'	15	15	1	71	X'1EDF'	87	72	0
15	X'3F25'	36	16	0	72	X'1AA9'	87	73	0
16	X'2CF2'	38	17	0	73	X'174E'	72	74	0
17	X'207C'	39	18	0	74	X'1424'	72	75	0
18	X'17B9'	40	19	0	75	X'119C'	74	76	0
19	X'1182'	42	20	0	76	X'0F6B'	74	77	0
20	X'0CEF'	43	21	0	77	X'0D51'	75	78	0
21	X'09A1'	45	22	0	78	X'0BB6'	77	79	0
22	X'072F'	46	23	0	79	X'0A40'	77	48	0
23	X'055C'	48	24	0	80	X'5832'	80	81	1
24	X'0406'	49	25	0	81	X'4D1C'	88	82	0
25	X'0303'	51	26	0	82	X'438E'	89	83	0
26	X'0240'	52	27	0	83	X'3BDD'	90	84	0
27	X'01B1'	54	28	0	84	X'34EE'	91	85	0
28	X'0144'	56	29	0	85	X'2EAE'	92	86	0
29	X'00F5'	57	30	0	86	X'299A'	93	87	0
30	X'00B7'	59	31	0	87	X'2516'	86	71	0
31	X'008A'	60	32	0	88	X'5570'	88	89	1
32	X'0068'	62	33	0	89	X'4CA9'	95	90	0
33	X'004E'	63	34	0	90	X'44D9'	96	91	0
34	X'003B'	32	35	0	91	X'3E22'	97	92	0
35	X'002C'	33	9	0	92	X'3824'	99	93	0
36	X'5AE1'	37	37	1	93	X'32B4'	99	94	0
37	X'484C'	64	38	0	94	X'2E17'	93	86	0
38	X'3A0D'	65	39	0	95	X'56A8'	95	96	1
39	X'2EF1'	67	40	0	96	X'4F46'	101	97	0
40	X'261F'	68	41	0	97	X'47E5'	102	98	0
41	X'1F33'	69	42	0	98	X'41CF'	103	99	0
42	X'19A8'	70	43	0	99	X'3C3D'	104	100	0
43	X'1518'	72	44	0	100	X'375E'	99	93	0
44	X'1177'	73	45	0	101	X'5231'	105	102	0
45	X'0E74'	74	46	0	102	X'4C0F'	106	103	0
46	X'0BFB'	75	47	0	103	X'4639'	107	104	0
47	X'09F8'	77	48	0	104	X'415E'	103	99	0
48	X'0861'	78	49	0	105	X'5627'	105	106	1
49	X'0706'	79	50	0	106	X'50E7'	108	107	0
50	X'05CD'	48	51	0	107	X'4B85'	109	103	0
51	X'04DE'	50	52	0	108	X'5597'	110	109	0
52	X'040F'	50	53	0	109	X'504F'	111	107	0
53	X'0363'	51	54	0	110	X'5A10'	110	111	1
54	X'02D4'	52	55	0	111	X'5522'	112	109	0
55	X'025C'	53	56	0	112	X'59EB'	112	111	1
56	X'01F8'	54	57	0					

D.1.5.2 Renormalization driven estimation

The change in state in Table D.3 occurs only when the arithmetic coder interval register is renormalized. This must always be done after coding an LPS, and whenever the probability interval register is less than X'8000' (0.75 in decimal notation) after coding an MPS.

When the LPS renormalization is required, Next_Index_LPS gives the new index for the LPS probability estimate. When the MPS renormalization is required, Next_Index_MPS gives the new index for the LPS probability estimate. If Switch_MPS is 1 for the old index, the MPS symbol sense must be inverted after an LPS.

D.1.5.3 Estimation following renormalization after MPS

The procedure for estimating the probability on the MPS renormalization path is given in Figure D.5. Index(S) is part of the information stored for context-index S. The new value of Index(S) is obtained from Table D.3 from the column labeled Next_Index_MPS, as that is the next index after an MPS renormalization. This next index is stored as the new value of Index(S) in the context storage at context-index S, and the value of Qe at this new Index(S) becomes the new Qe(S). MPS(S) does not change.

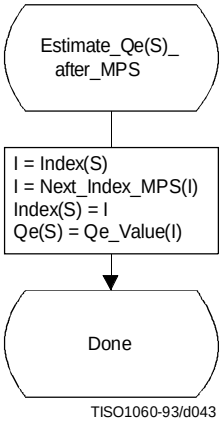


Figure D.5 – Probability estimation on MPS renormalization path

D.1.5.4 Estimation following renormalization after LPS

The procedure for estimating the probability on the LPS renormalization path is shown in Figure D.6. The procedure is similar to that of Figure D.5 except that when Switch_MPS(I) is 1, the sense of MPS(S) must be inverted.

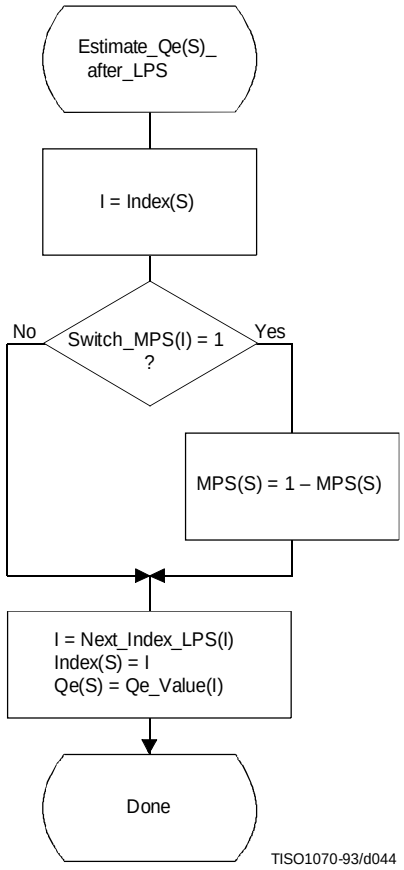


Figure D.6 – Probability estimation on LPS renormalization path

D.1.6 Renormalization in the encoder

The Renorm_e procedure for the encoder renormalization is shown in Figure D.7. Both the probability interval register A and the code register C are shifted, one bit at a time. The number of shifts is counted in the counter CT; when CT is zero, a byte of compressed data is removed from C by the procedure Byte_out and CT is reset to 8. Renormalization continues until A is no longer less than X'8000'.

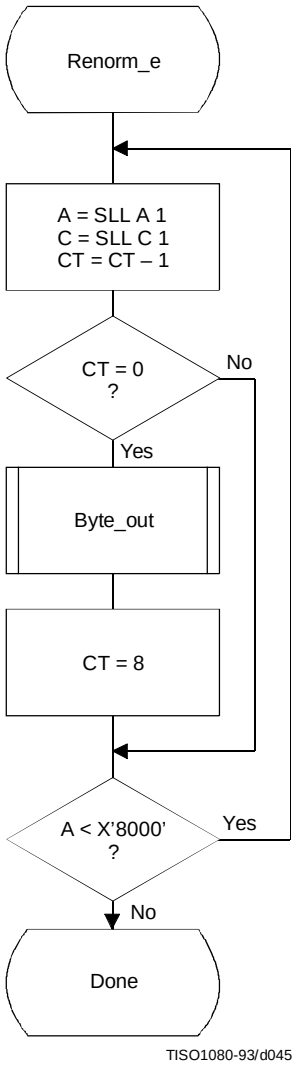


Figure D.7 – Encoder renormalization procedure

The Byte_out procedure used in Renorm_e is shown in Figure D.8. This procedure uses byte-stuffing procedures which prevent accidental generation of markers by the arithmetic encoding procedures. It also includes an example of a procedure for resolving carry-over. For simplicity of exposition, the buffer holding the entropy-coded segment is assumed to be large enough to contain the entire segment.

In Figure D.8 BP is the entropy-coded segment pointer and B is the compressed data byte pointed to by BP. T in Byte_out is a temporary variable which is used to hold the output byte and carry bit. ST is the stack counter which is used to count X'FF' output bytes until any carry-over through the X'FF' sequence has been resolved. The value of ST rarely exceeds 3. However, since the upper limit for the value of ST is bounded only by the total entropy-coded segment size, a precision of 32 bits is recommended for ST.

Since large values of ST represent a latent output of compressed data, the following procedure may be needed in high speed synchronous encoding systems for handling the burst of output data which occurs when the carry is resolved.

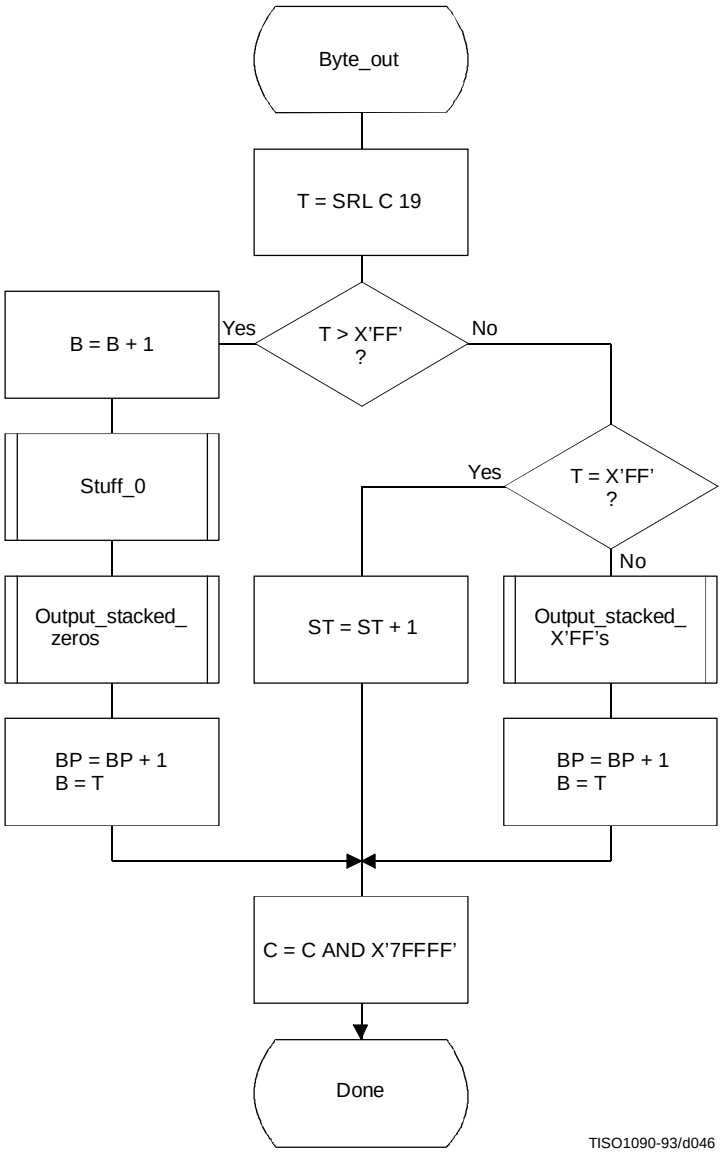


Figure D.8 – Byte_out procedure for encoder

When the stack count reaches an upper bound determined by output channel capacity, the stack is emptied and the stacked X'FF' bytes (and stuffed zero bytes) are added to the compressed data before the carry-over is resolved. If a carry-over then occurs, the carry is added to the final stuffed zero, thereby converting the final X'FF00' sequence to the X'FF01' temporary private marker. The entropy-coded segment must then be post-processed to resolve the carry-over and remove the temporary marker code. For any reasonable bound on ST this post processing is very unlikely.

Referring to Figure D.8, the shift of the code register by 19 bits aligns the output bits with the low order bits of T. The first test then determines if a carry-over has occurred. If so, the carry must be added to the previous output byte before advancing the segment pointer BP. The Stuff_0 procedure stuffs a zero byte whenever the addition of the carry to the data already in the entropy-coded segments creates a X'FF' byte. Any stacked output bytes – converted to zeros by the carry-over – are then placed in the entropy-coded segment. Note that when the output byte is later transferred from T to the entropy-coded segment (to byte B), the carry bit is ignored if it is set.

If a carry has not occurred, the output byte is tested to see if it is X'FF'. If so, the stack count ST is incremented, as the output must be delayed until the carry-over is resolved. If not, the carry-over has been resolved, and any stacked X'FF' bytes must then be placed in the entropy-coded segment. Note that a zero byte is stuffed following each X'FF'.

The procedures used by Byte_out are defined in Figures D.9 through D.11.

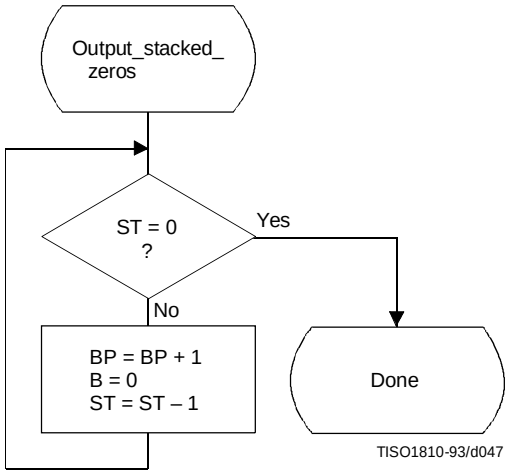


Figure D.9 – Output_stacked_zeros procedure for encoder

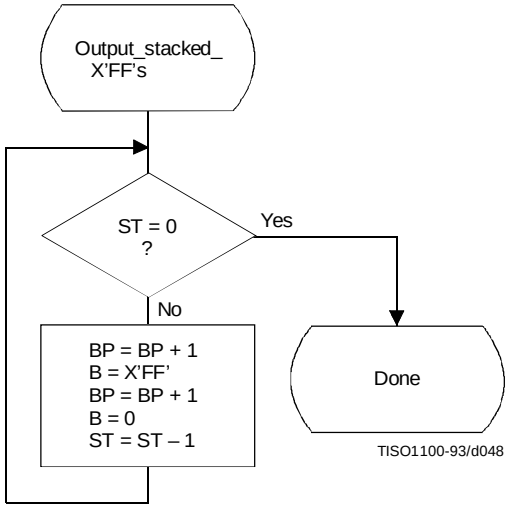


Figure D.10 – Output_stacked_X'FF's procedure for encoder

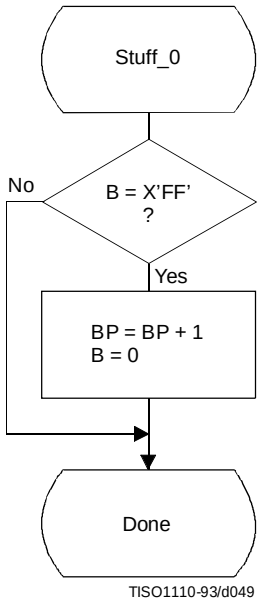


Figure D.11 – Stuff_0 procedure for encoder

D.1.7 Initialization of the encoder

The Initenc procedure is used to start the arithmetic coder. The basic steps are shown in Figure D.12.

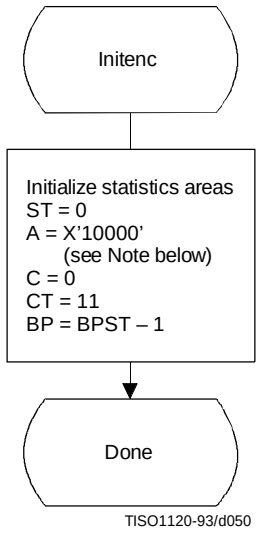


Figure D.12 – Initialization of the encoder

The probability estimation tables are defined by Table D.3. The statistics areas are initialized to an MPS sense of 0 and a Qe index of zero as defined by Table D.3. The stack count (ST) is cleared, the code register (C) is cleared, and the interval register is set to X'10000'. The counter (CT) is set to 11, reflecting the fact that when A is initialized to X'10000' three spacer bits plus eight output bits in C must be filled before the first byte is removed. Note that BP is initialized to point to the byte before the start of the entropy-coded segment (which is at BPST). Note also that the statistics areas are initialized for all values of context-index S to MPS(S) = 0 and Index(S) = 0.

NOTE – Although the probability interval is initialized to X'10000' in both Initenc and Initdec, the precision of the probability interval register can still be limited to 16 bits. When the precision of the interval register is 16 bits, it is initialized to zero.

D.1.8 Termination of encoding

The Flush procedure is used to terminate the arithmetic encoding procedures and prepare the entropy-coded segment for the addition of the X'FF' prefix of the marker which follows the arithmetically coded data. Figure D.13 shows this flush procedure. The first step in the procedure is to set as many low order bits of the code register to zero as possible without pointing outside of the final interval. Then, the output byte is aligned by shifting it left by CT bits; Byte_out then removes it from C. C is then shifted left by 8 bits to align the second output byte and Byte_out is used a second time. The remaining low order bits in C are guaranteed to be zero, and these trailing zero bits shall not be written to the entropy-coded segment.

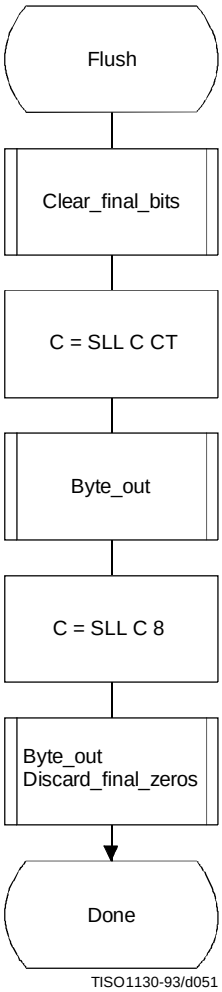


Figure D.13 – Flush procedure

Any trailing zero bytes already written to the entropy-coded segment and not preceded by a X'FF' may, optionally, be discarded. This is done in the Discard_final_zeros procedure. Stuffed zero bytes shall not be discarded.

Entropy coded segments are always followed by a marker. For this reason, the final zero bits needed to complete decoding shall not be included in the entropy coded segment. Instead, when the decoder encounters a marker, zero bits shall be supplied to the decoding procedure until decoding is complete. This convention guarantees that when a DNL marker is used, the decoder will intercept it in time to correctly terminate the decoding procedure.

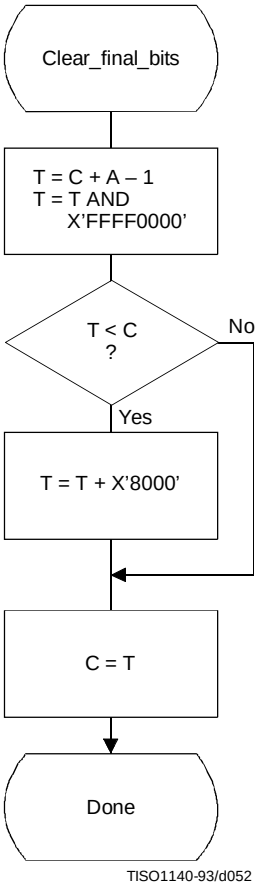


Figure D.14 – Clear_final_bits procedure in Flush

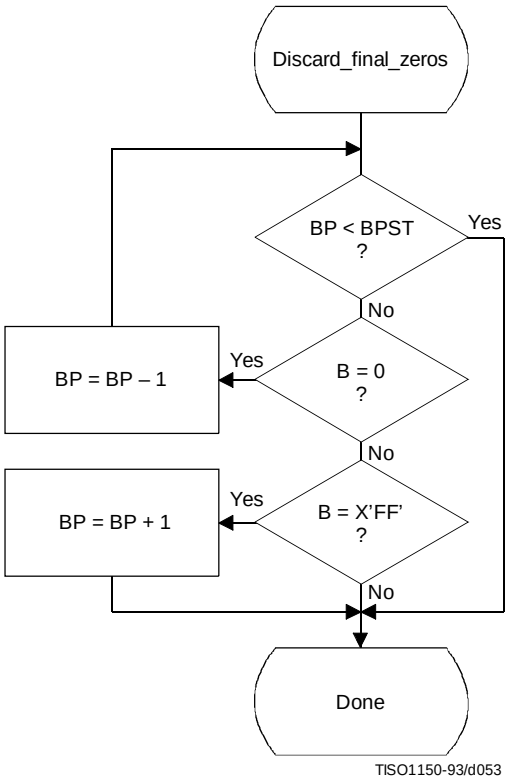


Figure D.15 – Discard_final_zeros procedure in Flush

D.2 Arithmetic decoding procedures

Two arithmetic decoding procedures are used for arithmetic decoding (see Table D.4).

The "Decode(S)" procedure decodes the binary decision for a given context-index S and returns a value of either 0 or 1. It is the inverse of the "Code_0(S)" and "Code_1(S)" procedures described in D.1. "Initdec" initializes the arithmetic coding entropy decoder.

Table D.4 – Procedures for binary arithmetic decoding

Procedure	Purpose
Decode(S)	Decode a binary decision with context-index S
Initdec	Initialize the decoder

D.2.1 Binary arithmetic decoding principles

The probability interval subdivision and sub-interval ordering defined for the arithmetic encoding procedures also apply to the arithmetic decoding procedures.

Since the bit stream always points within the current probability interval, the decoding process is a matter of determining, for each decision, which sub-interval is pointed to by the bit stream. This is done recursively, using the same probability interval sub-division process as in the encoder. Each time a decision is decoded, the decoder subtracts from the bit stream any interval the encoder added to the bit stream. Therefore, the code register in the decoder is a pointer into the current probability interval relative to the base of the interval.

If the size of the sub-interval allocated to the LPS is larger than the sub-interval allocated to the MPS, the encoder invokes the conditional exchange procedure. When the interval sizes are inverted in the decoder, the sense of the symbol decoded must be inverted.

D.2.2 Decoding conventions and approximations

The approximations and integer arithmetic defined for the probability interval subdivision in the encoder must also be used in the decoder. However, where the encoder would have added to the code register, the decoder subtracts from the code register.

D.2.3 Decoder code register conventions

The flow charts given in this section assume the register structures for the decoder as shown in Table D.5:

Table D.5 – Decoder register conventions

	MSB	LSB
Cx register	xxxxxxxx,	xxxxxxxx
C-low	bbbbbbbb,	00000000
A-register	aaaaaaaa,	aaaaaaaa

Cx and C-low can be regarded as one 32-bit C-register, in that renormalization of C shifts a bit of new data from bit 15 of C-low to bit 0 of Cx. However, the decoding comparisons use Cx alone. New data are inserted into the “b” bits of C-low one byte at a time.

NOTE – The comparisons shown in the various procedures use arithmetic comparisons, and therefore assume precisions greater than 16 bits for the variables. Unsigned (logical) comparisons should be used in 16-bit precision implementations.

D.2.4 The decode procedure

The decoder decodes one binary decision at a time. After decoding the decision, the decoder subtracts any amount from the code register that the encoder added. The amount left in the code register is the offset from the base of the current probability interval to the sub-interval allocated to the binary decisions not yet decoded. In the first test in the decode procedure shown in Figure D.16 the code register is compared to the size of the MPS sub-interval. Unless a conditional exchange is needed, this test determines whether the MPS or LPS for context-index S is decoded. Note that the LPS for context-index S is given by 1 – MPS(S).

When a renormalization is needed, the MPS/LPS conditional exchange may also be needed. For the LPS path, the conditional exchange procedure is shown in Figure D.17. Note that the probability estimation in the decoder is identical to the probability estimation in the encoder (Figures D.5 and D.6).

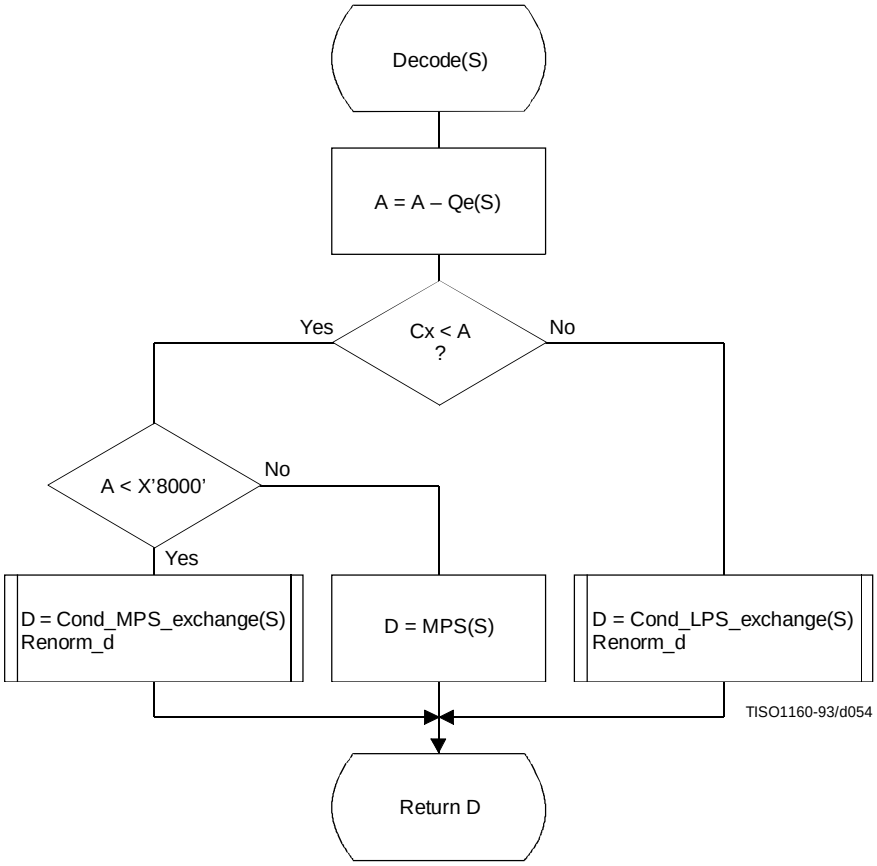


Figure D.16 – Decode(S) procedure

For the MPS path of the decoder the conditional exchange procedure is given in Figure D.18.

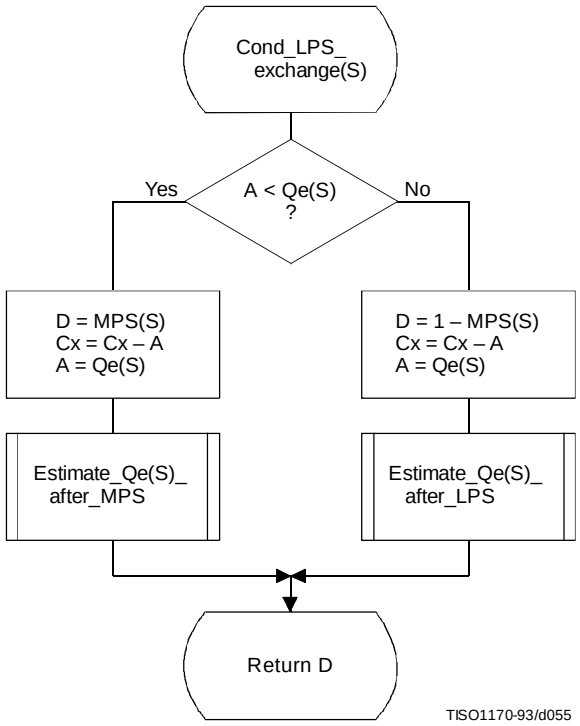


Figure D.17 – Decoder LPS path conditional exchange procedure

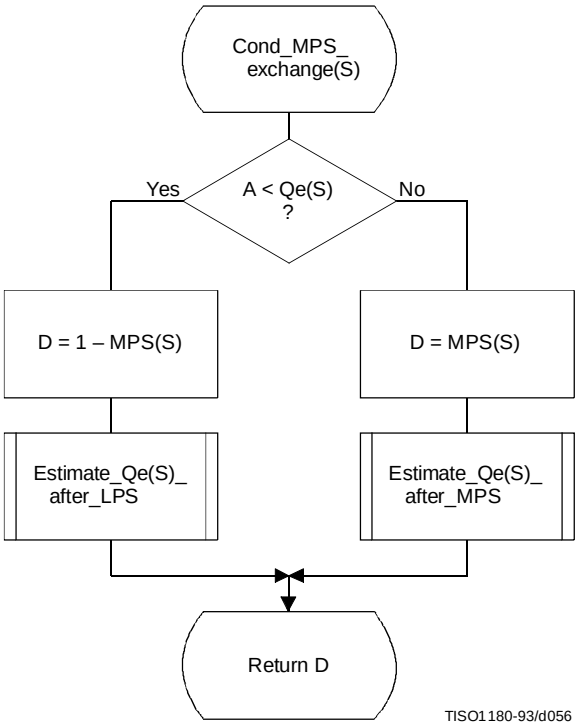


Figure D.18 – Decoder MPS path conditional exchange procedure

D.2.5 Probability estimation in the decoder

The procedures defined for obtaining a new LPS probability estimate in the encoder are also used in the decoder.

D.2.6 Renormalization in the decoder

The Renorm_d procedure for the decoder renormalization is shown in Figure D.19. CT is a counter which keeps track of the number of compressed bits in the C-low section of the C-register. When CT is zero, a new byte is inserted into C-low by the procedure Byte_in and CT is reset to 8.

Both the probability interval register A and the code register C are shifted, one bit at a time, until A is no longer less than X'8000'.

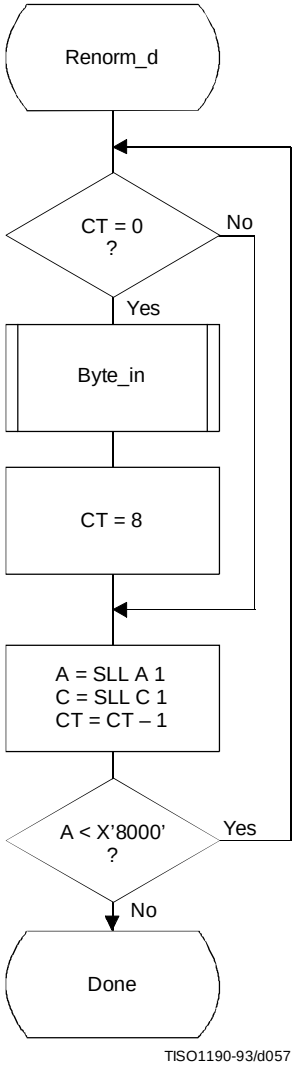


Figure D.19 – Decoder renormalization procedure

The Byte_in procedure used in Renorm_d is shown in Figure D.20. This procedure fetches one byte of data, compensating for the stuffed zero byte which follows any X'FF' byte. It also detects the marker which must follow the entropy-coded segment. The C-register in this procedure is the concatenation of the Cx and C-low registers. For simplicity of exposition, the buffer holding the entropy-coded segment is assumed to be large enough to contain the entire segment.

B is the byte pointed to by the entropy-coded segment pointer BP. BP is first incremented. If the new value of B is not a X'FF', it is inserted into the high order 8 bits of C-low.

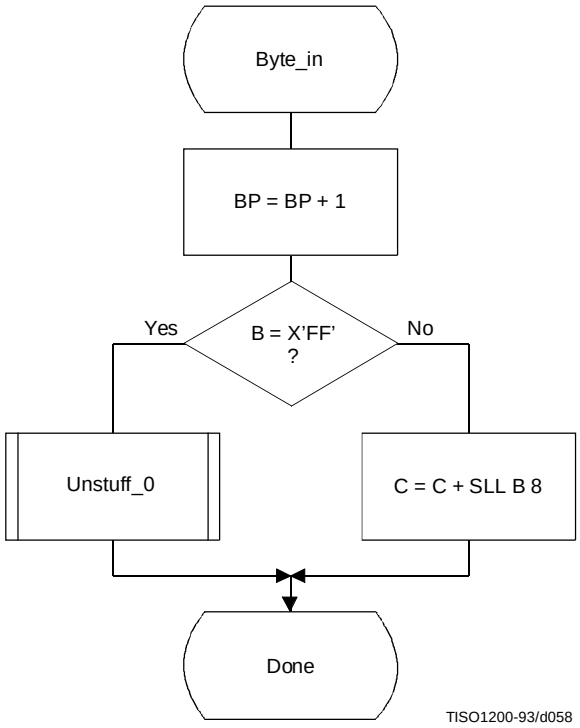


Figure D.20 – Byte_in procedure for decoder

The Unstuff_0 procedure is shown in Figure D.21. If the new value of B is X'FF', BP is incremented to point to the next byte and this next B is tested to see if it is zero. If so, B contains a stuffed byte which must be skipped. The zero B is ignored, and the X'FF' B value which preceded it is inserted in the C-register.

If the value of B after a X'FF' byte is not zero, then a marker has been detected. The marker is interpreted as required and the entropy-coded segment pointer is adjusted ("Adjust BP" in Figure D.21) so that 0-bytes will be fed to the decoder until decoding is complete. One way of accomplishing this is to point BP to the byte preceding the marker which follows the entropy-coded segment.

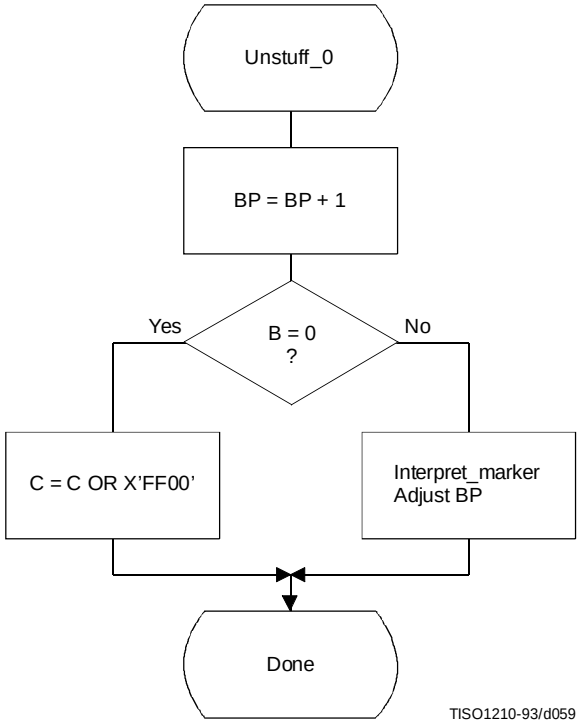


Figure D.21 – Unstuff_0 procedure for decoder

D.2.7 Initialization of the decoder

The Initdec procedure is used to start the arithmetic decoder. The basic steps are shown in Figure D.22.

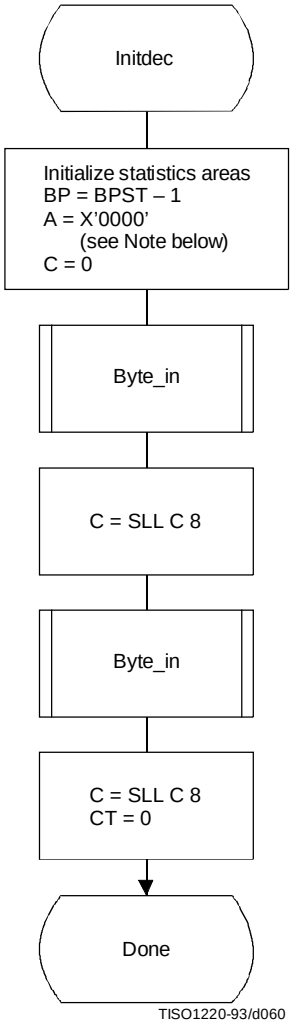


Figure D.22 – Initialization of the decoder

The estimation tables are defined by Table D.3. The statistics areas are initialized to an MPS sense of 0 and a Qe index of zero as defined by Table D.3. BP, the pointer to the entropy-coded segment, is then initialized to point to the byte before the start of the entropy-coded segment at BPST, and the interval register is set to the same starting value as in the encoder. The first byte of compressed data is fetched and shifted into Cx. The second byte is then fetched and shifted into Cx. The count is set to zero, so that a new byte of data will be fetched by Renorm_d.

NOTE – Although the probability interval is initialized to X'10000' in both Initenc and Initdec, the precision of the probability interval register can still be limited to 16 bits. When the precision of the interval register is 16 bits, it is initialized to zero.

D.3 Bit ordering within bytes

The arithmetically encoded entropy-coded segment is an integer of variable length. Therefore, the ordering of bytes and the bit ordering within bytes is the same as for parameters (see B.1.1.1).

Annex E

Encoder and decoder control procedures

(This annex forms an integral part of this Recommendation | International Standard)

This annex describes the encoder and decoder control procedures for the sequential, progressive, and lossless modes of operation.

The encoding and decoding control procedures for the hierarchical processes are specified in Annex J.

NOTES

1 There is **no requirement** in this Specification that any encoder or decoder shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

2 Implementation-specific setup steps are not indicated in this annex and may be necessary.

E.1 Encoder control procedures

E.1.1 Control procedure for encoding an image

The encoder control procedure for encoding an image is shown in Figure E.1.

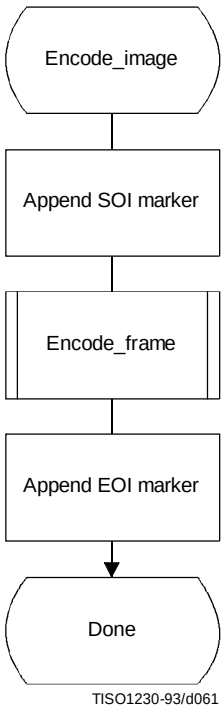


Figure E.1 – Control procedure for encoding an image

E.1.2 Control procedure for encoding a frame

In all cases where markers are appended to the compressed data, optional X'FF' fill bytes may precede the marker.

The control procedure for encoding a frame is oriented around the scans in the frame. The frame header is first appended, and then the scans are coded. Table specifications and other marker segments may precede the SOF_n marker, as indicated by [tables/miscellaneous] in Figure E.2.

Figure E.2 shows the encoding process frame control procedure.

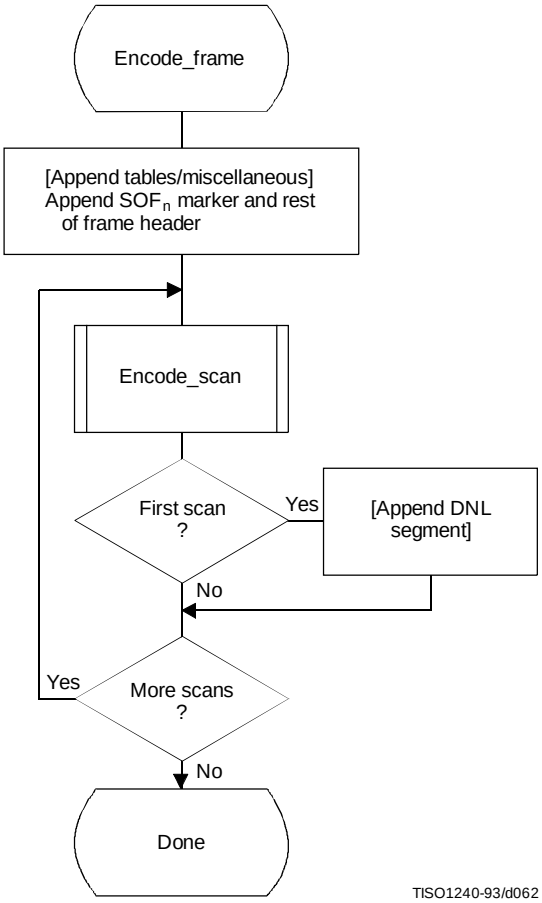


Figure E.2 – Control procedure for encoding a frame

E.1.3 Control procedure for encoding a scan

A scan consists of a single pass through the data of each component in the scan. Table specifications and other marker segments may precede the SOS marker. If more than one component is coded in the scan, the data are interleaved. If restart is enabled, the data are segmented into restart intervals. If restart is enabled, a RST_m marker is placed in the coded data between restart intervals. If restart is disabled, the control procedure is the same, except that the entire scan contains a single restart interval. The compressed image data generated by a scan is always followed by a marker, either the EOI marker or the marker of the next marker segment.

Figure E.3 shows the encoding process scan control procedure. The loop is terminated when the encoding process has coded the number of restart intervals which make up the scan. "m" is the restart interval modulo counter needed for the RST_m marker. The modulo arithmetic for this counter is shown after the "Append RST_m marker" procedure.

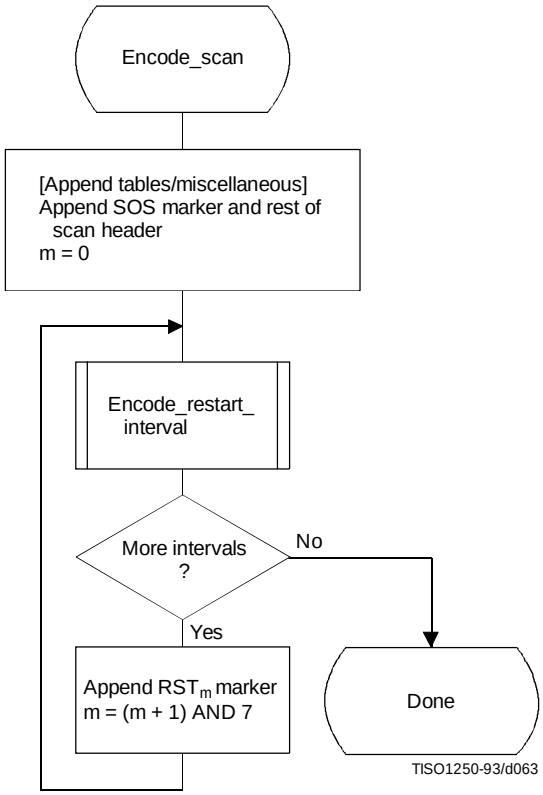


Figure E.3 – Control procedure for encoding a scan

E.1.4 Control procedure for encoding a restart interval

Figure E.4 shows the encoding process control procedure for a restart interval. The loop is terminated either when the encoding process has coded the number of minimum coded units (MCU) in the restart interval or when it has completed the image scan.

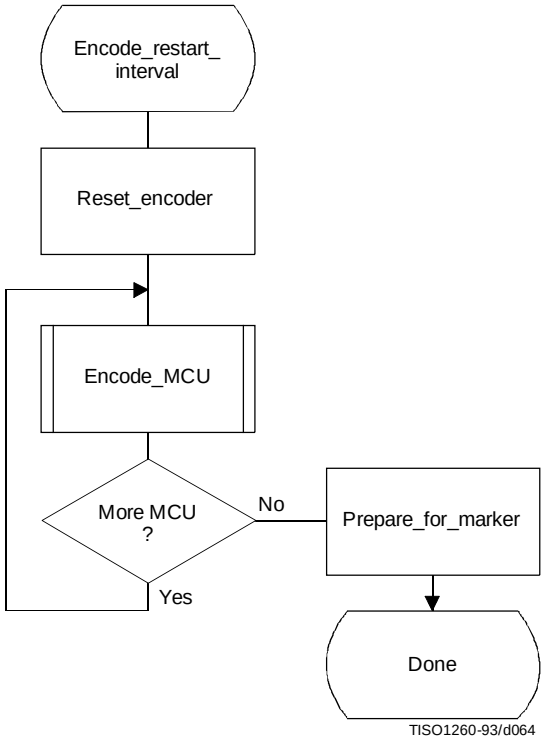


Figure E.4 – Control procedure for encoding a restart interval

The "Reset_encoder" procedure consists at least of the following:

a) if arithmetic coding is used, initialize the arithmetic encoder using the "Initenc" procedure described in D.1.7;

b) for DCT-based processes, set the DC prediction (PRED) to zero for all components in the scan (see F.1.1.5.1);

c) for lossless processes, reset the prediction to a default value for all components in the scan (see H.1.1);

d) do all other implementation-dependent setups that may be necessary.

The procedure "Prepare_for_marker" terminates the entropy-coded segment by:

a) padding a Huffman entropy-coded segment with 1-bits to complete the final byte (and if needed stuffing a zero byte) (see F.1.2.3); or

b) invoking the procedure "Flush" (see D.1.8) to terminate an arithmetic entropy-coded segment.

NOTE – The number of minimum coded units (MCU) in the final restart interval must be adjusted to match the number of MCU in the scan. The number of MCU is calculated from the frame and scan parameters. (See Annex B.)

E.1.5 Control procedure for encoding a minimum coded unit (MCU)

The minimum coded unit is defined in A.2. Within a given MCU the data units are coded in the order in which they occur in the MCU. The control procedure for encoding a MCU is shown in Figure E.5.

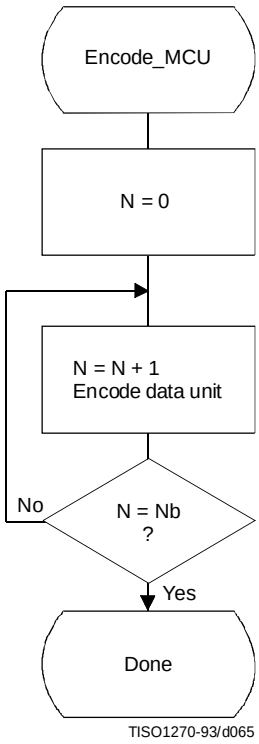


Figure E.5 – Control procedure for encoding a minimum coded unit (MCU)

In Figure E.5, Nb refers to the number of data units in the MCU. The order in which data units occur in the MCU is defined in A.2. The data unit is an 8×8 block for DCT-based processes, and a single sample for lossless processes.

The procedures for encoding a data unit are specified in Annexes F, G, and H.

E.2 Decoder control procedures

E.2.1 Control procedure for decoding compressed image data

Figure E.6 shows the decoding process control for compressed image data.

Decoding control centers around identification of various markers. The first marker must be the SOI (Start Of Image) marker. The "Decoder_setup" procedure resets the restart interval (Ri = 0) and, if the decoder has arithmetic decoding capabilities, sets the conditioning tables for the arithmetic coding to their default values. (See F.1.4.4.1.4 and F.1.4.4.2.1.) The next marker is normally a SOF_n (Start Of Frame) marker; if this is not found, one of the marker segments listed in Table E.1 has been received.

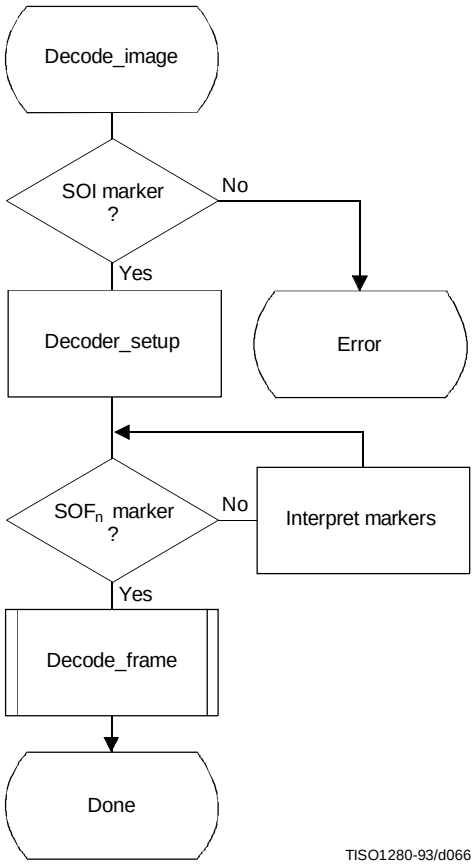


Figure E.6 – Control procedure for decoding compressed image data

Table E.1 – Markers recognized by "Interpret markers"

Marker	Purpose
DHT	Define Huffman Tables
DAC	Define Arithmetic Conditioning
DQT	Define Quantization Tables
DRI	Define Restart Interval
APP_n	Application defined marker
COM	Comment

Note that optional X'FF' fill bytes which may precede any marker shall be discarded before determining which marker is present.

The additional logic to interpret these various markers is contained in the box labeled “Interpret markers”. DHT markers shall be interpreted by processes using Huffman coding. DAC markers shall be interpreted by processes using arithmetic coding. DQT markers shall be interpreted by DCT-based decoders. DRI markers shall be interpreted by all decoders. APPn and COM markers shall be interpreted only to the extent that they do not interfere with the decoding.

By definition, the procedures in “Interpret markers” leave the system at the next marker. Note that if the expected SOI marker is missing at the start of the compressed image data, an error condition has occurred. The techniques for detecting and managing error conditions can be as elaborate or as simple as desired.

E.2.2 Control procedure for decoding a frame

Figure E.7 shows the control procedure for the decoding of a frame.

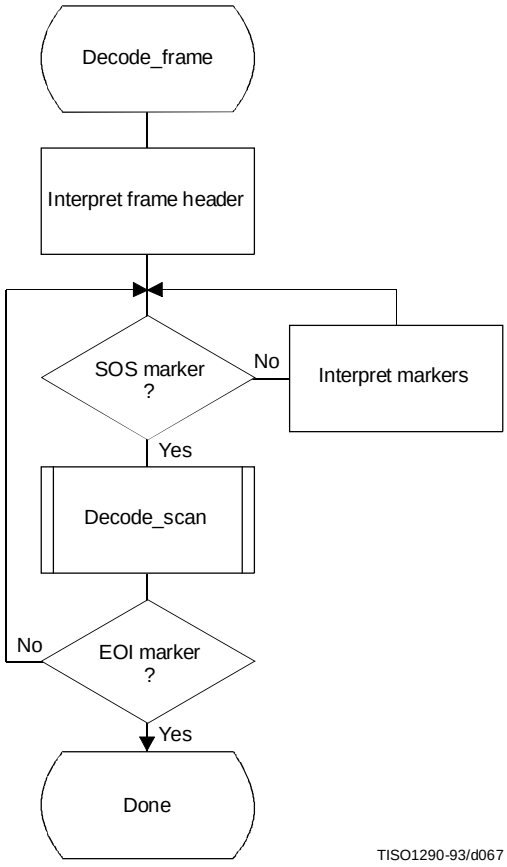


Figure E.7 – Control procedure for decoding a frame

The loop is terminated if the EOI marker is found at the end of the scan.

The markers recognized by “Interpret markers” are listed in Table E.1. Subclause E.2.1 describes the extent to which the various markers shall be interpreted.

E.2.3 Control procedure for decoding a scan

Figure E.8 shows the decoding of a scan.

The loop is terminated when the expected number of restart intervals has been decoded.

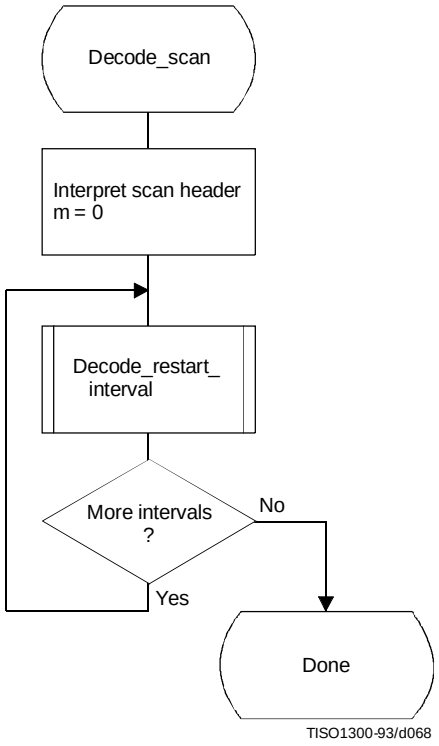


Figure E.8 – Control procedure for decoding a scan

E.2.4 Control procedure for decoding a restart interval

The procedure for decoding a restart interval is shown in Figure E.9. The "Reset_decoder" procedure consists at least of the following:

a) if arithmetic coding is used, initialize the arithmetic decoder using the "Initdec" procedure described in D.2.7;

b) for DCT-based processes, set the DC prediction (PRED) to zero for all components in the scan (see F.2.1.3.1);

c) for lossless process, reset the prediction to a default value for all components in the scan (see H.2.1);

d) do all other implementation-dependent setups that may be necessary.

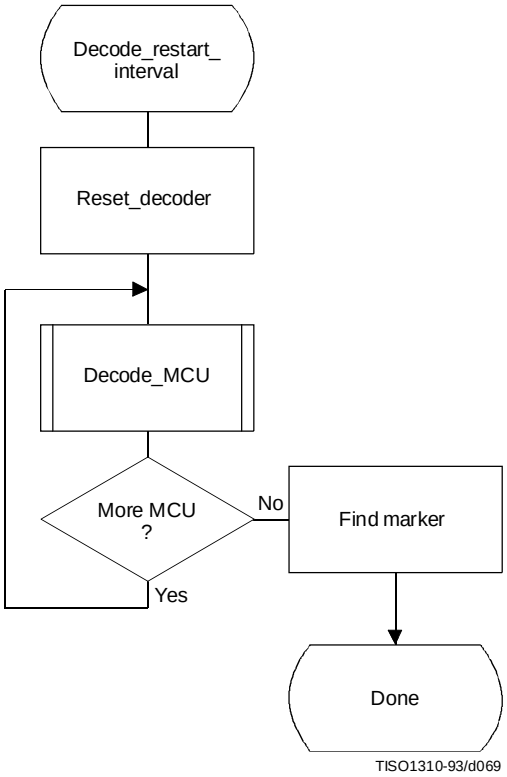


Figure E.9 – Control procedure for decoding a restart interval

At the end of the restart interval, the next marker is located. If a problem is detected in locating this marker, error handling procedures may be invoked. While such procedures are optional, the decoder shall be able to correctly recognize restart markers in the compressed data and reset the decoder when they are encountered. The decoder shall also be able to recognize the DNL marker, set the number of lines defined in the DNL segment, and end the "Decode_restart_interval" procedure.

NOTE – The final restart interval may be smaller than the size specified by the DRI marker segment, as it includes only the number of MCUs remaining in the scan.

E.2.5 Control procedure for decoding a minimum coded unit (MCU)

The procedure for decoding a minimum coded unit (MCU) is shown in Figure E.10.

In Figure E.10 Nb is the number of data units in a MCU.

The procedures for decoding a data unit are specified in Annexes F, G, and H.

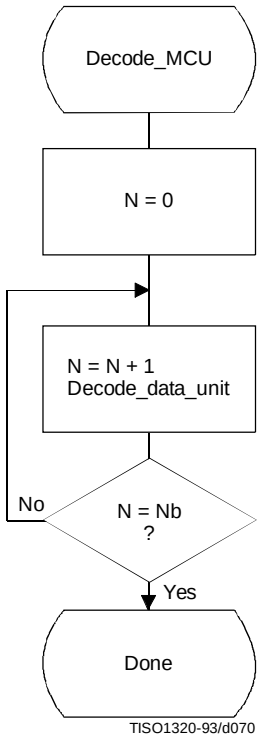


Figure E.10 – Control procedure for decoding a minimum coded unit (MCU)

Annex F

Sequential DCT-based mode of operation

(This annex forms an integral part of this Recommendation | International Standard)

This annex provides a **functional specification** of the following coding processes for the sequential DCT-based mode of operation:

1) baseline sequential;
2) extended sequential, Huffman coding, 8-bit sample precision;
3) extended sequential, arithmetic coding, 8-bit sample precision;
4) extended sequential, Huffman coding, 12-bit sample precision;
5) extended sequential, arithmetic coding, 12-bit sample precision.

For each of these, the encoding process is specified in F.1, and the decoding process is specified in F.2. The functional specification is presented by means of specific flow charts for the various procedures which comprise these coding processes.

NOTE – There is **no requirement** in this Specification that any encoder or decoder which embodies one of the above-named processes shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

F.1 Sequential DCT-based encoding processes

F.1.1 Sequential DCT-based control procedures and coding models

F.1.1.1 Control procedures for sequential DCT-based encoders

The control procedures for encoding an image and its constituent parts – the frame, scan, restart interval and MCU – are given in Figures E.1 to E.5. The procedure for encoding a MCU (see Figure E.5) repetitively calls the procedure for encoding a data unit. For DCT-based encoders the data unit is an 8×8 block of samples.

F.1.1.2 Procedure for encoding an 8×8 block data unit

For the sequential DCT-based processes encoding an 8×8 block data unit consists of the following procedures:

a) level shift, calculate forward 8×8 DCT and quantize the resulting coefficients using table destination specified in frame header;

b) encode DC coefficient for 8×8 block using DC table destination specified in scan header;

c) encode AC coefficients for 8×8 block using AC table destination specified in scan header.

F.1.1.3 Level shift and forward DCT (FDCT)

The mathematical definition of the FDCT is given in A.3.3.

Prior to computing the FDCT the input data are level shifted to a signed two's complement representation as described in A.3.1. For 8-bit input precision the level shift is achieved by subtracting 128. For 12-bit input precision the level shift is achieved by subtracting 2048.

F.1.1.4 Quantization of the FDCT

The uniform quantization procedure described in Annex A is used to quantize the DCT coefficients. One of four quantization tables may be used by the encoder. No default quantization tables are specified in this Specification. However, some typical quantization tables are given in Annex K.

The quantized DCT coefficient values are signed, two's complement integers with 11-bit precision for 8-bit input precision and 15-bit precision for 12-bit input precision.

F.1.1.5 Encoding models for the sequential DCT procedures

The two dimensional array of quantized DCT coefficients is rearranged in a zig-zag sequence order defined in A.3.6. The zig-zag order coefficients are denoted ZZ (0) through ZZ(63) with:

$$ZZ(0) = Sq_{00}, ZZ(1) = Sq_{01}, ZZ(2) = Sq_{10}, \bullet, \bullet, \bullet, ZZ(63) = Sq_{77}$$

Sq_{vu} are defined in Figure A.6.

Two coding procedures are used, one for the DC coefficient ZZ(0) and the other for the AC coefficients ZZ(1)..ZZ(63). The coefficients are encoded in the order in which they occur in zig-zag sequence order, starting with the DC coefficient. The coefficients are represented as two's complement integers.

F.1.1.5.1 Encoding model for DC coefficients

The DC coefficients are coded differentially, using a one-dimensional predictor, PRED, which is the quantized DC value from the most recently coded 8×8 block from the same component. The difference, DIFF, is obtained from

$$DIFF = ZZ(0) - PRED$$

At the beginning of the scan and at the beginning of each restart interval, the prediction for the DC coefficient prediction is initialized to 0. (Recall that the input data have been level shifted to two's complement representation.)

F.1.1.5.2 Encoding model for AC coefficients

Since many coefficients are zero, runs of zeros are identified and coded efficiently. In addition, if the remaining coefficients in the zig-zag sequence order are all zero, this is coded explicitly as an end-of-block (EOB).

F.1.2 Baseline Huffman encoding procedures

The baseline encoding procedure is for 8-bit sample precision. The encoder may employ up to two DC and two AC Huffman tables within one scan.

F.1.2.1 Huffman encoding of DC coefficients

F.1.2.1.1 Structure of DC code table

The DC code table consists of a set of Huffman codes (maximum length 16 bits) and appended additional bits (in most cases) which can code any possible value of DIFF, the difference between the current DC coefficient and the prediction. The Huffman codes for the difference categories are generated in such a way that no code consists entirely of 1-bits (X'FF' prefix marker code avoided).

The two's complement difference magnitudes are grouped into 12 categories, SSSS, and a Huffman code is created for each of the 12 difference magnitude categories (see Table F.1).

For each category, except SSSS = 0, an additional bits field is appended to the code word to uniquely identify which difference in that category actually occurred. The number of extra bits is given by SSSS; the extra bits are appended to the LSB of the preceding Huffman code, most significant bit first. When DIFF is positive, the SSSS low order bits of DIFF are appended. When DIFF is negative, the SSSS low order bits of (DIFF – 1) are appended. Note that the most significant bit of the appended bit sequence is 0 for negative differences and 1 for positive differences.

F.1.2.1.2 Defining Huffman tables for the DC coefficients

The syntax for specifying the Huffman tables is given in Annex B. The procedure for creating a code table from this information is described in Annex C. No more than two Huffman tables may be defined for coding of DC coefficients. Two examples of Huffman tables for coding of DC coefficients are provided in Annex K.

Table F.1 – Difference magnitude categories for DC coding

SSSS	DIFF values
0	0
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023
11	–2 047..–1 024,1 024..2 047

F.1.2.1.3 Huffman encoding procedures for DC coefficients

The encoding procedure is defined in terms of a set of extended tables, XHUFCO and XHUFSI, which contain the complete set of Huffman codes and sizes for all possible difference values. For full 12-bit precision the tables are relatively large. For the baseline system, however, the precision of the differences may be small enough to make this description practical.

XHUFCO and XHUFSI are generated from the encoder tables EHUFCO and EHUFSI (see Annex C) by appending to the Huffman codes for each difference category the additional bits that completely define the difference. By definition, XHUFCO and XHUFSI have entries for each possible difference value. XHUFCO contains the concatenated bit pattern of the Huffman code and the additional bits field; XHUFSI contains the total length in bits of this concatenated bit pattern. Both are indexed by DIFF, the difference between the DC coefficient and the prediction.

The Huffman encoding procedure for the DC difference, DIFF, is:

SIZE = XHUFSI(DIFF)

CODE = XHUFCO(DIFF)

code SIZE bits of CODE

where DC is the quantized DC coefficient value and PRED is the predicted quantized DC value. The Huffman code (CODE) (including any additional bits) is obtained from XHUFCO and SIZE (length of the code including additional bits) is obtained from XHUFSI, using DIFF as the index to the two tables.

F.1.2.2 Huffman encoding of AC coefficients

F.1.2.2.1 Structure of AC code table

Each non-zero AC coefficient in ZZ is described by a composite 8-bit value, RS, of the form

RS = binary 'RRRRSSSS'

The 4 least significant bits, 'SSSS', define a category for the amplitude of the next non-zero coefficient in ZZ, and the 4 most significant bits, 'RRRR', give the position of the coefficient in ZZ relative to the previous non-zero coefficient (i.e. the run-length of zero coefficients between non-zero coefficients). Since the run length of zero coefficients may exceed 15, the value 'RRRRSSSS' = X'F0' is defined to represent a run length of 15 zero coefficients followed by a coefficient of zero amplitude. (This can be interpreted as a run length of 16 zero coefficients.) In addition, a special value 'RRRRSSSS' = '00000000' is used to code the end-of-block (EOB), when all remaining coefficients in the block are zero.

The general structure of the code table is illustrated in Figure F.1. The entries marked "N/A" are undefined for the baseline procedure.

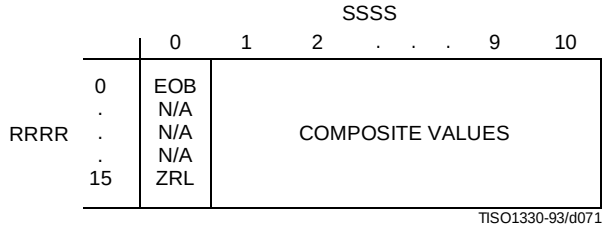


Figure F.1 – Two-dimensional value array for Huffman coding

The magnitude ranges assigned to each value of SSSS are defined in Table F.2.

Table F.2 – Categories assigned to coefficient values

SSSS	AC coefficients
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023

The composite value, RRRRSSSS, is Huffman coded and each Huffman code is followed by additional bits which specify the sign and exact amplitude of the coefficient.

The AC code table consists of one Huffman code (maximum length 16 bits, not including additional bits) for each possible composite value. The Huffman codes for the 8-bit composite values are generated in such a way that no code consists entirely of 1-bits.

The format for the additional bits is the same as in the coding of the DC coefficients. The value of SSSS gives the number of additional bits required to specify the sign and precise amplitude of the coefficient. The additional bits are either the low-order SSSS bits of ZZ(K) when ZZ(K) is positive or the low-order SSSS bits of ZZ(K) – 1 when ZZ(K) is negative. ZZ(K) is the *K*th coefficient in the zig-zag sequence of coefficients being coded.

F.1.2.2.2 Defining Huffman tables for the AC coefficients

The syntax for specifying the Huffman tables is given in Annex B. The procedure for creating a code table from this information is described in Annex C.

In the baseline system no more than two Huffman tables may be defined for coding of AC coefficients. Two examples of Huffman tables for coding of AC coefficients are provided in Annex K.

F.1.2.2.3 Huffman encoding procedures for AC coefficients

As defined in Annex C, the Huffman code table is assumed to be available as a pair of tables, EHUFCO (containing the code bits) and EHUFSI (containing the length of each code in bits), both indexed by the composite value defined above.

The procedure for encoding the AC coefficients in a block is shown in Figures F.2 and F.3. In Figure F.2, K is the index to the zig-zag scan position and R is the run length of zero coefficients.

The procedure "Append EHUFSI(X'F0') bits of EHUFCO(X'F0')" codes a run of 16 zero coefficients (ZRL code of Figure F.1). The procedure "Code EHUFSI(0) bits of EHUFCO(0)" codes the end-of-block (EOB code). If the last coefficient (K = 63) is not zero, the EOB code is bypassed.

CSIZE is a procedure which maps an AC coefficient to the SSSS value as defined in Table F.2.

F.1.2.3 Byte stuffing

In order to provide code space for marker codes which can be located in the compressed image data without decoding, byte stuffing is used.

Whenever, in the course of normal encoding, the byte value X'FF' is created in the code string, a X'00' byte is stuffed into the code string.

If a X'00' byte is detected after a X'FF' byte, the decoder must discard it. If the byte is not zero, a marker has been detected, and shall be interpreted to the extent needed to complete the decoding of the scan.

Byte alignment of markers is achieved by padding incomplete bytes with 1-bits. If padding with 1-bits creates a X'FF' value, a zero byte is stuffed before adding the marker.

F.1.3 Extended sequential DCT-based Huffman encoding process for 8-bit sample precision

This process is identical to the Baseline encoding process described in F.1.2, with the exception that the number of sets of Huffman table destinations which may be used within the same scan is increased to four. Four DC and four AC Huffman table destinations is the maximum allowed by this Specification.

F.1.4 Extended sequential DCT-based arithmetic encoding process for 8-bit sample precision

This subclause describes the use of arithmetic coding procedures in the sequential DCT-based encoding process.

NOTE – The arithmetic coding procedures in this Specification are defined for the maximum precision to encourage interchangeability.

The arithmetic coding extensions have the same DCT model as the Baseline DCT encoder. Therefore, Annex F.1.1 also applies to arithmetic coding. As with the Huffman coding technique, the binary arithmetic coding technique is lossless. It is possible to transcode between the two systems without either FDCT or IDCT computations, and without modification of the reconstructed image.

The basic principles of adaptive binary arithmetic coding are described in Annex D. Up to four DC and four AC conditioning table destinations and associated statistics areas may be used within one scan.

The arithmetic encoding procedures for encoding binary decisions, initializing the statistics area, initializing the encoder, terminating the code string, and adding restart markers are listed in Table D.1 of Annex D.

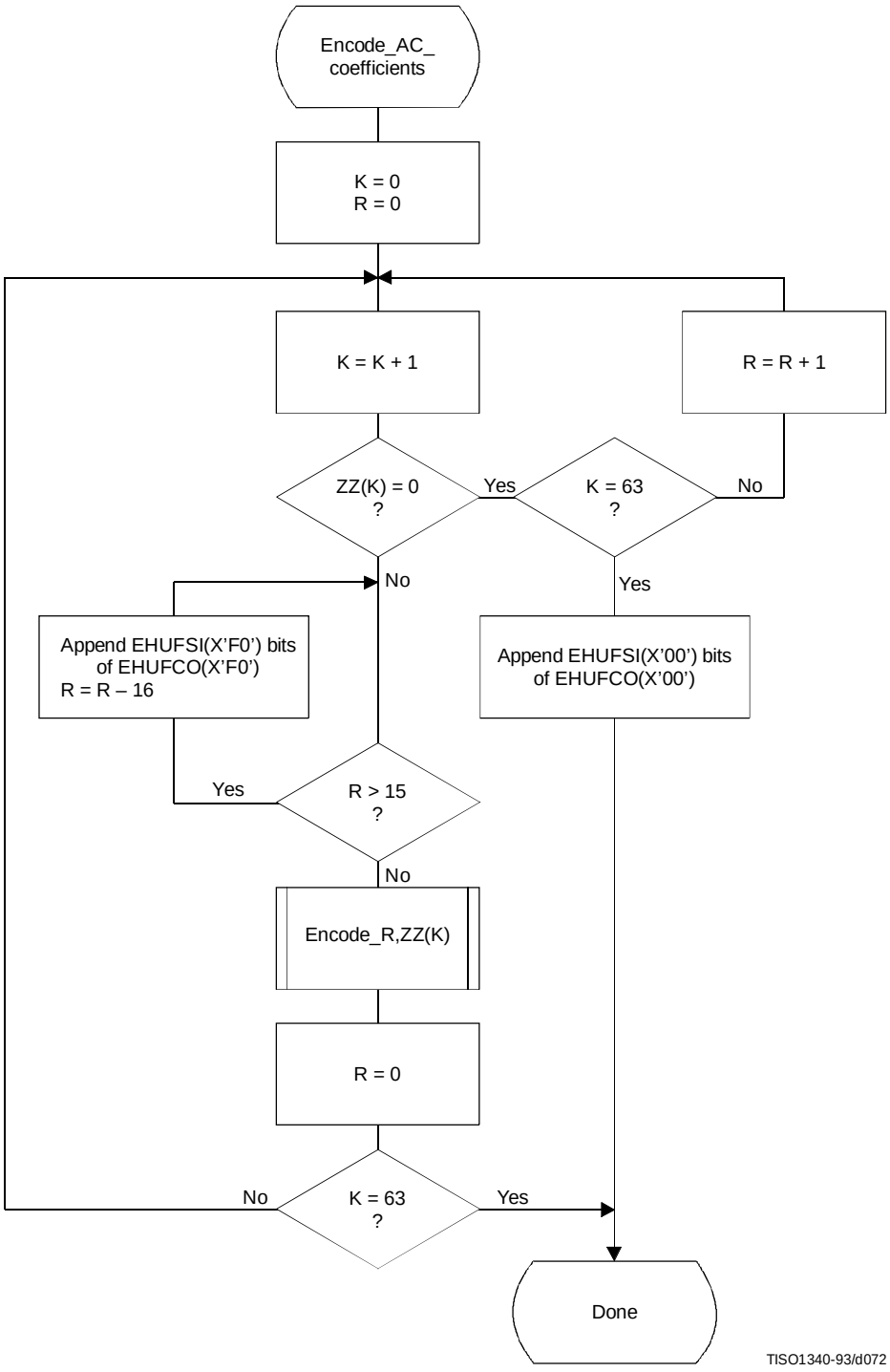


Figure F.2 – Procedure for sequential encoding of AC coefficients with Huffman coding

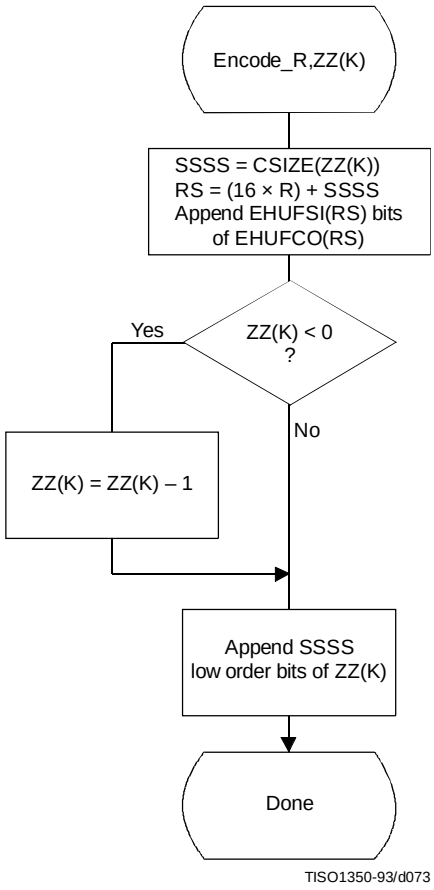


Figure F.3 – Sequential encoding of a non-zero AC coefficient

Some of the procedures in Table D.1 are used in the higher level control structure for scans and restart intervals described in Annex E. At the beginning of scans and restart intervals, the probability estimates used in the arithmetic coder are reset to the standard initial value as part of the Initenc procedure which restarts the arithmetic coder. At the end of scans and restart intervals, the Flush procedure is invoked to empty the code register before the next marker is appended.

F.1.4.1 Arithmetic encoding of DC coefficients

The basic structure of the decision sequence for encoding a DC difference value, DIFF, is shown in Figure F.4.

The context-index S0 and other context-indices used in the DC coding procedures are defined in Table F.4 (see F.1.4.4.1.3). A 0-decision is coded if the difference value is zero and a 1-decision is coded if the difference is not zero. If the difference is not zero, the sign and magnitude are coded using the procedure Encode_V(S0), which is described in F.1.4.3.1.

F.1.4.2 Arithmetic encoding of AC coefficients

The AC coefficients are coded in the order in which they occur in the zig-zag sequence ZZ(1,...,63). An end-of-block (EOB) binary decision is coded before coding the first AC coefficient in ZZ, and after each non-zero coefficient. If the EOB occurs, all remaining coefficients in ZZ are zero. Figure F.5 illustrates the decision sequence. The equivalent procedure for the Huffman coder is found in Figure F.2.

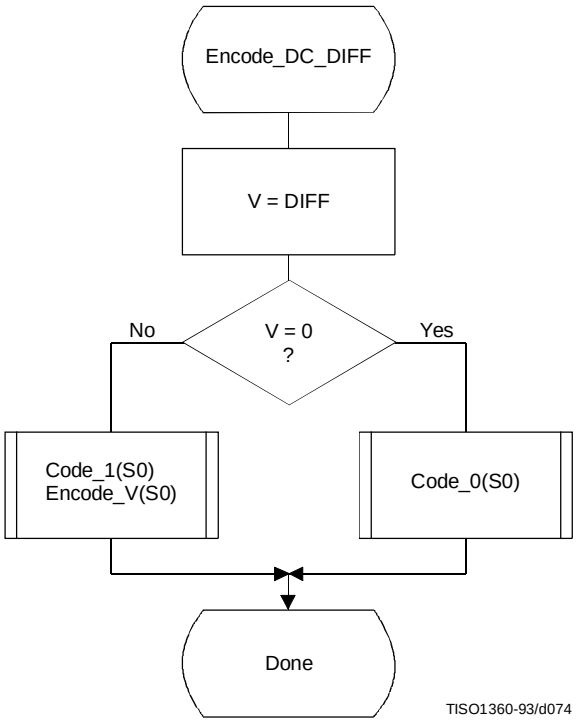


Figure F.4 – Coding model for arithmetic coding of DC difference

The context-indices SE and S0 used in the AC coding procedures are defined in Table F.5 (see F.1.4.4.2). In Figure F.5, K is the index to the zig-zag sequence position. For the sequential scan, Kmin is 1 and Se is 63. The V = 0 decision is part of a loop which codes runs of zero coefficients. Whenever the coefficient is non-zero, "Encode_V(S0)" codes the sign and magnitude of the coefficient. Each time a non-zero coefficient is coded, it is followed by an EOB decision. If the EOB occurs, a 1-decision is coded to indicate that the coding of the block is complete. If the coefficient for K = Se is not zero, the EOB decision is skipped.

F.1.4.3 Encoding the binary decision sequence for non-zero DC differences and AC coefficients

Both the DC difference and the AC coefficients are represented as signed two's complement integer values. The decomposition of these signed integer values into a binary decision tree is done in the same way for both the DC and AC coding models.

Although the binary decision trees for this section of the DC and AC coding models are the same, the statistical models for assigning statistics bins to the binary decisions in the tree are quite different.

F.1.4.3.1 Structure of the encoding decision sequence

The encoding sequence can be separated into three procedures, a procedure which encodes the sign, a second procedure which identifies the magnitude category, and a third procedure which identifies precisely which magnitude occurred within the category identified in the second procedure.

At the point where the binary decision sequence in Encode_V(S0) starts, the coefficient or difference has already been determined to be non-zero. That determination was made in the procedures in Figures F.4 and F.5.

Denoting either DC differences (DIFF) or AC coefficients as V, the non-zero signed integer value of V is encoded by the sequence shown in Figure F.6. This sequence first codes the sign of V. It then (after converting V to a magnitude and decrementing it by 1 to give Sz) codes the magnitude category of Sz (code_log2_Sz), and then codes the low order magnitude bits (code_Sz_bits) to identify the exact magnitude value.

There are two significant differences between this sequence and the similar set of operations described in F.1.2 for Huffman coding. First, the sign is encoded before the magnitude category is identified, and second, the magnitude is decremented by 1 before the magnitude category is identified.

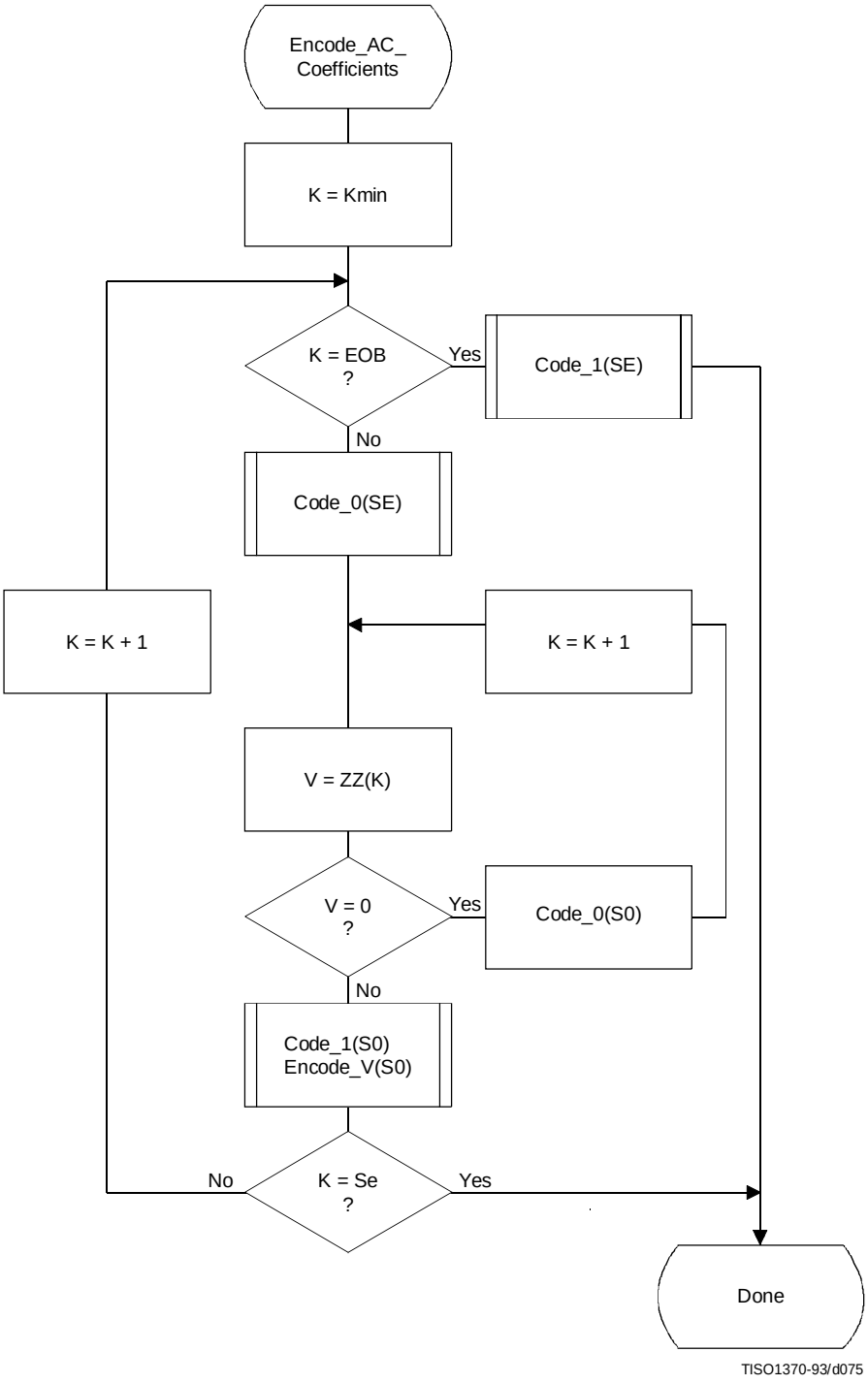


Figure F.5 – AC coding model for arithmetic coding

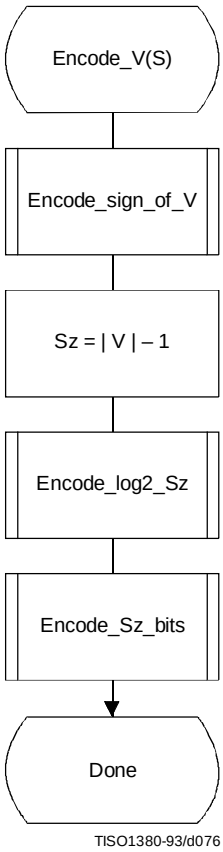


Figure F.6 – Sequence of procedures in encoding non-zero values of V

F.1.4.3.1.1 Encoding the sign

The sign is encoded by coding a 0-decision when the sign is positive and a 1-decision when the sign is negative (see Figure F.7).

The context-indices SS, SN and SP are defined for DC coding in Table F.4 and for AC coding in Table F.5. After the sign is coded, the context-index S is set to either SN or SP, establishing an initial value for Encode_log2_Sz.

F.1.4.3.1.2 Encoding the magnitude category

The magnitude category is determined by a sequence of binary decisions which compares Sz against an exponentially increasing bound (which is a power of 2) in order to determine the position of the leading 1-bit. This establishes the magnitude category in much the same way that the Huffman encoder generates a code for the value associated with the difference category. The flow chart for this procedure is shown in Figure F.8.

The starting value of the context-index S is determined in Encode_sign_of_V, and the context-index values X1 and X2 are defined for DC coding in Table F.4 and for AC coding in Table F.5. In Figure F.8, M is the exclusive upper bound for the magnitude and the abbreviations “SLL” and “SRL” refer to the shift-left-logical and shift-right-logical operations – in this case by one bit position. The SRL operation at the completion of the procedure aligns M with the most significant bit of Sz (see Table F.3).

The highest precision allowed for the DCT is 15 bits. Therefore, the highest precision required for the coding decision tree is 16 bits for the DC coefficient difference and 15 bits for the AC coefficients, including the sign bit.

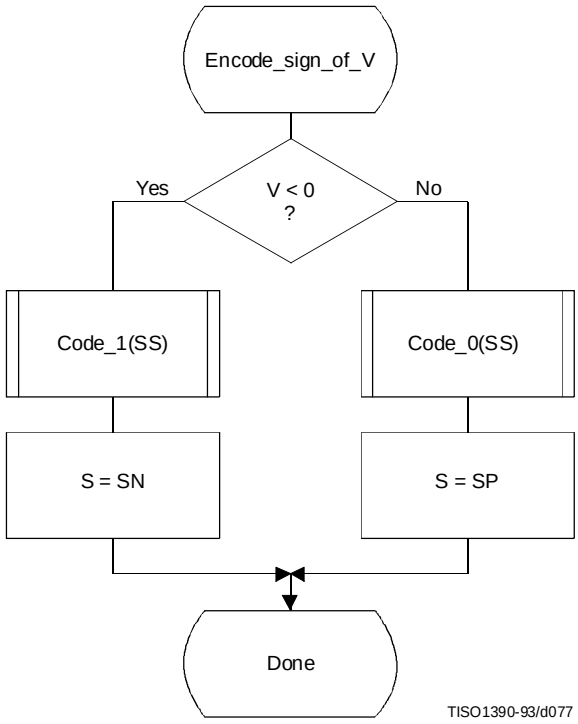


Figure F.7 – Encoding the sign of V

Table F.3 – Categories for each maximum bound

Exclusive upper bound (M)	Sz range	Number of low order magnitude bits
1	0	0
2	1	0
4	2,3	1
8	4,...,7	2
16	8,...,15	3
32	16,...,31	4
64	32,...,63	5
128	64,...,127	6
256	128,...,255	7
512	256,...,511	8
1 024	512,...,1 023	9
2 048	1 024,...,2 047	10
4 096	2 048,...,4 095	11
8 192	4 096,...,8 191	12
16 384	8 192,...,16 383	13
32 768	16 384,...,32 767	14

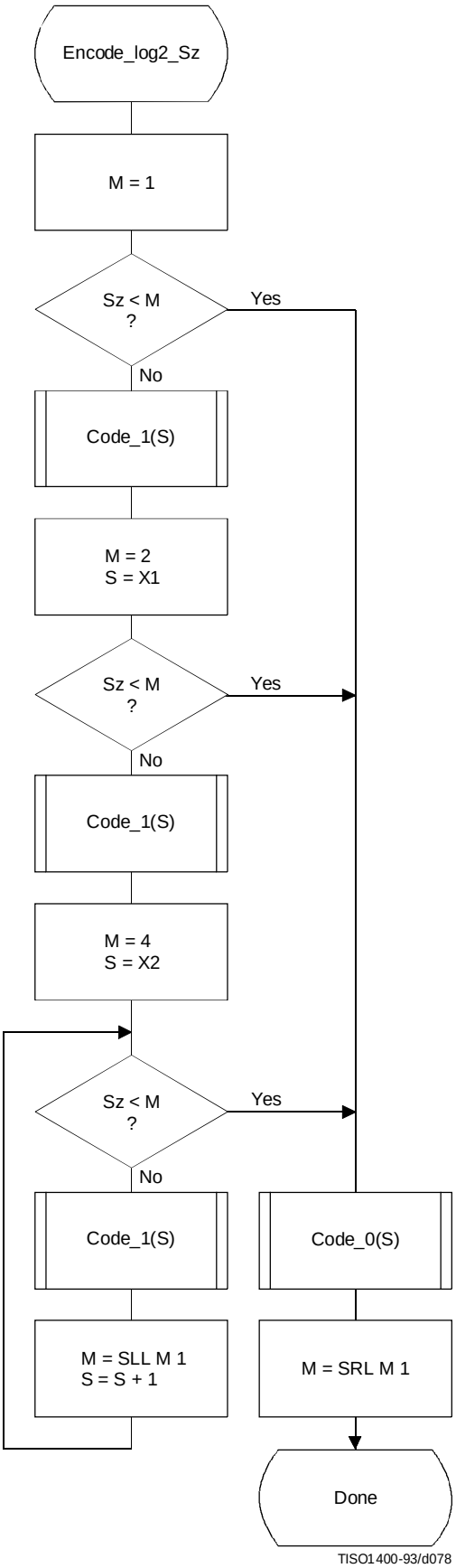


Figure F.8 – Decision sequence to establish the magnitude category

F.1.4.3.1.3 Encoding the exact value of the magnitude

After the magnitude category is encoded, the low order magnitude bits are encoded. These bits are encoded in order of decreasing bit significance. The procedure is shown in Figure F.9. The abbreviation "SRL" indicates the shift-right-logical operation, and M is the exclusive bound established in Figure F.8. Note that M has only one bit set – shifting M right converts it into a bit mask for the logical "AND" operation.

The starting value of the context-index S is determined in Encode_log2_Sz. The increment of S by 14 at the beginning of this procedure sets the context-index to the value required in Tables F.4 and F.5.

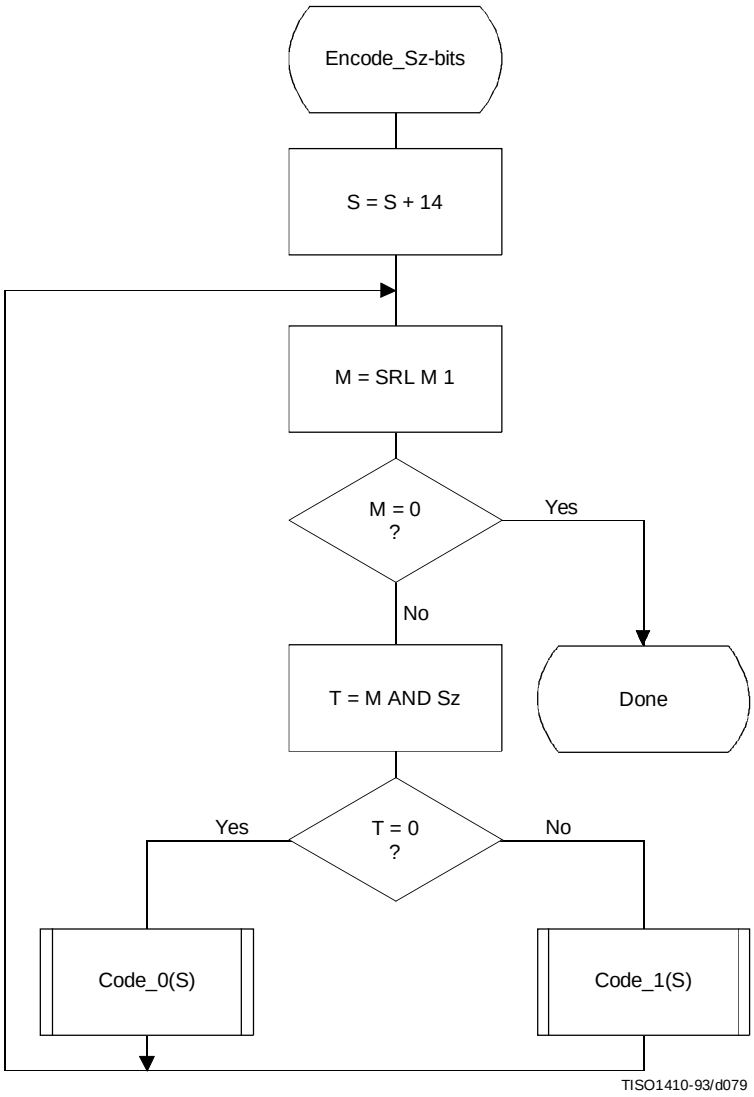


Figure F.9 – Decision sequence to code the magnitude bit pattern

F.1.4.4 Statistical models

An adaptive binary arithmetic coder requires a statistical model. The statistical model defines the contexts which are used to select the conditional probability estimates used in the encoding and decoding procedures.

Each decision in the binary decision trees is associated with one or more contexts. These contexts identify the sense of the MPS and the index in Table D.3 of the conditional probability estimate Qe which is used to encode and decode the binary decision.

The arithmetic coder is adaptive, which means that the probability estimates for each context are developed and maintained by the arithmetic coding system on the basis of prior coding decisions for that context.

F.1.4.4.1 Statistical model for coding DC prediction differences

The statistical model for coding the DC difference conditions some of the probability estimates for the binary decisions on previous DC coding decisions.

F.1.4.4.1.1 Statistical conditioning on sign

In coding the DC coefficients, four separate statistics bins (probability estimates) are used in coding the zero/not-zero (V = 0) decision, the sign decision and the first magnitude category decision. Two of these bins are used to code the V = 0 decision and the sign decision. The other two bins are used in coding the first magnitude decision, Sz < 1; one of these bins is used when the sign is positive, and the other is used when the sign is negative. Thus, the first magnitude decision probability estimate is conditioned on the sign of V.

F.1.4.4.1.2 Statistical conditioning on DC difference in previous block

The probability estimates for these first three decisions are also conditioned on Da, the difference value coded for the previous DCT block of the same component. The differences are classified into five groups: zero, small positive, small negative, large positive and large negative. The relationship between the default classification and the quantization scale is shown in Figure F.10.

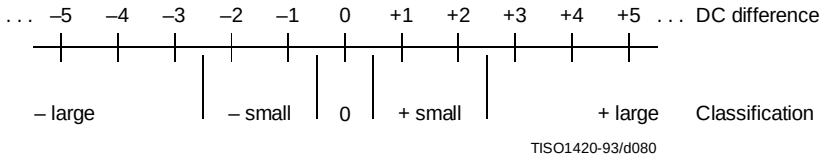


Figure F.10 – Conditioning classification of difference values

The bounds for the “small” difference category determine the classification. Defining L and U as integers in the range 0 to 15 inclusive, the lower bound (exclusive) for difference magnitudes classified as “small” is zero for L = 0, and is 2^{L-1} for L > 0.

The upper bound (inclusive) for difference magnitudes classified as “small” is 2^{U}.

L shall be less than or equal to U.

These bounds for the conditioning category provide a segmentation which is identical to that listed in Table F.3.

F.1.4.4.1.3 Assignment of statistical bins to the DC binary decision tree

As shown in Table F.4, each statistics area for DC coding consists of a set of 49 statistics bins. In the following explanation, it is assumed that the bins are contiguous. The first 20 bins consist of five sets of four bins selected by a context-index S0. The value of S0 is given by DC_Context(Da), which provides a value of 0, 4, 8, 12 or 16, depending on the difference classification of Da (see F.1.4.4.1.2). The remaining 29 bins, X1,...,X15,M2,...,M15, are used to code magnitude category decisions and magnitude bits.

Table F.4 – Statistical model for DC coefficient coding

Context-index	Value	Coding decision
S0	DC_Context(Da)	V = 0
SS	S0 + 1	Sign of V
SP	S0 + 2	Sz < 1 if V > 0
SN	S0 + 3	Sz < 1 if V < 0
X1	20	Sz < 2
X2	X1 + 1	Sz < 4
X3	X1 + 2	Sz < 8
.	.	.
.	.	.
X15	X1 + 14	$Sz < 2^{15}$
M2	X2 + 14	Magnitude bits if Sz < 4
M3	X3 + 14	Magnitude bits if Sz < 8
.	.	.
.	.	.
M15	X15 + 14	Magnitude bits if $Sz < 2^{15}$

F.1.4.4.1.4 Default conditioning for DC statistical model

The bounds, L and U, for determining the conditioning category have the default values L = 0 and U = 1. Other bounds may be set using the DAC (Define Arithmetic coding Conditioning) marker segment, as described in Annex B.

F.1.4.4.1.5 Initial conditions for DC statistical model

At the start of a scan and at the beginning of each restart interval, the difference for the previous DC value is defined to be zero in determining the conditioning state.

F.1.4.4.2 Statistical model for coding the AC coefficients

As shown in Table F.5, each statistics area for AC coding consists of a contiguous set of 245 statistics bins. Three bins are used for each value of the zig-zag index K, and two sets of 28 additional bins X2,...,X15,M2,...,M15 are used for coding the magnitude category and magnitude bits.

The value of SE (and also S0, SP and SN) is determined by the zig-zag index K. Since K is in the range 1 to 63, the lowest value for SE is 0 and the largest value for SP is 188. SS is not assigned a value in AC coefficient coding, as the signs of the coefficients are coded with a fixed probability value of approximately 0.5 (Qe = X'5A1D', MPS = 0).

The value of X2 is given by AC_Context(K). This gives X2 = 189 when $K \leq Kx$ and X2 = 217 when K > Kx, where Kx is defined using the DAC marker segment (see B.2.4.3).

Note that a X1 statistics bin is not used in this sequence. Instead, the 63×1 array of statistics bins for the magnitude category is used for two decisions. Once the magnitude bound has been determined – at statistics bin Xn, for example – a single statistics bin, Mn, is used to code the magnitude bit sequence for that bound.

F.1.4.4.2.1 Default conditioning for AC coefficient coding

The default value of Kx is 5. This may be modified using the DAC marker segment, as described in Annex B.

F.1.4.4.2.2 Initial conditions for AC statistical model

At the start of a scan and at each restart, all statistics bins are re-initialized to the standard default value described in Annex D.

Table F.5 – Statistical model for AC coefficient coding

Context-index	Value	Coding decision
SE	$3 \times (K - 1)$	K = EOB
S0	SE + 1	V = 0
SS	Fixed estimate	Sign of V
SN,SP	S0 + 1	Sz < 1
X1	S0 + 1	Sz < 2
X2	AC_Context(K)	Sz < 4
X3	X2 + 1	Sz < 8
.	.	.
.	.	.
X15	X2 + 13	$Sz < 2^{15}$
M2	X2 + 14	Magnitude bits if Sz < 4
M3	X3 + 14	Magnitude bits if Sz < 8
.	.	.
.	.	.
M15	X15 + 14	Magnitude bits if $Sz < 2^{15}$

F.1.5 Extended sequential DCT-based Huffman encoding process for 12-bit sample precision

This process is identical to the sequential DCT process for 8-bit precision extended to four Huffman table destinations as documented in F.1.3, with the following changes.

F.1.5.1 Structure of DC code table for 12-bit sample precision

The two's complement difference magnitudes are grouped into 16 categories, SSSS, and a Huffman code is created for each of the 16 difference magnitude categories.

The Huffman table for DC coding (see Table F.1) is extended as shown in Table F.6.

Table F.6 – Difference magnitude categories for DC coding

SSSS	Difference values
12	–4 095..–2 048,2 048..4 095
13	–8 191..–4 096,4 096..8 191
14	–16 383..–8 192,8 192..16 383
15	–32 767..–16 384,16 384..32 767

F.1.5.2 Structure of AC code table for 12-bit sample precision

The general structure of the code table is extended as illustrated in Figure F.11. The Huffman table for AC coding is extended as shown in Table F.7.

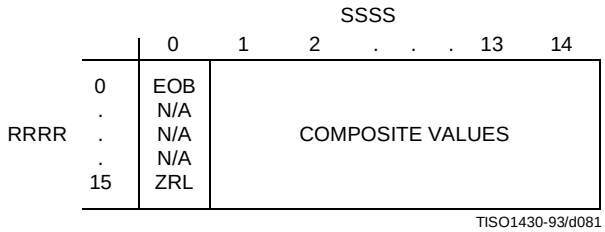


Figure F.11 – Two-dimensional value array for Huffman coding

Table F.7 – Values assigned to coefficient amplitude ranges

SSSS	AC coefficients
11	–2 047..–1 024,1 024..2 047
12	–4 095..–2 048,2 048..4 095
13	–8 191..–4 096,4 096..8 191
14	–16 383..–8 192,8 192..16 383

F.1.6 Extended sequential DCT-based arithmetic encoding process for 12-bit sample precision

The process is identical to the sequential DCT process for 8-bit precision except for changes in the precision of the FDCT computation.

The structure of the encoding procedure is identical to that specified in F.1.4 which was already defined for a 12-bit sample precision.

F.2 Sequential DCT-based decoding processes

F.2.1 Sequential DCT-based control procedures and coding models

F.2.1.1 Control procedures for sequential DCT-based decoders

The control procedures for decoding compressed image data and its constituent parts – the frame, scan, restart interval and MCU – are given in Figures E.6 to E.10. The procedure for decoding a MCU (Figure E.10) repetitively calls the procedure for decoding a data unit. For DCT-based decoders the data unit is an 8×8 block of samples.

F.2.1.2 Procedure for decoding an 8×8 block data unit

In the sequential DCT-based decoding process, decoding an 8×8 block data unit consists of the following procedures:

a) decode DC coefficient for 8×8 block using the DC table destination specified in the scan header;

b) decode AC coefficients for 8×8 block using the AC table destination specified in the scan header;

c) dequantize using table destination specified in the frame header and calculate the inverse 8×8 DCT.

F.2.1.3 Decoding models for the sequential DCT procedures

Two decoding procedures are used, one for the DC coefficient ZZ(0) and the other for the AC coefficients ZZ(1)...ZZ(63). The coefficients are decoded in the order in which they occur in the zig-zag sequence order, starting with the DC coefficient. The coefficients are represented as two's complement integers.

F.2.1.3.1 Decoding model for DC coefficients

The decoded difference, DIFF, is added to PRED, the DC value from the most recently decoded 8×8 block from the same component. Thus ZZ(0) = PRED + DIFF.

At the beginning of the scan and at the beginning of each restart interval, the prediction for the DC coefficient is initialized to zero.

F.2.1.3.2 Decoding model for AC coefficients

The AC coefficients are decoded in the order in which they occur in ZZ. When the EOB is decoded, all remaining coefficients in ZZ are initialized to zero.

F.2.1.4 Dequantization of the quantized DCT coefficients

The dequantization of the quantized DCT coefficients as described in Annex A, is accomplished by multiplying each quantized coefficient value by the quantization table value for that coefficient. The decoder shall be able to use up to four quantization table destinations.

F.2.1.5 Inverse DCT (IDCT)

The mathematical definition of the IDCT is given in A.3.3.

After computation of the IDCT, the signed output samples are level-shifted, as described in Annex A, converting the output to an unsigned representation. For 8-bit precision the level shift is performed by adding 128. For 12-bit precision the level shift is performed by adding 2 048. If necessary, the output samples shall be clamped to stay within the range appropriate for the precision (0 to 255 for 8-bit precision and 0 to 4 095 for 12-bit precision).

F.2.2 Baseline Huffman Decoding procedures

The baseline decoding procedure is for 8-bit sample precision. The decoder shall be capable of using up to two DC and two AC Huffman tables within one scan.

F.2.2.1 Huffman decoding of DC coefficients

The decoding procedure for the DC difference, DIFF, is:

T = DECODE

DIFF = RECEIVE(T)

DIFF = EXTEND(DIFF,T)

where DECODE is a procedure which returns the 8-bit value associated with the next Huffman code in the compressed image data (see F.2.2.3) and RECEIVE(T) is a procedure which places the next T bits of the serial bit string into the low order bits of DIFF, MSB first. If T is zero, DIFF is set to zero. EXTEND is a procedure which converts the partially decoded DIFF value of precision T to the full precision difference. EXTEND is shown in Figure F.12.

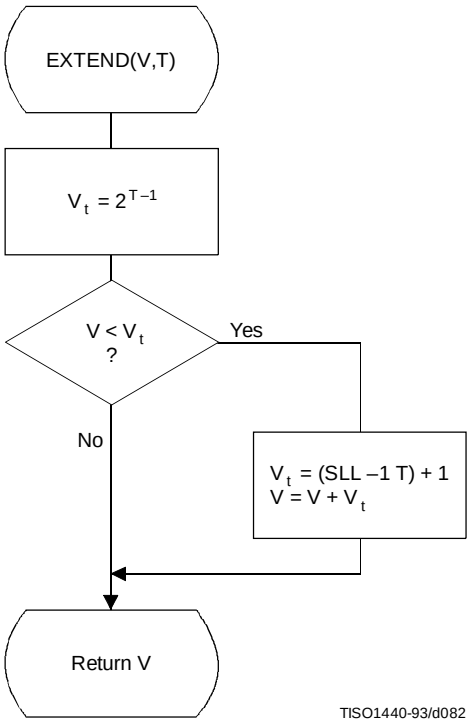


Figure F.12 – Extending the sign bit of a decoded value in V

F.2.2.2 Decoding procedure for AC coefficients

The decoding procedure for AC coefficients is shown in Figures F.13 and F.14.

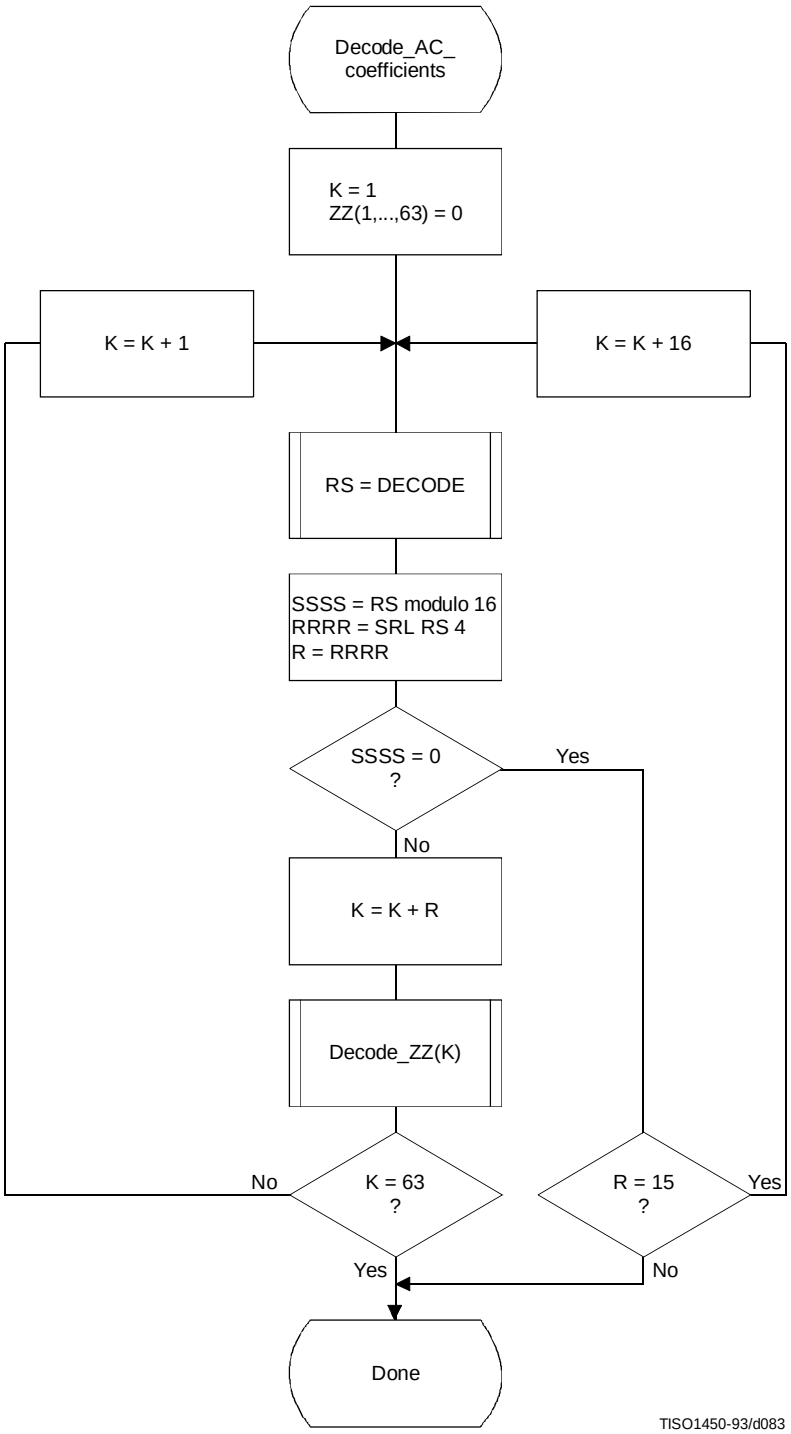


Figure F.13 – Huffman decoding procedure for AC coefficients

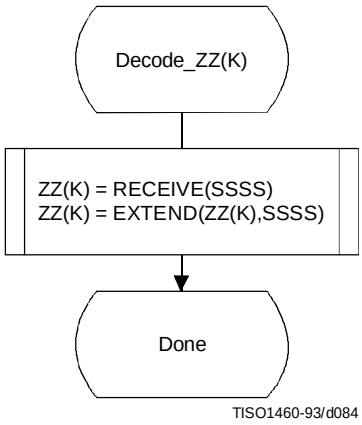


Figure F.14 – Decoding a non-zero AC coefficient

The decoding of the amplitude and sign of the non-zero coefficient is done in the procedure "Decode_ZZ(K)", shown in Figure F.14.

DECODE is a procedure which returns the value, RS, associated with the next Huffman code in the code stream (see F.2.2.3). The values SSSS and R are derived from RS. The value of SSSS is the four low order bits of the composite value and R contains the value of RRRR (the four high order bits of the composite value). The interpretation of these values is described in F.1.2.2. EXTEND is shown in Figure F.12.

F.2.2.3 The DECODE procedure

The DECODE procedure decodes an 8-bit value which, for the DC coefficient, determines the difference magnitude category. For the AC coefficient this 8-bit value determines the zero run length and non-zero coefficient category.

Three tables, HUFFVAL, HUFFCODE, and HUFFSIZE, have been defined in Annex C. This particular implementation of DECODE makes use of the ordering of the Huffman codes in HUFFCODE according to both value and code size. Many other implementations of DECODE are possible.

NOTE – The values in HUFFVAL are assigned to each code in HUFFCODE and HUFFSIZE in sequence. There are no ordering requirements for the values in HUFFVAL which have assigned codes of the same length.

The implementation of DECODE described in this subclause uses three tables, MINCODE, MAXCODE and VALPTR, to decode a pointer to the HUFFVAL table. MINCODE, MAXCODE and VALPTR each have 16 entries, one for each possible code size. MINCODE(I) contains the smallest code value for a given length I, MAXCODE(I) contains the largest code value for a given length I, and VALPTR(I) contains the index to the start of the list of values in HUFFVAL which are decoded by code words of length I. The values in MINCODE and MAXCODE are signed 16-bit integers; therefore, a value of –1 sets all of the bits.

The procedure for generating these tables is shown in Figure F.15. The procedure for DECODE is shown in Figure F.16. Note that the 8-bit "VALUE" is returned to the procedure which invokes DECODE.

APPENDIX F

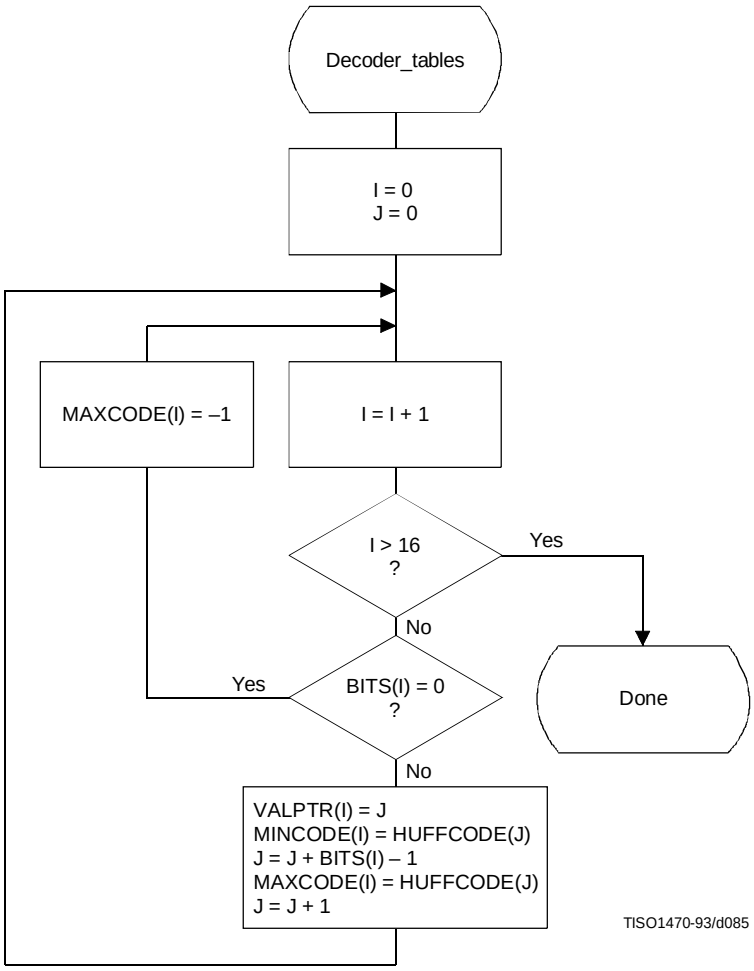


Figure F.15 – Decoder table generation

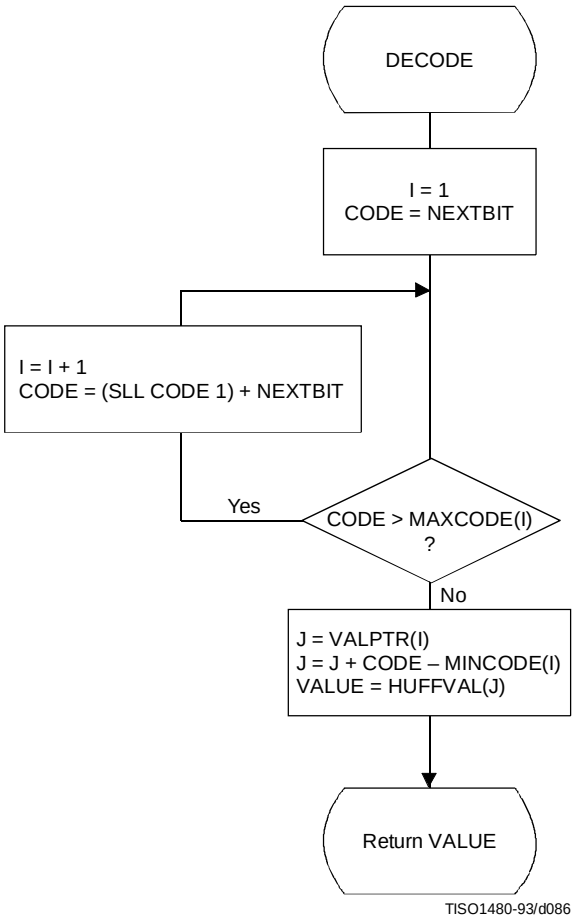


Figure F.16 – Procedure for DECODE

F.2.2.4 The RECEIVE procedure

RECEIVE(SSSS) is a procedure which places the next SSSS bits of the entropy-coded segment into the low order bits of DIFF, MSB first. It calls NEXTBIT and it returns the value of DIFF to the calling procedure (see Figure F.17).

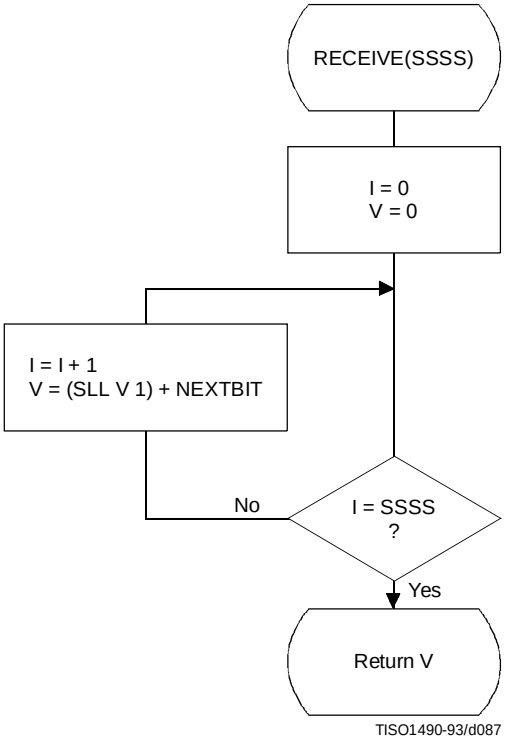


Figure F.17 – Procedure for RECEIVE(SSSS)

F.2.2.5 The NEXTBIT procedure

NEXTBIT reads the next bit of compressed data and passes it to higher level routines. It also intercepts and removes stuff bytes and detects markers. NEXTBIT reads the bits of a byte starting with the MSB (see Figure F.18).

Before starting the decoding of a scan, and after processing a RST marker, CNT is cleared. The compressed data are read one byte at a time, using the procedure NEXTBYTE. Each time a byte, B, is read, CNT is set to 8.

The only valid marker which may occur within the Huffman coded data is the RST_m marker. Other than the EOI or markers which may occur at or before the start of a scan, the only marker which can occur at the end of the scan is the DNL (define-number-of-lines).

Normally, the decoder will terminate the decoding at the end of the final restart interval before the terminating marker is intercepted. If the DNL marker is encountered, the current line count is set to the value specified by that marker. Since the DNL marker can only be used at the end of the first scan, the scan decode procedure must be terminated when it is encountered.

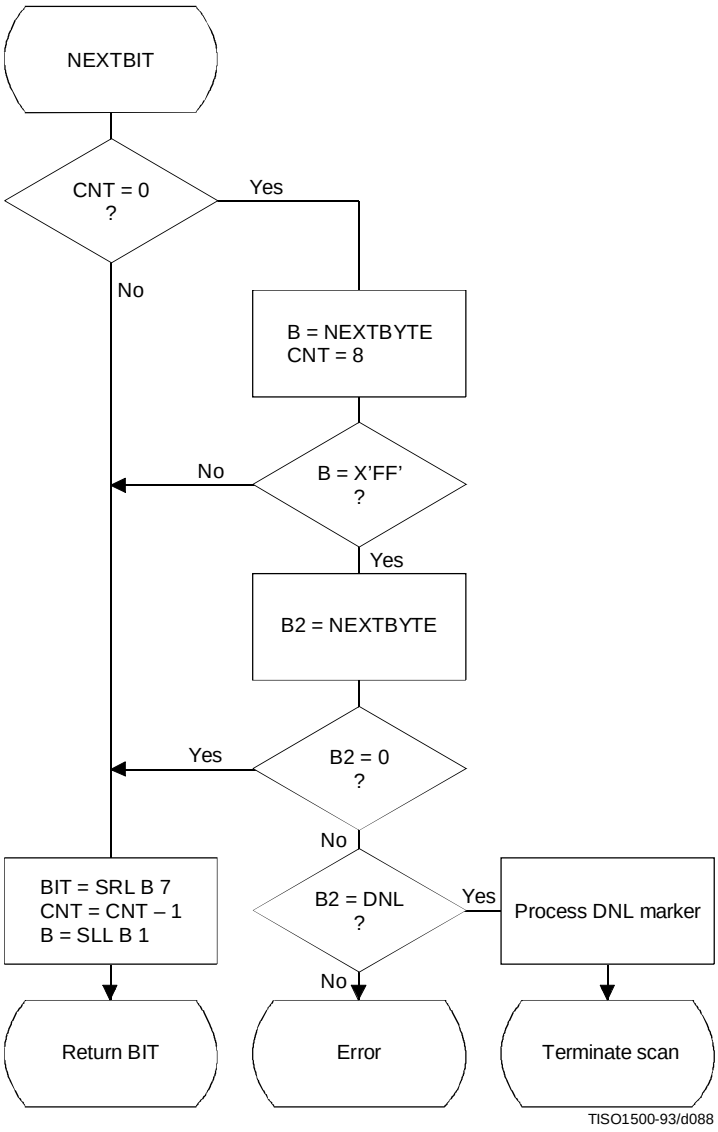


Figure F.18 – Procedure for fetching the next bit of compressed data

F.2.3 Sequential DCT decoding process with 8-bit precision extended to four sets of Huffman tables

This process is identical to the Baseline decoding process described in F.2.2, with the exception that the decoder shall be capable of using up to four DC and four AC Huffman tables within one scan. Four DC and four AC Huffman tables is the maximum allowed by this Specification.

F.2.4 Sequential DCT decoding process with arithmetic coding

This subclause describes the sequential DCT decoding process with arithmetic decoding.

The arithmetic decoding procedures for decoding binary decisions, initializing the statistical model, initializing the decoder, and resynchronizing the decoder are listed in Table D.4 of Annex D.

Some of the procedures in Table D.4 are used in the higher level control structure for scans and restart intervals described in F.2. At the beginning of scans and restart intervals, the probability estimates used in the arithmetic decoder are reset to the standard initial value as part of the Initdec procedure which restarts the arithmetic coder.

The statistical models defined in F.1.4.4 also apply to this decoding process.

The decoder shall be capable of using up to four DC and four AC conditioning tables and associated statistics areas within one scan.

F.2.4.1 Arithmetic decoding of DC coefficients

The basic structure of the decision sequence for decoding a DC difference value, DIFF, is shown in Figure F.19. The equivalent structure for the encoder is found in Figure F.4.

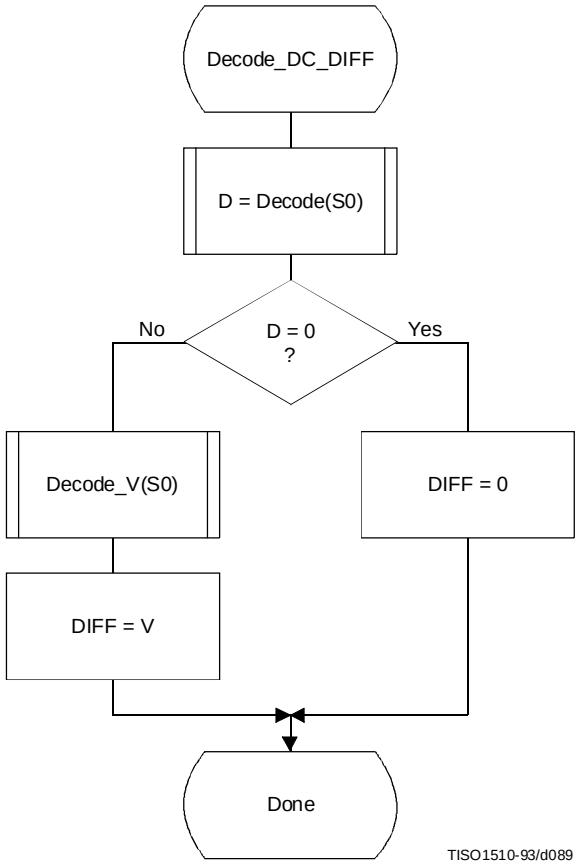


Figure F.19 – Arithmetic decoding of DC difference

The context-indices used in the DC decoding procedures are defined in Table F.4 (see F.1.4.4.1.3).

The “Decode” procedure returns the value “D” of the binary decision. If the value is not zero, the sign and magnitude of the non-zero DIFF must be decoded by the procedure “Decode_V(S0)”.

F.2.4.2 Arithmetic Decoding of AC coefficients

The AC coefficients are decoded in the order that they occur in ZZ(1,...,63). The encoder procedure for the coding process is found in Figure F.5. Figure F.20 illustrates the decoding sequence.

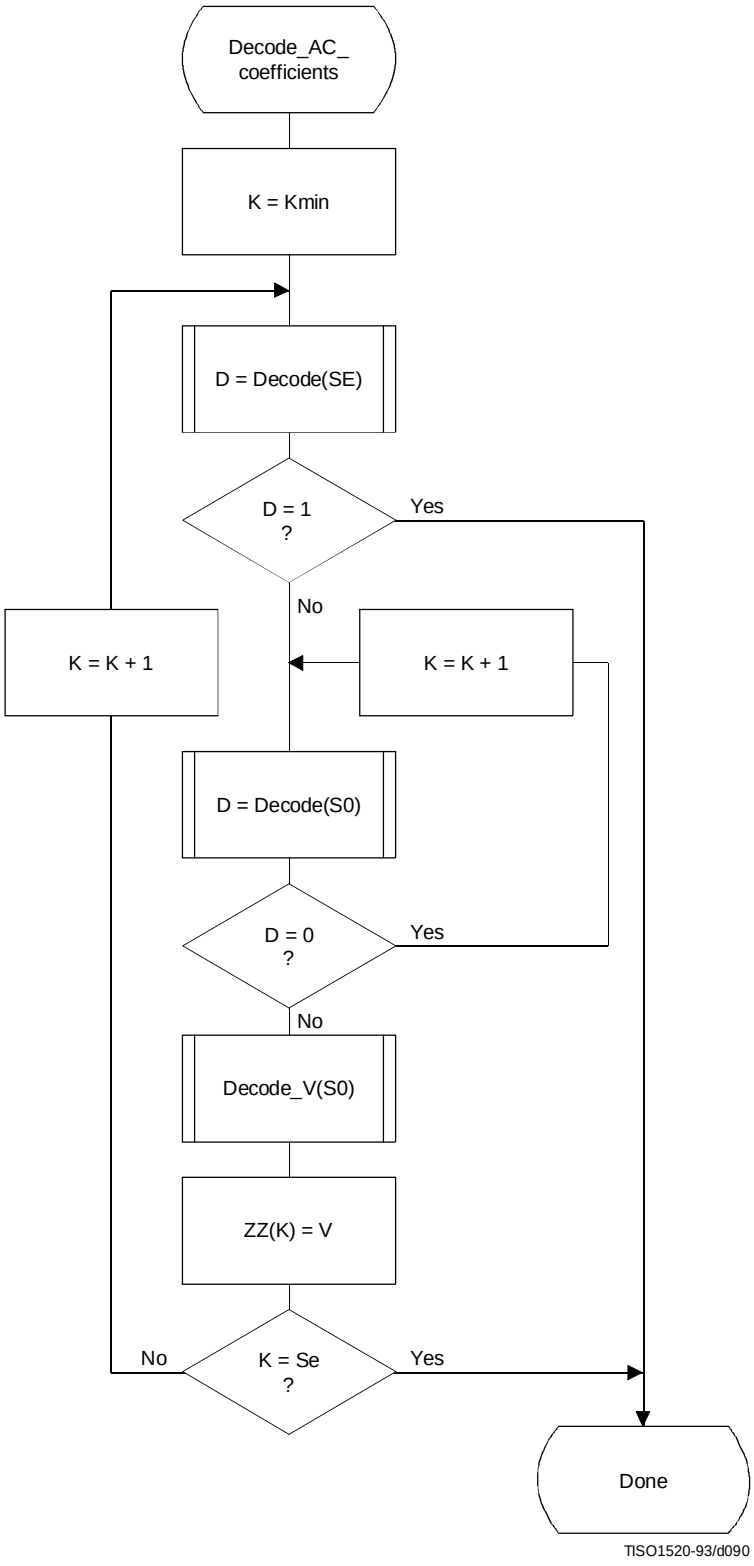


Figure F.20 – Procedure for decoding the AC coefficients

The context-indices used in the AC decoding procedures are defined in Table F.5 (see F.1.4.4.2).

In Figure F.20, K is the index to the zig-zag sequence position. For the sequential scan, Kmin = 1 and Se = 63. The decision at the top of the loop is the EOB decision. If the EOB occurs (D = 1), the remaining coefficients in the block are set to zero. The inner loop just below the EOB decoding decodes runs of zero coefficients. Whenever the coefficient is non-zero, "Decode_V" decodes the sign and magnitude of the coefficient. After each non-zero coefficient is decoded, the EOB decision is again decoded unless K = Se.

F.2.4.3 Decoding the binary decision sequence for non-zero DC differences and AC coefficients

Both the DC difference and the AC coefficients are represented as signed two's complement 16-bit integer values. The decoding decision tree for these signed integer values is the same for both the DC and AC coding models. Note, however, that the statistical models are not the same.

F.2.4.3.1 Arithmetic decoding of non-zero values

Denoting either DC differences or AC coefficients as V, the non-zero signed integer value of V is decoded by the sequence shown in Figure F.21. This sequence first decodes the sign of V. It then decodes the magnitude category of V (Decode_log2_Sz), and then decodes the low order magnitude bits (Decode_Sz_bits). Note that the value decoded for Sz must be incremented by 1 to get the actual coefficient magnitude.

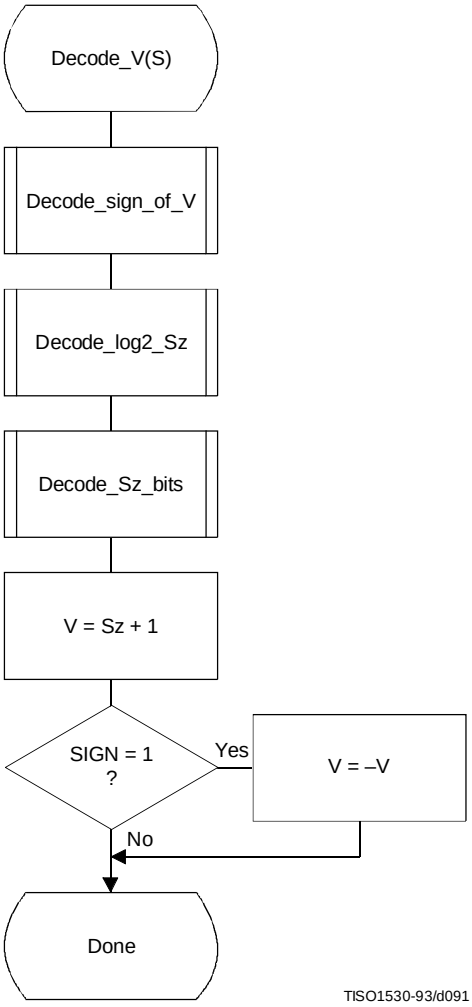


Figure F.21 – Sequence of procedures in decoding non-zero values of V

F.2.4.3.1.1 Decoding the sign

The sign is decoded by the procedure shown in Figure F.22.

The context-indices are defined for DC decoding in Table F.4 and AC decoding in Table F.5.

If SIGN = 0, the sign of the coefficient is positive; if SIGN = 1, the sign of the coefficient is negative.

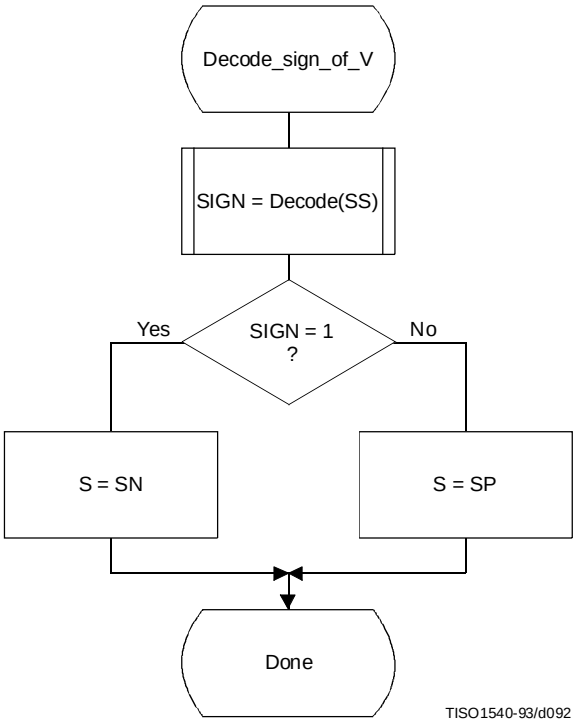


Figure F.22 – Decoding the sign of V

F.2.4.3.1.2 Decoding the magnitude category

The context-index S is set in Decode_sign_of_V and the context-index values X1 and X2 are defined for DC coding in Table F.4 and for AC coding in Table F.5.

In Figure F.23, M is set to the upper bound for the magnitude and shifted left until the decoded decision is zero. It is then shifted right by 1 to become the leading bit of the magnitude of Sz.

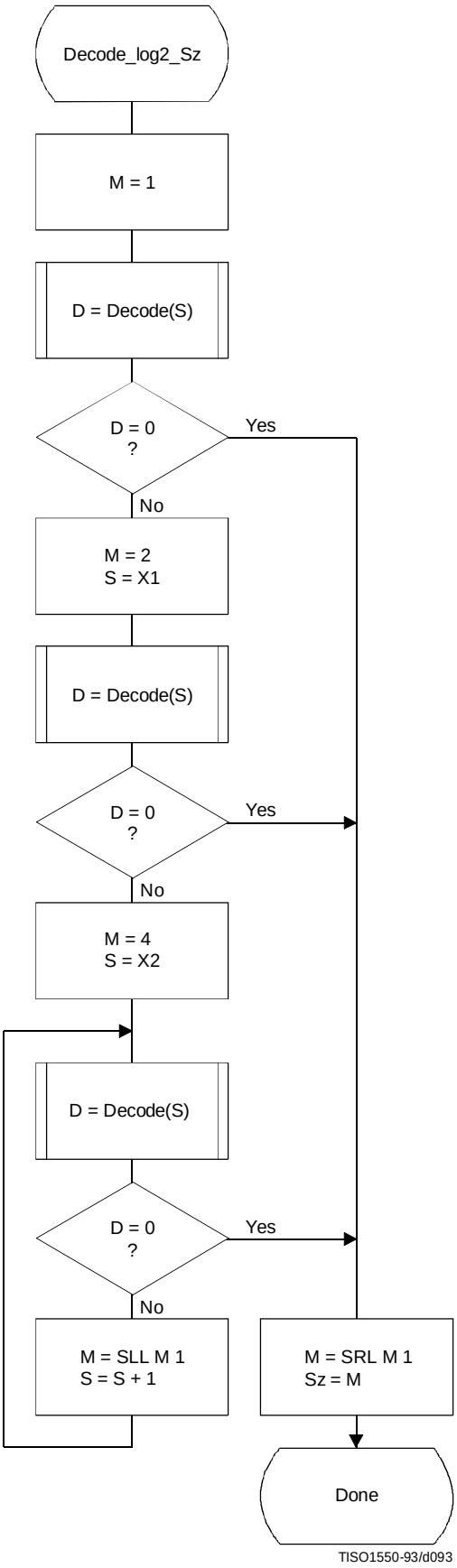


Figure F.23 – Decoding procedure to establish the magnitude category

F.2.4.3.1.3 Decoding the exact value of the magnitude

After the magnitude category is decoded, the low order magnitude bits are decoded. These bits are decoded in order of decreasing bit significance. The procedure is shown in Figure F.24.

The context-index S is set in Decode_log2_Sz.

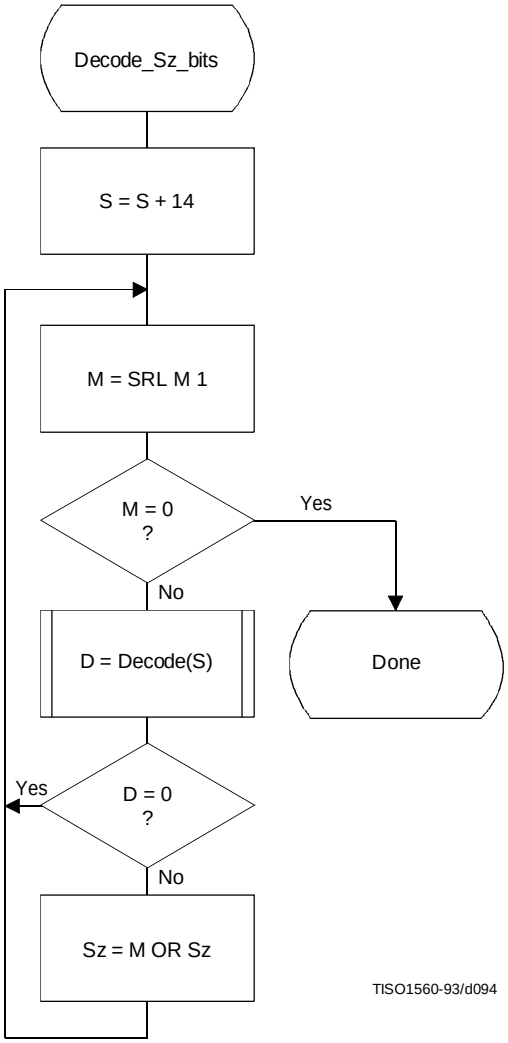


Figure F.24 – Decision sequence to decode the magnitude bit pattern

F.2.4.4 Decoder restart

The RST_m markers which are added to the compressed data between each restart interval have a two byte value which cannot be generated by the coding procedures. These two byte sequences can be located without decoding, and can therefore be used to resynchronize the decoder. RST_m markers can therefore be used for error recovery.

Before error recovery procedures can be invoked, the error condition must first be detected. Errors during decoding can show up in two places:

a) The decoder fails to find the expected marker at the point where it is expecting resynchronization.

b) Physically impossible data are decoded. For example, decoding a magnitude beyond the range of values allowed by the model is quite likely when the compressed data are corrupted by errors. For arithmetic decoders this error condition is extremely important to detect, as otherwise the decoder may reach a condition where it uses the compressed data very slowly.

NOTE – Some errors will not cause the decoder to lose synchronization. In addition, recovery is not possible for all errors; for example, errors in the headers are likely to be catastrophic. The two error conditions listed above, however, almost always cause the decoder to lose synchronization in a way which permits recovery.

In regaining synchronization, the decoder can make use of the modulo 8 coding restart interval number in the low order bits of the RST_m marker. By comparing the expected restart interval number to the value in the next RST_m marker in the compressed image data, the decoder can usually recover synchronization. It then fills in missing lines in the output data by replication or some other suitable procedure, and continues decoding. Of course, the reconstructed image will usually be highly corrupted for at least a part of the restart interval where the error occurred.

F.2.5 Sequential DCT decoding process with Huffman coding and 12-bit precision

This process is identical to the sequential DCT process defined for 8-bit sample precision and extended to four Huffman tables, as documented in F.2.3, but with the following changes.

F.2.5.1 Structure of DC Huffman decode table

The general structure of the DC Huffman decode table is extended as described in F.1.5.1.

F.2.5.2 Structure of AC Huffman decode table

The general structure of the AC Huffman decode table is extended as described in F.1.5.2.

F.2.6 Sequential DCT decoding process with arithmetic coding and 12-bit precision

The process is identical to the sequential DCT process for 8-bit precision except for changes in the precision of the IDCT computation.

The structure of the decoding procedure in F.2.4 is already defined for a 12-bit input precision.

Annex G

Progressive DCT-based mode of operation

(This annex forms an integral part of this Recommendation | International Standard)

This annex provides a **functional specification** of the following coding processes for the progressive DCT-based mode of operation:

1) spectral selection only, Huffman coding, 8-bit sample precision;
2) spectral selection only, arithmetic coding, 8-bit sample precision;
3) full progression, Huffman coding, 8-bit sample precision;
4) full progression, arithmetic coding, 8-bit sample precision;
5) spectral selection only, Huffman coding, 12-bit sample precision;
6) spectral selection only, arithmetic coding, 12-bit sample precision;
7) full progression, Huffman coding, 12-bit sample precision;
8) full progression, arithmetic coding, 12-bit sample precision.

For each of these, the encoding process is specified in G.1, and the decoding process is specified in G.2. The functional specification is presented by means of specific flow charts for the various procedures which comprise these coding processes.

NOTE – There is **no requirement** in this Specification that any encoder or decoder which embodies one of the above-named processes shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

The number of Huffman or arithmetic conditioning tables which may be used within the same scan is four.

Two complementary progressive procedures are defined, spectral selection and successive approximation.

In spectral selection the DCT coefficients of each block are segmented into frequency bands. The bands are coded in separate scans.

In successive approximation the DCT coefficients are divided by a power of two before coding. In the decoder the coefficients are multiplied by that same power of two before computing the IDCT. In the succeeding scans the precision of the coefficients is increased by one bit in each scan until full precision is reached.

An encoder or decoder implementing a full progression uses spectral selection within successive approximation. An allowed subset is spectral selection alone.

Figure G.1 illustrates the spectral selection and successive approximation progressive processes.

G.1 Progressive DCT-based encoding processes

G.1.1 Control procedures and coding models for progressive DCT-based procedures

G.1.1.1 Control procedures for progressive DCT-based encoders

The control procedures for encoding an image and its constituent parts – the frame, scan, restart interval and MCU – are given in Figures E.1 through E.5.

The control structure for encoding a frame is the same as for the sequential procedures. However, it is convenient to calculate the FDCT for the entire set of components in a frame before starting the scans. A buffer which is large enough to store all of the DCT coefficients may be used for this progressive mode of operation.

The number of scans is determined by the progression defined; the number of scans may be much larger than the number of components in the frame.

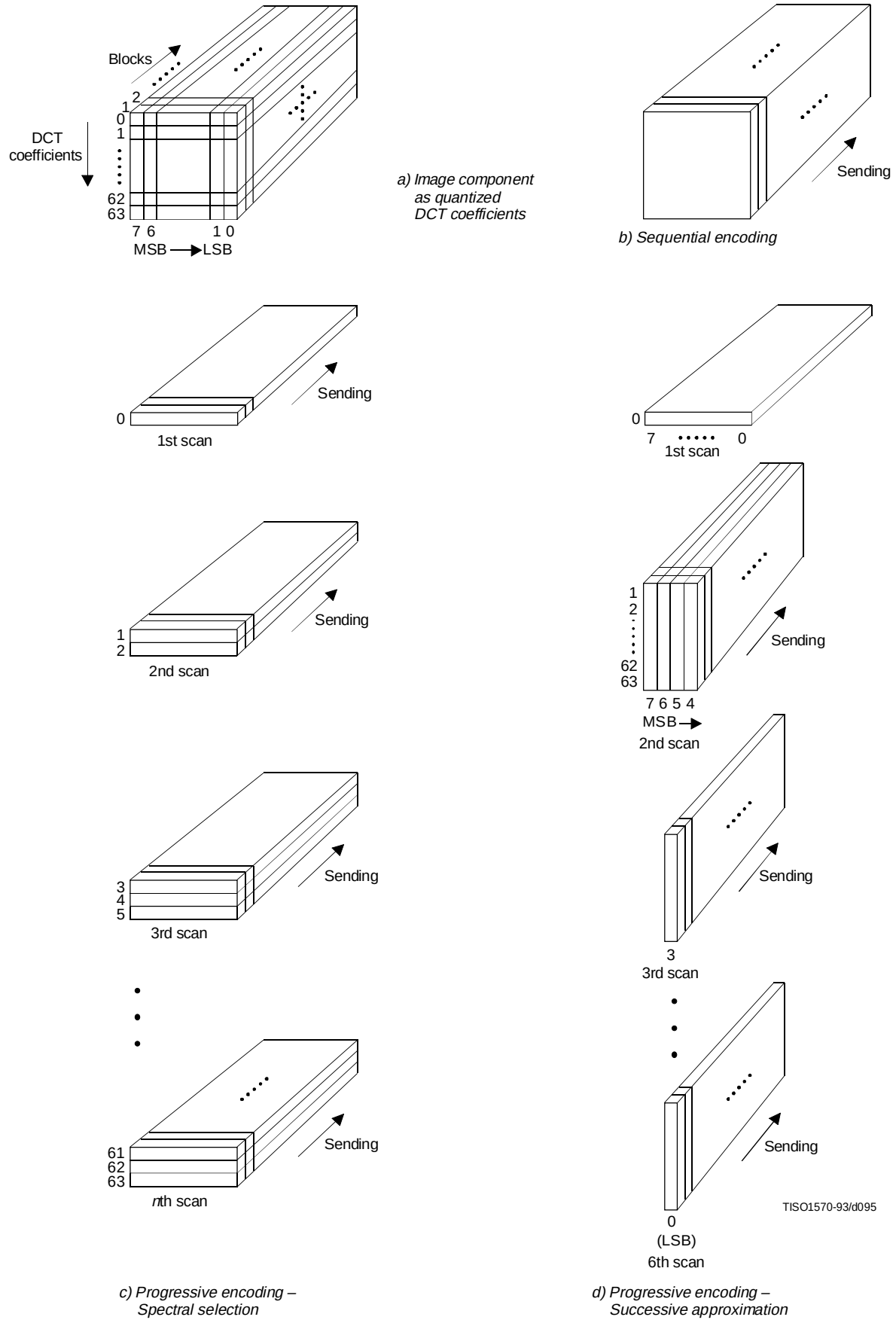


Figure G.1 – Spectral selection and successive approximation progressive processes

The procedure for encoding a MCU (see Figure E.5) repetitively invokes the procedure for coding a data unit. For DCT-based encoders the data unit is an 8×8 block of samples.

Only a portion of each 8×8 block is coded in each scan, the portion being determined by the scan header parameters Ss, Se, Ah, and Al (see B.2.3). The procedures used to code portions of each 8×8 block are described in this annex. Note, however, that where these procedures are identical to those used in the sequential DCT-based mode of operation, the sequential procedures are simply referenced.

G.1.1.1.1 Spectral selection control

In spectral selection the zig-zag sequence of DCT coefficients is segmented into bands. A band is defined in the scan header by specifying the starting and ending indices in the zig-zag sequence. One band is coded in a given scan of the progression. DC coefficients are always coded separately from AC coefficients, and only scans which code DC coefficients may have interleaved blocks from more than one component. All other scans shall have only one component. With the exception of the first DC scans for the components, the sequence of bands defined in the scans need not follow the zig-zag ordering. For each component, a first DC scan shall precede any AC scans.

G.1.1.1.2 Successive approximation control

If successive approximation is used, the DCT coefficients are reduced in precision by the point transform (see A.4) defined in the scan header (see B.2.3). The successive approximation bit position parameter Al specifies the actual point transform, and the high four bits (Ah) – if there are preceding scans for the band – contain the value of the point transform used in those preceding scans. If there are no preceding scans for the band, Ah is zero.

Each scan which follows the first scan for a given band progressively improves the precision of the coefficients by one bit, until full precision is reached.

G.1.1.2 Coding models for progressive DCT-based encoders

If successive approximation is used, the DCT coefficients are reduced in precision by the point transform (see A.4) defined in the scan header (see B.2.3). These models also apply to the progressive DCT-based encoders, but with the following changes.

G.1.1.2.1 Progressive encoding model for DC coefficients

If Al is not zero, the point transform for DC coefficients shall be used to reduce the precision of the DC coefficients. If Ah is zero, the coefficient values (as modified by the point transform) shall be coded, using the procedure described in Annex F. If Ah is not zero, the least significant bit of the point transformed DC coefficients shall be coded, using the procedures described in this annex.

G.1.1.2.2 Progressive encoding model for AC coefficients

If Al is not zero, the point transform for AC coefficients shall be used to reduce the precision of the AC coefficients. If Ah is zero, the coefficient values (as modified by the point transform) shall be coded using modifications of the procedures described in Annex F. These modifications are described in this annex. If Ah is not zero, the precision of the coefficients shall be improved using the procedures described in this annex.

G.1.2 Progressive encoding procedures with Huffman coding

G.1.2.1 Progressive encoding of DC coefficients with Huffman coding

The first scan for a given component shall encode the DC coefficient values using the procedures described in F.1.2.1. If the successive approximation bit position parameter Al is not zero, the coefficient values shall be reduced in precision by the point transform described in Annex A before coding.

In subsequent scans using successive approximation the least significant bits are appended to the compressed bit stream without compression or modification (see G.1.2.3), except for byte stuffing.

G.1.2.2 Progressive encoding of AC coefficients with Huffman coding

In spectral selection and in the first scan of successive approximation for a component, the AC coefficient coding model is similar to that used by the sequential procedures. However, the Huffman code tables are extended to include coding of runs of End-Of-Bands (EOBs). See Table G.1.

Table G.1 – EOBn code run length extensions

EOBn code	Run length
EOB0	1
EOB1	2,3
EOB2	4..7
EOB3	8..15
EOB4	16..31
EOB5	32..63
EOB6	64..127
EOB7	128..255
EOB8	256..511
EOB9	512..1 023
EOB10	1 024..2 047
EOB11	2 048..4 095
EOB12	4 096..8 191
EOB13	8 192..16 383
EOB14	16 384..32 767

The end-of-band run structure allows efficient coding of blocks which have only zero coefficients. An EOB run of length 5 means that the current block and the next four blocks have an end-of-band with no intervening non-zero coefficients. The EOB run length is limited only by the restart interval.

The extension of the code table is illustrated in Figure G.2.

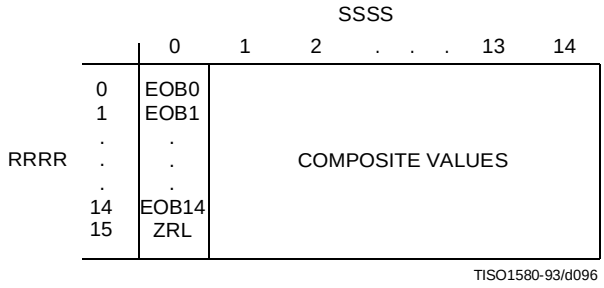


Figure G.2 – Two-dimensional value array for Huffman coding

The EOBn code sequence is defined as follows. Each EOBn code is followed by an extension field similar to the extension field for the coefficient amplitudes (but with positive numbers only). The number of bits appended to the EOBn code is the minimum number required to specify the run length.

If an EOB run is greater than 32 767, it is coded as a sequence of EOB runs of length 32 767 followed by a final EOB run sufficient to complete the run.

At the beginning of each restart interval the EOB run count, EOBRUN, is set to zero. At the end of each restart interval any remaining EOB run is coded.

The Huffman encoding procedure for AC coefficients in spectral selection and in the first scan of successive approximation is illustrated in Figures G.3, G.4, G.5, and G.6.

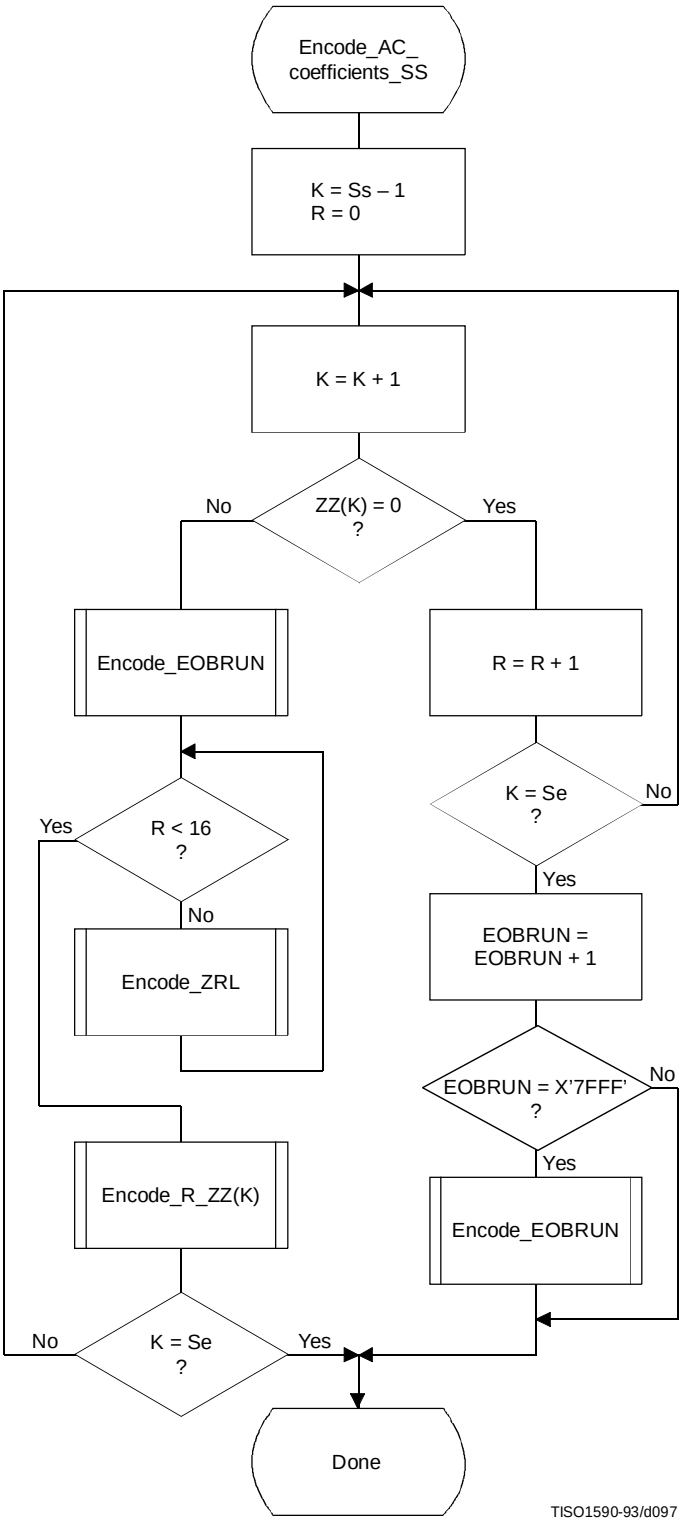


Figure G.3 – Procedure for progressive encoding of AC coefficients with Huffman coding

In Figure G.3, Ss is the start of spectral selection, Se is the end of spectral selection, K is the index into the list of coefficients stored in the zig-zag sequence ZZ, R is the run length of zero coefficients, and EOBRUN is the run length of EOBs. EOBRUN is set to zero at the start of each restart interval.

If the scan header parameter Al (successive approximation bit position low) is not zero, the DCT coefficient values ZZ(K) in Figure G.3 and figures which follow in this annex, including those in the arithmetic coding section, shall be replaced by the point transformed values ZZ'(K), where ZZ'(K) is defined by:

$$ZZ'(K) = \frac{ZZ(K)x}{2^{Al}}$$

EOBSIZE is a procedure which returns the size of the EOB extension field given the EOB run length as input. CSIZE is a procedure which maps an AC coefficient to the SSSS value defined in the subclauses on sequential encoding (see F.1.1 and F.1.3).

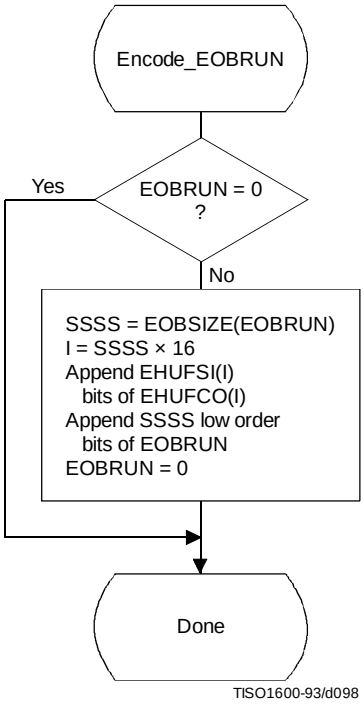


Figure G.4 – Progressive encoding of a non-zero AC coefficient

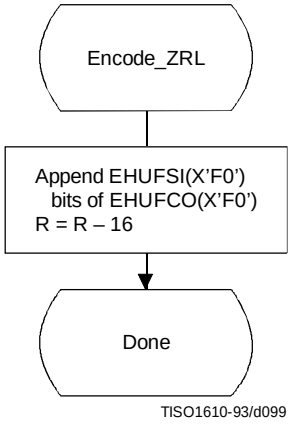


Figure G.5 – Encoding of the run of zero coefficients

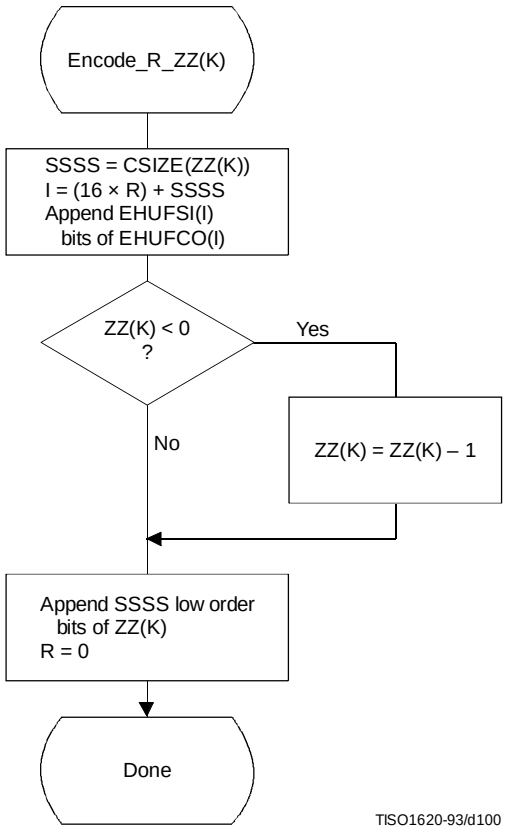


Figure G.6 – Encoding of the zero run and non-zero coefficient

G.1.2.3 Coding model for subsequent scans of successive approximation

The Huffman coding structure of the subsequent scans of successive approximation for a given component is similar to the coding structure of the first scan of that component.

The structure of the AC code table is identical to the structure described in G.1.2.2. Each non-zero point transformed coefficient that has a zero history (i.e. that has a value ± 1, and therefore has not been coded in a previous scan) is defined by a composite 8-bit run length-magnitude value of the form:

RRRRSSSS

The four most significant bits, RRRR, give the number of zero coefficients that are between the current coefficient and the previously coded coefficient (or the start of band). Coefficients with non-zero history (a non-zero value coded in a previous scan) are skipped over when counting the zero coefficients. The four least significant bits, SSSS, provide the magnitude category of the non-zero coefficient; for a given component the value of SSSS can only be one.

The run length-magnitude composite value is Huffman coded and each Huffman code is followed by additional bits:

a) One bit codes the sign of the newly non-zero coefficient. A 0-bit codes a negative sign; a 1-bit codes a positive sign.

b) For each coefficient with a non-zero history, one bit is used to code the correction. A 0-bit means no correction and a 1-bit means that one shall be added to the (scaled) decoded magnitude of the coefficient.

Non-zero coefficients with zero history are coded with a composite code of the form:

HUFFCO(RRRRSSSS) + additional bit (rule a) + correction bits (rule b)

In addition whenever zero runs are coded with ZRL or EOBn codes, correction bits for those coefficients with non-zero history contained within the zero run are appended according to rule b above.

For the Huffman coding version of Encode_AC_Coefficients_SA the EOB is defined to be the position of the last point transformed coefficient of magnitude 1 in the band. If there are no coefficients of magnitude 1, the EOB is defined to be zero.

NOTE – The definition of EOB is different for Huffman and arithmetic coding procedures.

In Figures G.7 and G.8 BE is the count of buffered correction bits at the start of coding of the block. BE is initialized to zero at the start of each restart interval. At the end of each restart interval any remaining buffered bits are appended to the bit stream following the last EOBn Huffman code and associated appended bits.

In Figures G.7 and G.9, BR is the count of buffered correction bits which are appended to the bit stream according to rule b. BR is set to zero at the beginning of each Encode_AC_Coefficients_SA. At the end of each restart interval any remaining buffered bits are appended to the bit stream following the last Huffman code and associated appended bits.

G.1.3 Progressive encoding procedures with arithmetic coding

G.1.3.1 Progressive encoding of DC coefficients with arithmetic coding

The first scan for a given component shall encode the DC coefficient values using the procedures described in F.1.4.1. If the successive approximation bit position parameter is not zero, the coefficient values shall be reduced in precision by the point transform described in Annex A before coding.

In subsequent scans using successive approximation the least significant bits shall be coded as binary decisions using a fixed probability estimate of 0.5 (Qe = X'5A1D', MPS = 0).

G.1.3.2 Progressive encoding of AC coefficients with arithmetic coding

Except for the point transform scaling of the DCT coefficients and the grouping of the coefficients into bands, the first scan(s) of successive approximation is identical to the sequential encoding procedure described in F.1.4. If Kmin is equated to Ss, the index of the first AC coefficient index in the band, the flow chart shown in Figure F.5 applies. The EOB decision in that figure refers to the "end-of-band" rather than the "end-of-block". For the arithmetic coding version of Encode_AC_Coefficients_SA (and all other AC coefficient coding procedures) the EOB is defined to be the position following the last non-zero coefficient in the band.

NOTE - The definition of EOB is different for Huffman and arithmetic coding procedures.

The statistical model described in F.1.4 also holds. For this model the default value of Kx is 5. Other values of Kx may be specified using the DAC marker code (Annex B). The following calculation for Kx has proven to give good results for 8-bit precision samples:

$$Kx = Kmin + SRL\ (8 + Se - Kmin)\ 4$$

This expression reduces to the default of Kx = 5 when the band is from index 1 to index 63.

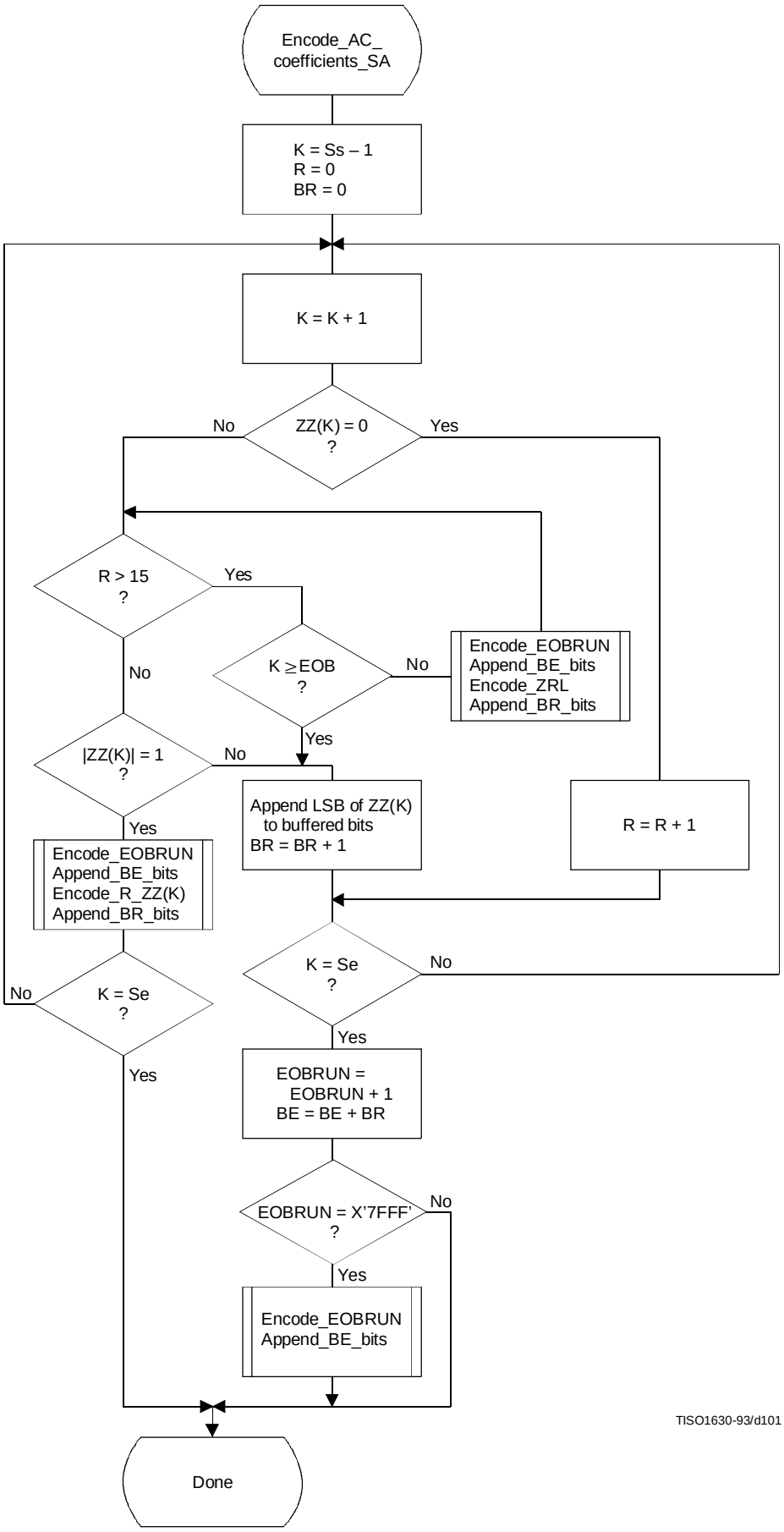


Figure G.7 – Successive approximation coding of AC coefficients using Huffman coding

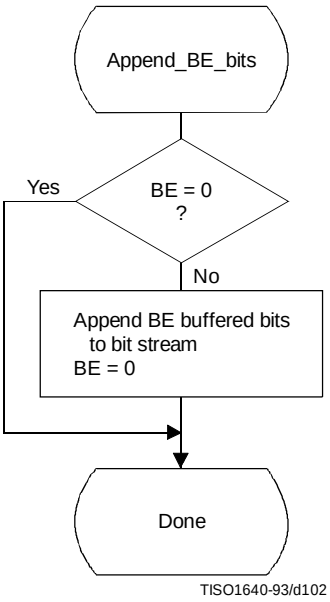


Figure G.8 – Transferring BE buffered bits from buffer to bit stream

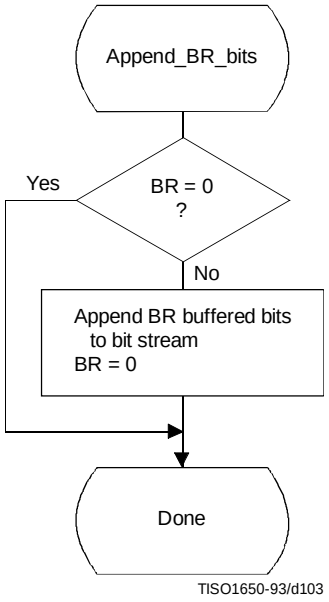


Figure G.9 – Transferring BR buffered bits from buffer to bit stream

G.1.3.3 Coding model for subsequent scans of successive approximation

The procedure “Encode_AC_Coefficient_SA” shown in Figure G.10 increases the precision of the AC coefficient values in the band by one bit.

As in the first scan of successive approximation for a component, an EOB decision is coded at the start of the band and after each non-zero coefficient.

However, since the end-of-band index of the previous successive approximation scan for a given component, EOBx, is known from the data coded in the prior scan of that component, this decision is bypassed whenever the current index, K, is less than EOBx. As in the first scan(s), the EOB decision is also bypassed whenever the last coefficient in the band is not zero. The decision ZZ(K) = 0 decodes runs of zero coefficients. If the decoder is at this step of the procedure, at least one non-zero coefficient remains in the band of the block being coded. If ZZ(K) is not zero, the procedure in Figure G.11 is followed to code the value.

The context-indices in Figures G.10 and G.11 are defined in Table G.2 (see G.1.3.3.1). The signs of coefficients with magnitude of one are coded with a fixed probability value of approximately 0.5 (Qe = X’5A1D’, MPS = 0).

G.1.3.3.1 Statistical model for subsequent successive approximation scans

As shown in Table G.2, each statistics area for subsequent successive approximation scans of AC coefficients consists of a contiguous set of 189 statistics bins. The signs of coefficients with magnitude of one are coded with a fixed probability value of approximately 0.5 (Qe = X’5A1D’, MPS = 0).

G.2 Progressive decoding of the DCT

The description of the computation of the IDCT and the dequantization procedure contained in A.3.3 and A.3.4 apply to the progressive operation.

Progressive decoding processes must be able to decompress compressed image data which requires up to four sets of Huffman or arithmetic coder conditioning tables within a scan.

In order to avoid repetition, detailed flow diagrams of progressive decoder operation are not included. Decoder operation is defined by reversing the function of each step described in the encoder flow charts, and performing the steps in reverse order.

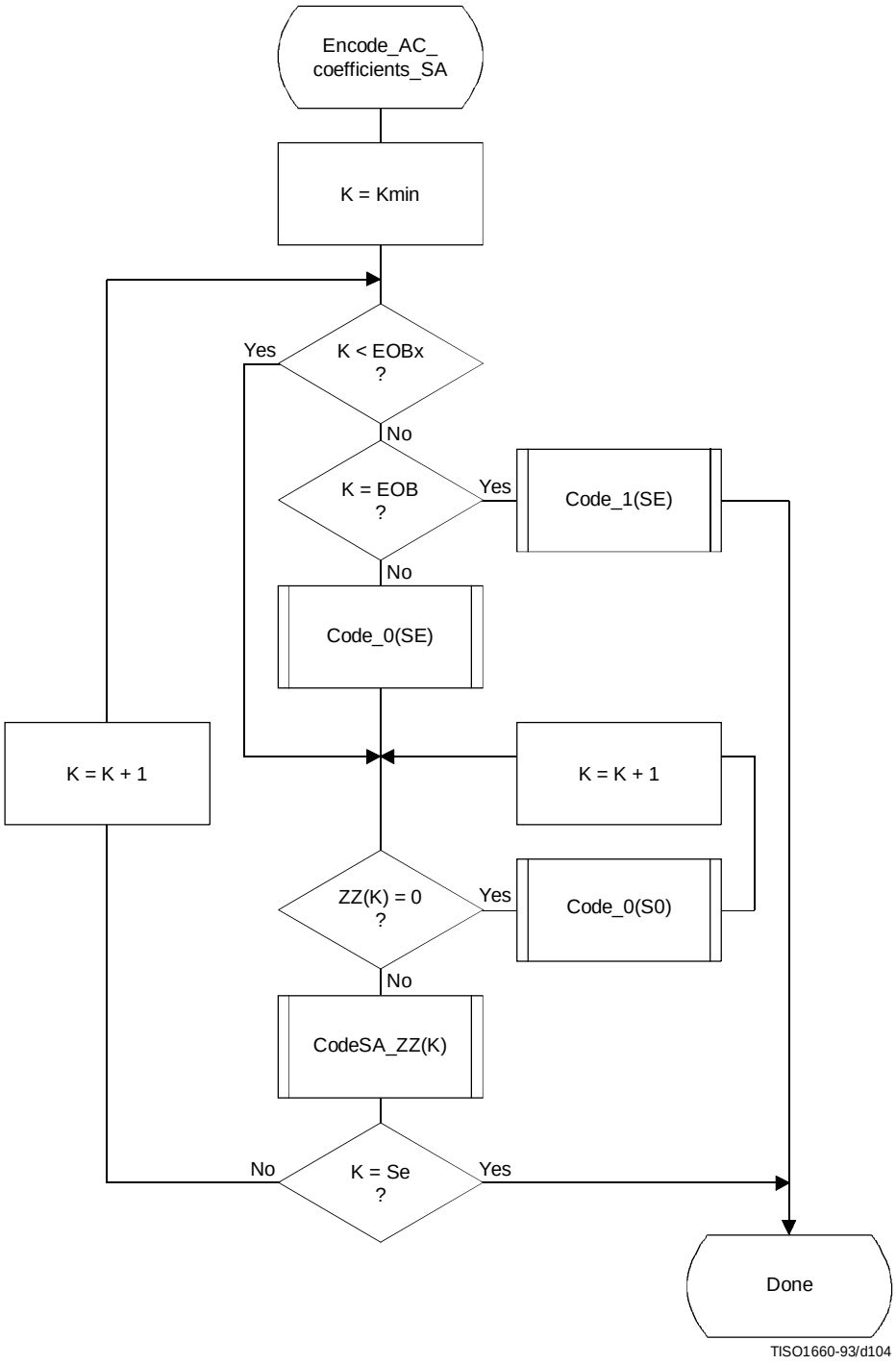


Figure G.10 – Subsequent successive approximation scans for coding of AC coefficients using arithmetic coding

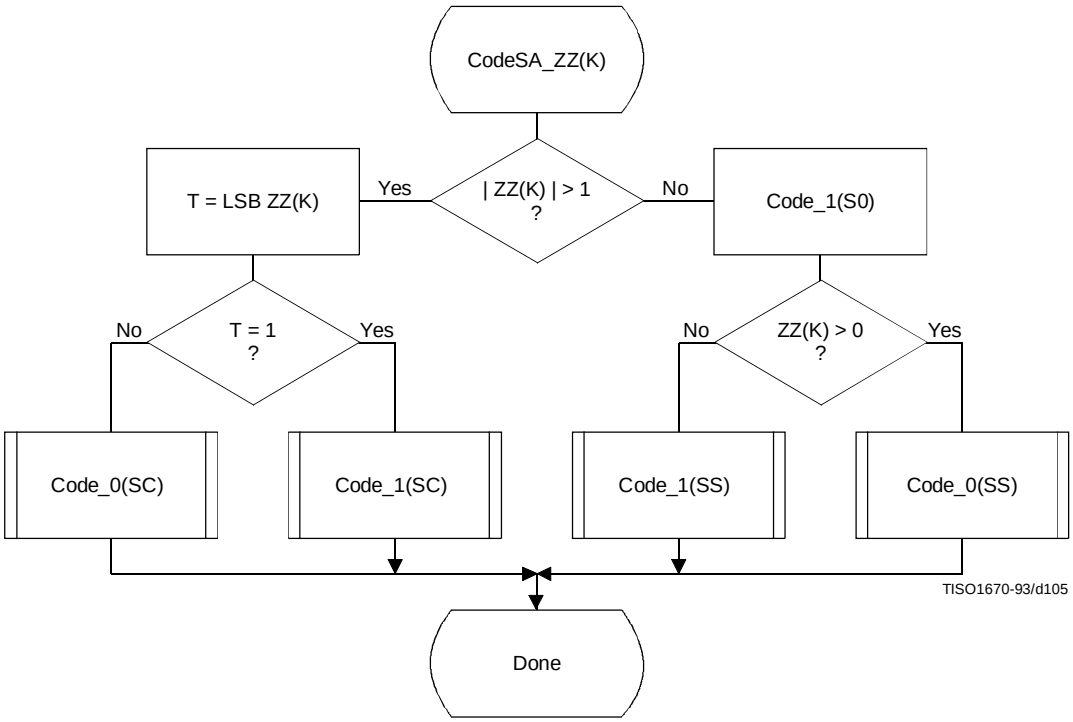


Figure G.11 – Coding non-zero coefficients for subsequent successive approximation scans

Table G.2 – Statistical model for subsequent scans of successive approximation coding of AC coefficient

Context-index	AC coding	Coding decision
SE	3 × (K–1)	K = EOB
S0	SE + 1	V = 0
SS	Fixed estimate	Sign
SC	S0 + 1	LSB ZZ(K) = 1

Annex H

Lossless mode of operation

(This annex forms an integral part of this Recommendation | International Standard)

This annex provides a **functional specification** of the following coding processes for the lossless mode of operation:

1) lossless processes with Huffman coding;

2) lossless processes with arithmetic coding.

For each of these, the encoding process is specified in H.1, and the decoding process is specified in H.2. The functional specification is presented by means of specific procedures which comprise these coding processes.

NOTE – There is **no requirement** in this Specification that any encoder or decoder which embodies one of the above-named processes shall implement the procedures in precisely the manner specified in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

The processes which provide for sequential lossless encoding and decoding are not based on the DCT. The processes used are spatial processes based on the coding model developed for the DC coefficients of the DCT. However, the model is extended by incorporating a set of selectable one- and two-dimensional predictors, and for interleaved data the ordering of samples for the one-dimensional predictor can be different from that used in the DCT-based processes.

Either Huffman coding or arithmetic coding entropy coding may be employed for these lossless encoding and decoding processes. The Huffman code table structure is extended to allow up to 16-bit precision for the input data. The arithmetic coder statistical model is extended to a two-dimensional form.

H.1 Lossless encoder processes

H.1.1 Lossless encoder control procedures

Subclause E.1 contains the encoder control procedures. In applying these procedures to the lossless encoder, the data unit is one sample.

Input data precision may be from 2 to 16 bits/sample. If the input data path has different precision from the input data, the data shall be aligned with the least significant bits of the input data path. Input data is represented as unsigned integers and is not level shifted prior to coding.

When the encoder is reset in the restart interval control procedure (see E.1.4), the prediction is reset to a default value. If arithmetic coding is used, the statistics are also reset.

For the lossless processes the restart interval shall be an integer multiple of the number of MCU in an MCU-row.

H.1.2 Coding model for lossless encoding

The coding model developed for encoding the DC coefficients of the DCT is extended to allow a selection from a set of seven one-dimensional and two-dimensional predictors. The predictor is selected in the scan header (see Annex B). The same predictor is used for all components of the scan. Each component in the scan is modeled independently, using predictions derived from neighbouring samples of that component.

H.1.2.1 Prediction

Figure H.1 shows the relationship between the positions (a, b, c) of the reconstructed neighboring samples used for prediction and the position of x, the sample being coded.

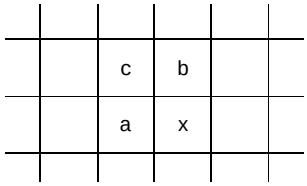


Figure H.1 – Relationship between sample and prediction samples

Define Px to be the prediction and Ra, Rb, and Rc to be the reconstructed samples immediately to the left, immediately above, and diagonally to the left of the current sample. The allowed predictors, one of which is selected in the scan header, are listed in Table H.1.

Table H.1 – Predictors for lossless coding

Selection-value	Prediction
0	No prediction (See Annex J)
1	Px = Ra
2	Px = Rb
3	Px = Rc
4	Px = Ra + Rb – Rc
5	Px = Ra + ((Rb – Rc)/2)[a)]
6	Px = Rb + ((Ra – Rc)/2)[a)]
7	Px = (Ra + Rb)/2

[a)] Shift right arithmetic operation

Selection-value 0 shall only be used for differential coding in the hierarchical mode of operation. Selections 1, 2 and 3 are one-dimensional predictors and selections 4, 5, 6, and 7 are two-dimensional predictors.

The one-dimensional horizontal predictor (prediction sample Ra) is used for the first line of samples at the start of the scan and at the beginning of each restart interval. The selected predictor is used for all other lines. The sample from the line above (prediction sample Rb) is used at the start of each line, except for the first line. At the beginning of the first line and at the beginning of each restart interval the prediction value of 2^{P-1} is used, where P is the input precision.

If the point transformation parameter (see A.4) is non-zero, the prediction value at the beginning of the first lines and the beginning of each restart interval is 2^{P-Pt-1}, where Pt is the value of the point transformation parameter.

Each prediction is calculated with full integer arithmetic precision, and without clamping of either underflow or overflow beyond the input precision bounds. For example, if Ra and Rb are both 16-bit integers, the sum is a 17-bit integer. After dividing the sum by 2 (predictor 7), the prediction is a 16-bit integer.

For simplicity of implementation, the divide by 2 in the prediction selections 5 and 6 of Table H.1 is done by an arithmetic-right-shift of the integer values.

The difference between the prediction value and the input is calculated modulo 2^{16}. In the decoder the difference is decoded and added, modulo 2^{16}, to the prediction.

H.1.2.2 Huffman coding of the modulo difference

The Huffman coding procedures defined in Annex F for coding the DC coefficients are used to code the modulo 2^{16} differences. The table for DC coding contained in Tables F.1 and F.6 is extended by one additional entry. No extra bits are appended after SSSS = 16 is encoded. See Table H.2.

Table H.2 – Difference categories for lossless Huffman coding

SSSS	Difference values
0	0
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023
11	–2 047..–1 024,1 024..2 047
12	–4 095..–2 048,2 048..4 095
13	–8 191..–4 096,4 096..8 191
14	–16 383..–8 192,8 192..16 383
15	–32 767..–16 384,16 384..32 767
16	32 768

H.1.2.3 Arithmetic coding of the modulo difference

The statistical model defined for the DC coefficient arithmetic coding model (see F.1.4.4.1) is generalized to a two-dimensional form in which differences coded for the sample to the left and for the line above are used for conditioning.

H.1.2.3.1 Two-dimensional statistical model

The binary decisions are conditioned on the differences coded for the neighbouring samples immediately above and immediately to the left from the same component. As in the coding of the DC coefficients, the differences are classified into 5 categories: zero(0), small positive (+S), small negative (–S), large positive (+L), and large negative (–L). The two independent difference categories combine to give 25 different conditioning states. Figure H.2 shows the two-dimensional array of conditioning indices. For each of the 25 conditioning states probability estimates for four binary decisions are kept.

At the beginning of the scan and each restart interval the conditioning derived from the line above is set to zero for the first line of each component. At the start of each line, the difference to the left is set to zero for the purposes of calculating the conditioning.

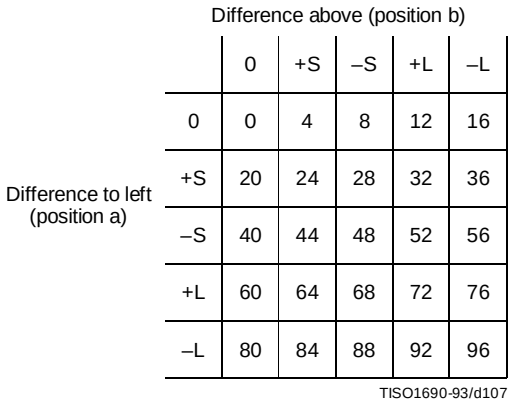


Figure H.2 – 5 × 5 Conditioning array for two-dimensional statistical model

H.1.2.3.2 Assignment of statistical bins to the DC binary decision tree

Each statistics area for lossless coding consists of a contiguous set of 158 statistics bins. The first 100 bins consist of 25 sets of four bins selected by a context-index S0. The value of S0 is given by L_Context(Da,Db), which provides a value of 0, 4,..., 92 or 96, depending on the difference classifications of Da and Db (see H.1.2.3.1). The value for S0 provided by L_Context(Da,Db) is from the array in Figure H.2.

The remaining 58 bins consist of two sets of 29 bins, X1, ..., X15, M2, ..., M15, which are used to code magnitude category decisions and magnitude bits. The value of X1 is given by X1_Context(Db), which provides a value of 100 when Db is in the zero, small positive or small negative categories and a value of 129 when Db is in the large positive or large negative categories.

The assignment of statistical bins to the binary decision tree used for coding the difference is given in Table H.3.

Table H.3 – Statistical model for lossless coding

Context-index	Value	Coding decision
S0	L_Context(Da,Db)	V = 0
SS	S0 + 1	Sign
SP	S0 + 2	Sz < 1 if V > 0
SN	S0 + 3	Sz < 1 if V < 0
X1	X1_Context(Db)	Sz < 2
X2	X1 + 1	Sz < 4
X3	X1 + 2	Sz < 8
.	.	.
.	.	.
X15	X1 + 14	$Sz < 2^{15}$
M2	X2 + 14	Magnitude bits if Sz < 4
M3	X3 + 14	Magnitude bits if Sz < 8
.	.	.
.	.	.
M15	X15 + 14	Magnitude bits if $Sz < 2^{15}$

H.1.2.3.3 Default conditioning bounds

The bounds, L and U, for determining the conditioning category have the default values L = 0 and U = 1. Other bounds may be set using the DAC (Define-Arithmetic-Conditioning) marker segment, as described in Annex B.

H.1.2.3.4 Initial conditions for statistical model

At the start of a scan and at each restart, all statistics bins are re-initialized to the standard default value described in Annex D.

H.2 Lossless decoder processes

Lossless decoders may employ either Huffman decoding or arithmetic decoding. They shall be capable of using up to four tables in a scan. Lossless decoders shall be able to decode encoded image source data with any input precision from 2 to 16 bits per sample.

H.2.1 Lossless decoder control procedures

Subclause E.2 contains the decoder control procedures. In applying these procedures to the lossless decoder the data unit is one sample.

When the decoder is reset in the restart interval control procedure (see E.2.4) the prediction is reset to the same value used in the encoder (see H.1.2.1). If arithmetic coding is used, the statistics are also reset.

Restrictions on the restart interval are specified in H.1.1.

H.2.2 Coding model for lossless decoding

The predictor calculations defined in H.1.2 also apply to the lossless decoder processes.

The lossless decoders, decode the differences and add them, modulo 2^{16}, to the predictions to create the output. The lossless decoders shall be able to interpret the point transform parameter, and if non-zero, multiply the output of the lossless decoder by 2^{Pt}.

In order to avoid repetition, detailed flow charts of the lossless decoding procedures are omitted.

Annex J

Hierarchical mode of operation

(This annex forms an integral part of this Recommendation | International Standard)

This annex provides a **functional specification** of the coding processes for the hierarchical mode of operation.

In the hierarchical mode of operation each component is encoded or decoded in a non-differential frame. Such frames may be followed by a sequence of differential frames. A non-differential frame shall be encoded or decoded using the procedures defined in Annexes F, G and H. Differential frame procedures are defined in this annex.

The coding process for a hierarchical encoding containing DCT-based processes is defined as the highest numbered process listed in Table J.1 which is used to code any non-differential DCT-based or differential DCT-based frame in the compressed image data format. The coding process for a hierarchical encoding containing only lossless processes is defined to be the process used for the non-differential frames.

Table J.1 – Coding processes for hierarchical mode

Process	Non-differential frame specification	
1	Extended sequential DCT, Huffman, 8-bit	Annex F, process 2
2	Extended sequential DCT, arithmetic, 8-bit	Annex F, process 3
3	Extended sequential DCT, Huffman, 12-bit	Annex F, process 4
4	Extended sequential DCT, arithmetic, 12-bit	Annex F, process 5
5	Spectral selection only, Huffman, 8-bit	Annex G, process 1
6	Spectral selection only, arithmetic, 8-bit	Annex G, process 2
7	Full progression, Huffman, 8-bit	Annex G, process 3
8	Full progression, arithmetic, 8-bit	Annex G, process 4
9	Spectral selection only, Huffman, 12-bit	Annex G, process 5
10	Spectral selection only, arithmetic, 12-bit	Annex G, process 6
11	Full progression, Huffman, 12-bit	Annex G, process 7
12	Full progression, arithmetic, 12-bit	Annex G, process 8
13	Lossless, Huffman, 2 through 16 bits	Annex H, process 1
14	Lossless, arithmetic, 2 through 16 bits	Annex H, process 2

Hierarchical mode syntax requires a DHP marker segment that appears before the non-differential frame or frames. It may include EXP marker segments and differential frames which shall follow the initial non-differential frame. The frame structure in hierarchical mode is identical to the frame structure in non-hierarchical mode.

Either all non-differential frames within an image shall be coded with DCT-based processes, or all non-differential frames shall be coded with lossless processes. All frames within an image must use the same entropy coding procedure, either Huffman or arithmetic, with the exception that non-differential frames coded with the baseline process may occur in the same image with frames coded with arithmetic coding processes.

If the non-differential frames use DCT-based processes, all differential frames except the final frame for a component shall use DCT-based processes. The final differential frame for each component may use a differential lossless process.

If the non-differential frames use lossless processes, all differential frames shall use differential lossless processes.

For each of the processes listed in Table J.1, the encoding processes are specified in J.1, and decoding processes are specified in J.2.

NOTE – There is **no requirement** in this Specification that any encoder or decoder which embodies one of the above-named processes shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

In the hierarchical mode of operation each component is encoded or decoded in a non-differential frame followed by a sequence of differential frames. A non-differential frame shall use the procedures defined in Annexes F, G, and H. Differential frame procedures are defined in this annex.

J.1 Hierarchical encoding

J.1.1 Hierarchical control procedure for encoding an image

The control structure for encoding of an image using the hierarchical mode is given in Figure J.1.

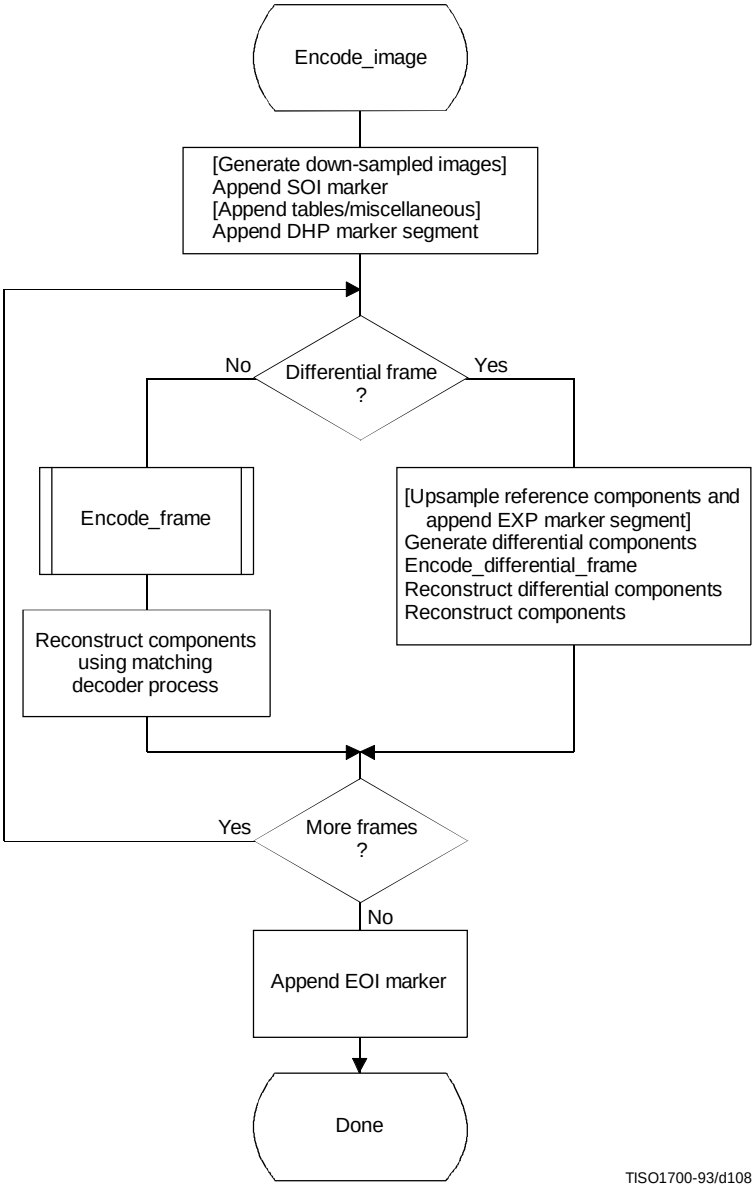


Figure J.1 – Hierarchical control procedure for encoding an image

In Figure J.1 procedures in brackets shall be performed whenever the particular hierarchical encoding sequence being followed requires them.

In the hierarchical mode the define-hierarchical-progression (DHP) marker segment shall be placed in the compressed image data before the first start-of-frame. The DHP segment is used to signal the size of the image components of the completed image. The syntax of the DHP segment is specified in Annex B.

The first frame for each component or group of components in a hierarchical process shall be encoded by a non-differential frame. Differential frames shall then be used to encode the two's complement differences between source input components (possibly downsampled) and the reference components (possibly upsampled). The reference components are reconstructed components created by previous frames in the hierarchical process. For either differential or non-differential frames, reconstructions of the components shall be generated if needed as reference components for a subsequent frame in the hierarchical process.

Resolution changes may occur between hierarchical frames in a hierarchical process. These changes occur if downsampling filters are used to reduce the spatial resolution of some or all of the components of the source image. When the resolution of a reference component does not match the resolution of the component input to a differential frame, an upsampling filter shall be used to increase the spatial resolution of the reference component. The EXP marker segment shall be added to the compressed image data before the start-of-frame whenever upsampling of a reference component is required. No more than one EXP marker segment shall precede a given frame.

Any of the marker segments allowed before a start-of-frame for the encoding process selected may be used before either non-differential or differential frames.

For 16-bit input precision (lossless encoder), the differential components which are input to a differential frame are calculated modulo 2^{16}. The reconstructed components calculated from the reconstructed differential components are also calculated modulo 2^{16}.

If a hierarchical encoding process uses a DCT encoding process for the first frame, all frames in the hierarchical process except for the final frame for each component shall use the DCT encoding processes defined in either Annex F or Annex G, or the modified DCT encoding processes defined in this annex. The final frame may use a modified lossless process defined in this annex.

If a hierarchical encoding process uses a lossless encoding process for the first frame, all frames in the hierarchical process shall use a lossless encoding process defined in Annex H, or a modified lossless process defined in this annex.

J.1.1.1 Downsampling filter

The downsampled components are generated using a downsampling filter that is not specified in this Specification. This filter should, however, be consistent with the upsampling filter. An example of a downsampling filter is provided in K.5.

J.1.1.2 Upsampling filter

The upsampling filter increases the spatial resolution by a factor of two horizontally, vertically, or both. Bi-linear interpolation is used for the upsampling filter, as illustrated in Figure J.2.

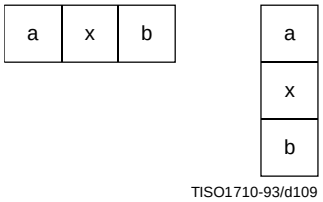


Figure J.2 – Diagram of sample positions for upsampling rules

The rule for calculating the interpolated value is:

$$P_x = (Ra + Rb) / 2$$

where Ra and Rb are sample values from adjacent positions a and b of the lower resolution image and Px is the interpolated value. The division indicates truncation, not rounding. The left-most column of the upsampled image matches the left-most column of the lower resolution image. The top line of the upsampled image matches the top line of the lower resolution image. The right column and the bottom line of the lower resolution image are replicated to provide the values required for the right column edge and bottom line interpolations. The upsampling process always doubles the line length or the number of lines.

If both horizontal and vertical expansions are signalled, they are done in sequence – first the horizontal expansion and then the vertical.

J.1.2 Control procedure for encoding a differential frame

The control procedures in Annex E for frames, scans, restart intervals, and MCU also apply to the encoding of differential frames, and the scans, restart intervals, and MCU from which the differential frame is constructed. The differential frames differ from the frames of Annexes F, G, and H only at the coding model level.

J.1.3 Encoder coding models for differential frames

The coding models defined in Annexes F, G, and H are modified to allow them to be used for coding of two's complement differences.

J.1.3.1 Modifications to encoder DCT encoding models for differential frames

Two modifications are made to the DCT coding models to allow them to be used in differential frames. First, the FDCT of the differential input is calculated without the level shift. Second, the DC coefficient of the DCT is coded directly – without prediction.

J.1.3.2 Modifications to lossless encoding models for differential frames

One modification is made to the lossless coding models. The difference is coded directly – without prediction. The prediction selection parameter in the scan header shall be set to zero. The point transform which may be applied to the differential inputs is defined in Annex A.

J.1.4 Modifications to the entropy encoders for differential frames

The coding of two's complement differences requires one extra bit of precision for the Huffman coding of AC coefficients. The extension to Tables F.1 and F.7 is given in Table J.2.

Table J.2 – Modifications to table of AC coefficient amplitude ranges

SSSS	AC coefficients
15	–32 767..–16 384, 16 384..32 767

The arithmetic coding models are already defined for the precision needed in differential frames.

J.2 Hierarchical decoding

J.2.1 Hierarchical control procedure for decoding an image

The control structure for decoding an image using the hierarchical mode is given in Figure J.3.

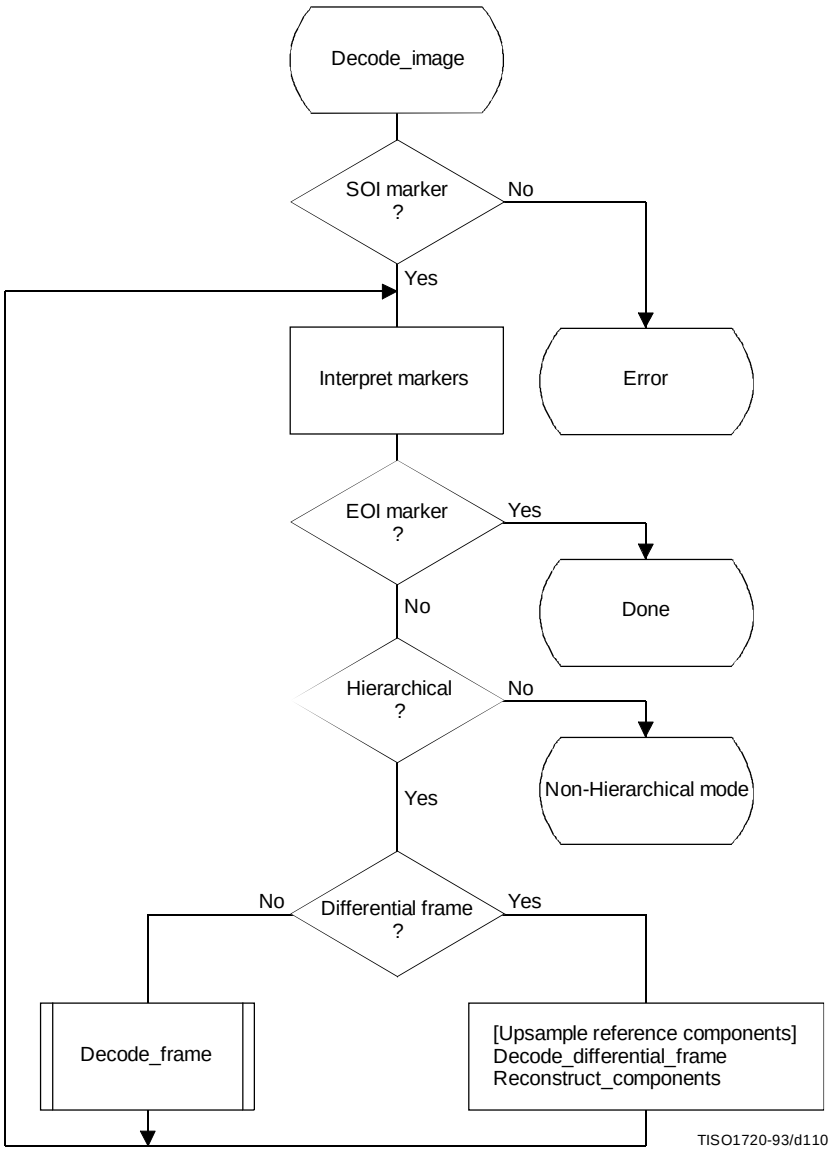


Figure J.3 – Hierarchical control procedure for decoding an image

The Interpret markers procedure shall decode the markers which may precede the SOF marker, continuing this decoding until either a SOF or EOI marker is found. If the DHP marker is encountered before the first frame, a flag is set which selects the hierarchical decoder at the "hierarchical?" decision point. In addition to the DHP marker (which shall precede any SOF) and the EXP marker (which shall precede any differential SOF requiring resolution changes in the reference components), any other markers which may precede a SOF shall be interpreted to the extent required for decoding of the compressed image data.

If a differential SOF marker is found, the differential frame path is followed. If the EXP was encountered in the Interpret markers procedure, the reference components for the frame shall be upsampled as required by the parameters in the EXP segment. The upsampling procedure described in J.1.1.2 shall be followed.

The Decode_differential_frame procedure generates a set of differential components. These differential components shall be added, modulo 2^{16}, to the upsampled reference components in the Reconstruct_components procedure. This creates a new set of reference components which shall be used when required in subsequent frames of the hierarchical process.

J.2.2 Control procedure for decoding a differential frame

The control procedures in Annex E for frames, scans, restart intervals, and MCU also apply to the decoding of differential frames and the scans, restart intervals, and MCU from which the differential frame is constructed. The differential frame differs from the frames of Annexes F, G, and H only at the decoder coding model level.

J.2.3 Decoder coding models for differential frames

The decoding models described in Annexes F, G, and H are modified to allow them to be used for decoding of two's complement differential components.

J.2.3.1 Modifications to the differential frame decoder DCT coding model

Two modifications are made to the decoder DCT coding models to allow them to code differential frames. First, the IDCT of the differential output is calculated without the level shift. Second, the DC coefficient of the DCT is decoded directly – without prediction.

J.2.3.2 Modifications to the differential frame decoder lossless coding model

One modification is made to the lossless decoder coding model. The difference is decoded directly – without prediction. If the point transformation parameter in the scan header is not zero, the point transform, defined in Annex A, shall be applied to the differential output.

J.2.4 Modifications to the entropy decoders for differential frames

The decoding of two's complement differences requires one extra bit of precision in the Huffman code table. This is described in J.1.4. The arithmetic coding models are already defined for the precision needed in differential frames.

Annex K

Examples and guidelines

(This annex does not form an integral part of this Recommendation | International Standard)

This annex provides examples of various tables, procedures, and other guidelines.

K.1 Quantization tables for luminance and chrominance components

Two examples of quantization tables are given in Tables K.1 and K.2. These are based on psychovisual thresholding and are derived empirically using luminance and chrominance and 2:1 horizontal subsampling. These tables are provided as examples only and are not necessarily suitable for any particular application. These quantization values have been used with good results on 8-bit per sample luminance and chrominance images of the format illustrated in Figure 13. Note that these quantization values are appropriate for the DCT normalization defined in A.3.3.

If these quantization values are divided by 2, the resulting reconstructed image is usually nearly indistinguishable from the source image.

Table K.1 – Luminance quantization table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table K.2 – Chrominance quantization table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

K.2 A procedure for generating the lists which specify a Huffman code table

A Huffman table is generated from a collection of statistics in two steps. The first step is the generation of the list of lengths and values which are in accord with the rules for generating the Huffman code tables. The second step is the generation of the Huffman code table from the list of lengths and values.

The first step, the topic of this section, is needed only for custom Huffman table generation and is done only in the encoder. In this step the statistics are used to create a table associating each value to be coded with the size (in bits) of the corresponding Huffman code. This table is sorted by code size.

A procedure for creating a Huffman table for a set of up to 256 symbols is shown in Figure K.1. Three vectors are defined for this procedure:

FREQ(V)	Frequency of occurrence of symbol V
CODESIZE(V)	Code size of symbol V
OTHERS(V)	Index to next symbol in chain of all symbols in current branch of code tree

where V goes from 0 to 256.

Before starting the procedure, the values of FREQ are collected for V = 0 to 255 and the FREQ value for V = 256 is set to 1 to reserve one code point. FREQ values for unused symbols are defined to be zero. In addition, the entries in CODESIZE are all set to 0, and the indices in OTHERS are set to –1, the value which terminates a chain of indices. Reserving one code point guarantees that no code word can ever be all “1” bits.

The search for the entry with the least value of FREQ(V) selects the largest value of V with the least value of FREQ(V) greater than zero.

The procedure “Find V1 for least value of FREQ(V1) > 0” always selects the value with the largest value of V1 when more than one V1 with the same frequency occurs. The reserved code point is then guaranteed to be in the longest code word category.

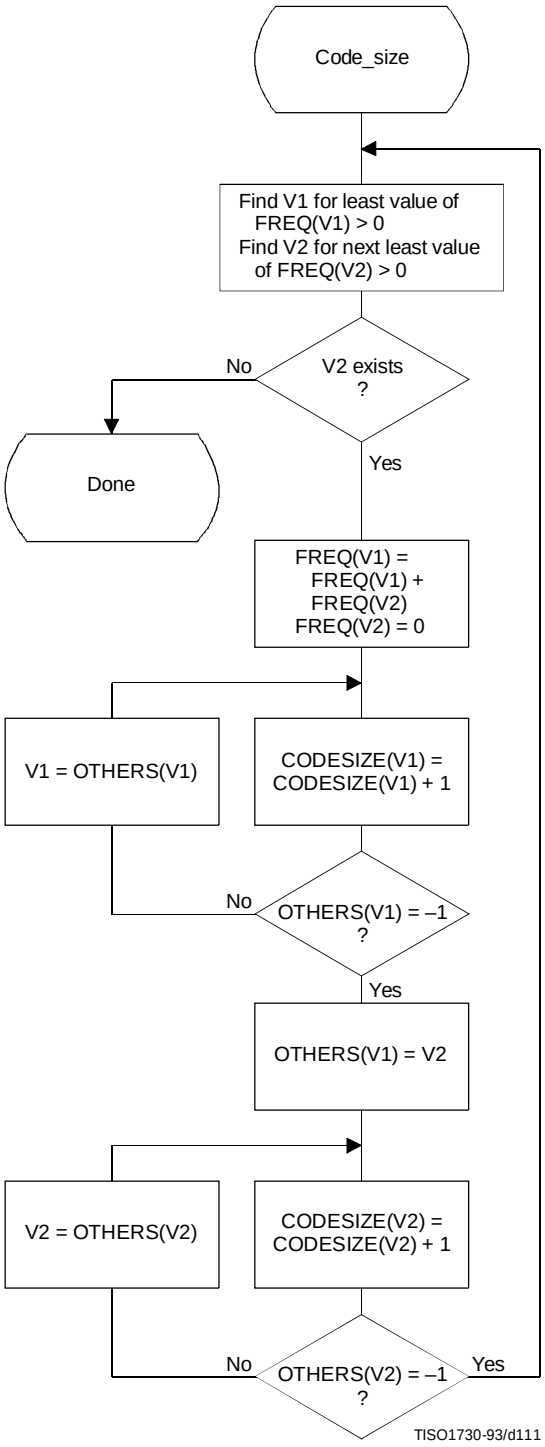


Figure K.1 – Procedure to find Huffman code sizes

Once the code lengths for each symbol have been obtained, the number of codes of each length is obtained using the procedure in Figure K.2. The count for each size is contained in the list, BITS. The counts in BITS are zero at the start of the procedure. The procedure assumes that the probabilities are large enough that code lengths greater than 32 bits never occur. Note that until the final Adjust_BITS procedure is complete, BITS may have more than the 16 entries required in the table specification (see Annex C).

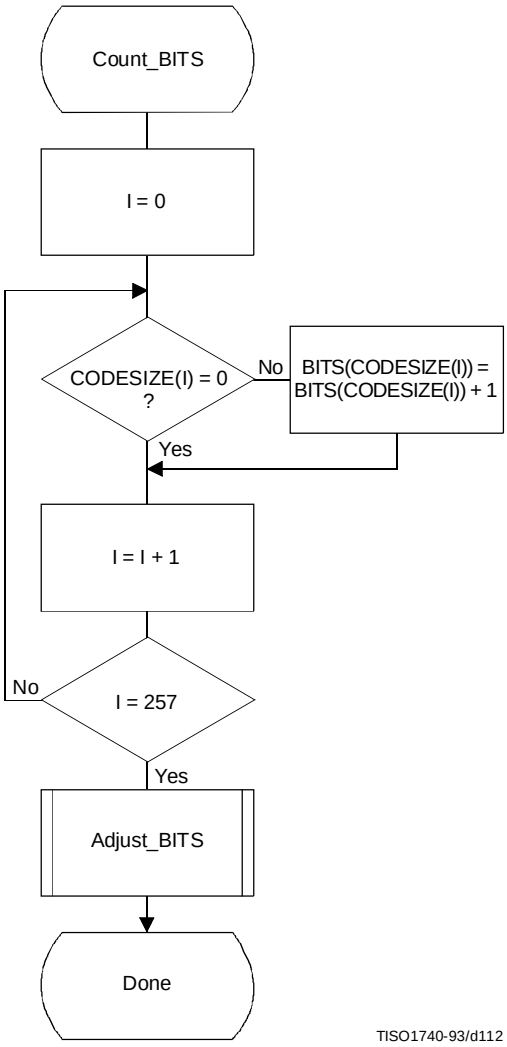


Figure K.2 – Procedure to find the number of codes of each size

Figure K.3 gives the procedure for adjusting the BITS list so that no code is longer than 16 bits. Since symbols are paired for the longest Huffman code, the symbols are removed from this length category two at a time. The prefix for the pair (which is one bit shorter) is allocated to one of the pair; then (skipping the BITS entry for that prefix length) a code word from the next shortest non-zero BITS entry is converted into a prefix for two code words one bit longer. After the BITS list is reduced to a maximum code length of 16 bits, the last step removes the reserved code point from the code length count.

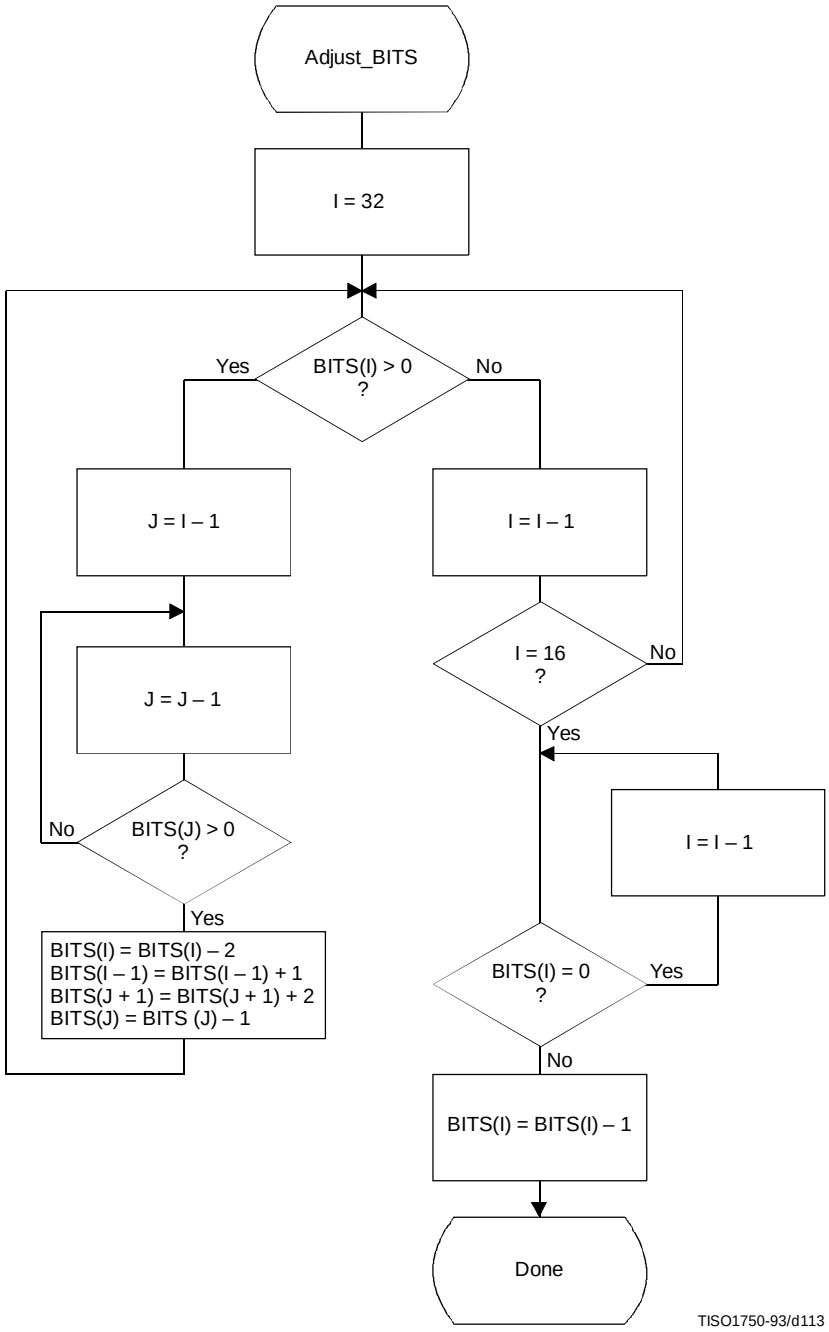


Figure K.3 – Procedure for limiting code lengths to 16 bits

The input values are sorted according to code size as shown in Figure K.4. HUFFVAL is the list containing the input values associated with each code word, in order of increasing code length.

At this point, the list of code lengths (BITS) and the list of values (HUFFVAL) can be used to generate the code tables. These procedures are described in Annex C.

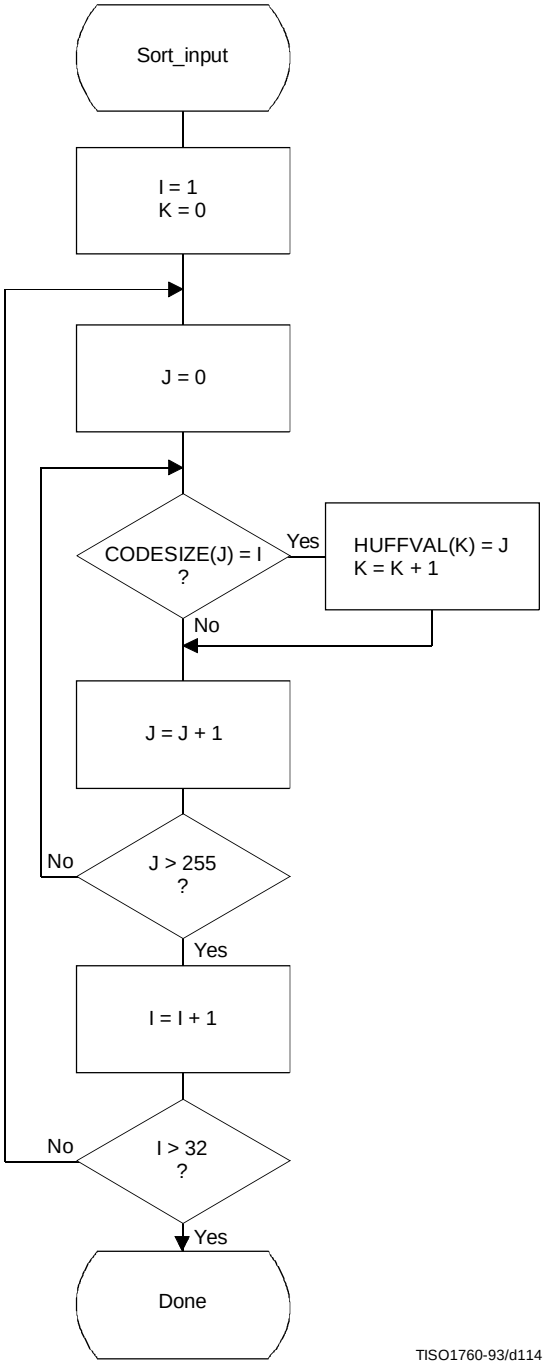


Figure K.4 – Sorting of input values according to code size

K.3 Typical Huffman tables for 8-bit precision luminance and chrominance

Huffman table-specification syntax is specified in B.2.4.2.

K.3.1 Typical Huffman tables for the DC coefficient differences

Tables K.3 and K.4 give Huffman tables for the DC coefficient differences which have been developed from the average statistics of a large set of video images with 8-bit precision. Table K.3 is appropriate for luminance components and Table K.4 is appropriate for chrominance components. Although there are no default tables, these tables may prove to be useful for many applications.

Table K.3 – Table for luminance DC coefficient differences

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Table K.4 – Table for chrominance DC coefficient differences

Category	Code length	Code word
0	2	00
1	2	01
2	2	10
3	3	110
4	4	1110
5	5	11110
6	6	111110
7	7	1111110
8	8	11111110
9	9	111111110
10	10	1111111110
11	11	11111111110

K.3.2 Typical Huffman tables for the AC coefficients

Tables K.5 and K.6 give Huffman tables for the AC coefficients which have been developed from the average statistics of a large set of images with 8-bit precision. Table K.5 is appropriate for luminance components and Table K.6 is appropriate for chrominance components. Although there are no default tables, these tables may prove to be useful for many applications.

Table K.5 – Table for luminance AC coefficients (sheet 1 of 4)

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110
3/1	6	111010
3/2	9	111110111
3/3	12	111111110101
3/4	16	1111111110001111
3/5	16	1111111110010000
3/6	16	1111111110010001
3/7	16	1111111110010010
3/8	16	1111111110010011
3/9	16	1111111110010100
3/A	16	1111111110010101

Table K.5 (sheet 2 of 4)

Run/Size	Code length	Code word
4/1	6	111011
4/2	10	1111111000
4/3	16	1111111110010110
4/4	16	1111111110010111
4/5	16	1111111110011000
4/6	16	1111111110011001
4/7	16	1111111110011010
4/8	16	1111111110011011
4/9	16	1111111110011100
4/A	16	1111111110011101
5/1	7	1111010
5/2	11	11111110111
5/3	16	1111111110011110
5/4	16	1111111110011111
5/5	16	1111111110100000
5/6	16	1111111110100001
5/7	16	1111111110100010
5/8	16	1111111110100011
5/9	16	1111111110100100
5/A	16	1111111110100101
6/1	7	1111011
6/2	12	111111110110
6/3	16	1111111110100110
6/4	16	1111111110100111
6/5	16	1111111110101000
6/6	16	1111111110101001
6/7	16	1111111110101010
6/8	16	1111111110101011
6/9	16	1111111110101100
6/A	16	1111111110101101
7/1	8	11111010
7/2	12	111111110111
7/3	16	1111111110101110
7/4	16	1111111110101111
7/5	16	1111111110110000
7/6	16	1111111110110001
7/7	16	1111111110110010
7/8	16	1111111110110011
7/9	16	1111111110110100
7/A	16	1111111110110101
8/1	9	111111000
8/2	15	111111111000000

Table K.5 (sheet 3 of 4)

Run/Size	Code length	Code word
8/3	16	1111111110110110
8/4	16	1111111110110111
8/5	16	1111111110111000
8/6	16	1111111110111001
8/7	16	1111111110111010
8/8	16	1111111110111011
8/9	16	1111111110111100
8/A	16	1111111110111101
9/1	9	111111001
9/2	16	1111111110111110
9/3	16	1111111110111111
9/4	16	1111111111000000
9/5	16	1111111111000001
9/6	16	1111111111000010
9/7	16	1111111111000011
9/8	16	1111111111000100
9/9	16	1111111111000101
9/A	16	1111111111000110
A/1	9	111111010
A/2	16	1111111111000111
A/3	16	1111111111001000
A/4	16	1111111111001001
A/5	16	1111111111001010
A/6	16	1111111111001011
A/7	16	1111111111001100
A/8	16	1111111111001101
A/9	16	1111111111001110
A/A	16	1111111111001111
B/1	10	1111111001
B/2	16	1111111111010000
B/3	16	1111111111010001
B/4	16	1111111111010010
B/5	16	1111111111010011
B/6	16	1111111111010100
B/7	16	1111111111010101
B/8	16	1111111111010110
B/9	16	1111111111010111
B/A	16	1111111111011000
C/1	10	1111111010
C/2	16	1111111111011001
C/3	16	1111111111011010
C/4	16	1111111111011011

Table K.5 (sheet 4 of 4)

Run/Size	Code length	Code word
C/5	16	1111111111011100
C/6	16	1111111111011101
C/7	16	1111111111011110
C/8	16	1111111111011111
C/9	16	1111111111100000
C/A	16	1111111111100001
D/1	11	11111111000
D/2	16	1111111111100010
D/3	16	1111111111100011
D/4	16	1111111111100100
D/5	16	1111111111100101
D/6	16	1111111111100110
D/7	16	1111111111100111
D/8	16	1111111111101000
D/9	16	1111111111101001
D/A	16	1111111111101010
E/1	16	1111111111101011
E/2	16	1111111111101100
E/3	16	1111111111101101
E/4	16	1111111111101110
E/5	16	1111111111101111
E/6	16	1111111111110000
E/7	16	1111111111110001
E/8	16	1111111111110010
E/9	16	1111111111110011
E/A	16	1111111111110100
F/0 (ZRL)	11	11111111001
F/1	16	1111111111110101
F/2	16	1111111111110110
F/3	16	1111111111110111
F/4	16	1111111111111000
F/5	16	1111111111111001
F/6	16	1111111111111010
F/7	16	1111111111111011
F/8	16	1111111111111100
F/9	16	1111111111111101
F/A	16	1111111111111110

Table K.6 – Table for chrominance AC coefficients (sheet 1 of 4)

Run/Size	Code length	Code word
0/0 (EOB)	2	00
0/1	2	01
0/2	3	100
0/3	4	1010
0/4	5	11000
0/5	5	11001
0/6	6	111000
0/7	7	1111000
0/8	9	111110100
0/9	10	1111110110
0/A	12	111111110100
1/1	4	1011
1/2	6	111001
1/3	8	11110110
1/4	9	111110101
1/5	11	11111110110
1/6	12	111111110101
1/7	16	1111111110001000
1/8	16	1111111110001001
1/9	16	1111111110001010
1/A	16	1111111110001011
2/1	5	11010
2/2	8	11110111
2/3	10	1111110111
2/4	12	111111110110
2/5	15	111111111000010
2/6	16	1111111110001100
2/7	16	1111111110001101
2/8	16	1111111110001110
2/9	16	1111111110001111
2/A	16	1111111110010000
3/1	5	11011
3/2	8	11111000
3/3	10	1111111000
3/4	12	111111110111
3/5	16	1111111110010001
3/6	16	1111111110010010
3/7	16	1111111110010011
3/8	16	1111111110010100
3/9	16	1111111110010101
3/A	16	1111111110010110
4/1	6	111010

Table K.6 (sheet 2 of 4)

Run/Size	Code length	Code word
4/2	9	111110110
4/3	16	1111111110010111
4/4	16	1111111110011000
4/5	16	1111111110011001
4/6	16	1111111110011010
4/7	16	1111111110011011
4/8	16	1111111110011100
4/9	16	1111111110011101
4/A	16	1111111110011110
5/1	6	111011
5/2	10	1111111001
5/3	16	1111111110011111
5/4	16	1111111110100000
5/5	16	1111111110100001
5/6	16	1111111110100010
5/7	16	1111111110100011
5/8	16	1111111110100100
5/9	16	1111111110100101
5/A	16	1111111110100110
6/1	7	1111001
6/2	11	11111110111
6/3	16	1111111110100111
6/4	16	1111111110101000
6/5	16	1111111110101001
6/6	16	1111111110101010
6/7	16	1111111110101011
6/8	16	1111111110101100
6/9	16	1111111110101101
6/A	16	1111111110101110
7/1	7	1111010
7/2	11	11111111000
7/3	16	1111111110101111
7/4	16	1111111110110000
7/5	16	1111111110110001
7/6	16	1111111110110010
7/7	16	1111111110110011
7/8	16	1111111110110100
7/9	16	1111111110110101
7/A	16	1111111110110110
8/1	8	11111001
8/2	16	1111111110110111
8/3	16	1111111110111000

Table K.6 (sheet 3 of 4)

Run/Size	Code length	Code word
8/4	16	1111111110111001
8/5	16	1111111110111010
8/6	16	1111111110111011
8/7	16	1111111110111100
8/8	16	1111111110111101
8/9	16	1111111110111110
8/A	16	1111111110111111
9/1	9	111110111
9/2	16	1111111111000000
9/3	16	1111111111000001
9/4	16	1111111111000010
9/5	16	1111111111000011
9/6	16	1111111111000100
9/7	16	1111111111000101
9/8	16	1111111111000110
9/9	16	1111111111000111
9/A	16	1111111111001000
A/1	9	111111000
A/2	16	1111111111001001
A/3	16	1111111111001010
A/4	16	1111111111001011
A/5	16	1111111111001100
A/6	16	1111111111001101
A/7	16	1111111111001110
A/8	16	1111111111001111
A/9	16	1111111111010000
A/A	16	1111111111010001
B/1	9	111111001
B/2	16	1111111111010010
B/3	16	1111111111010011
B/4	16	1111111111010100
B/5	16	1111111111010101
B/6	16	1111111111010110
B/7	16	1111111111010111
B/8	16	1111111111011000
B/9	16	1111111111011001
B/A	16	1111111111011010
C/1	9	111111010
C/2	16	1111111111011011
C/3	16	1111111111011100
C/4	16	1111111111011101
C/5	16	1111111111011110

Table K.6 (sheet 4 of 4)

Run/Size	Code length	Code word
C/6	16	1111111111011111
C/7	16	1111111111100000
C/8	16	1111111111100001
C/9	16	1111111111100010
C/A	16	1111111111100011
D/1	11	11111111001
D/2	16	1111111111100100
D/3	16	1111111111100101
D/4	16	1111111111100110
D/5	16	1111111111100111
D/6	16	1111111111101000
D/7	16	1111111111101001
D/8	16	1111111111101010
D/9	16	1111111111101011
D/A	16	1111111111101100
E/1	14	11111111100000
E/2	16	1111111111101101
E/3	16	1111111111101110
E/4	16	1111111111101111
E/5	16	1111111111110000
E/6	16	1111111111110001
E/7	16	1111111111110010
E/8	16	1111111111110011
E/9	16	1111111111110100
E/A	16	1111111111110101
F/0 (ZRL)	10	1111111010
F/1	15	111111111000011
F/2	16	1111111111110110
F/3	16	1111111111110111
F/4	16	1111111111111000
F/5	16	1111111111111001
F/6	16	1111111111111010
F/7	16	1111111111111011
F/8	16	1111111111111100
F/9	16	1111111111111101
F/A	16	1111111111111110

K.3.3 Huffman table-specification examples

K.3.3.1 Specification of typical tables for DC difference coding

A set of typical tables for DC component coding is given in K.3.1. The specification of these tables is as follows:

For Table K.3 (for luminance DC coefficients), the 16 bytes which specify the list of code lengths for the table are

X'00 01 05 01 01 01 01 01 01 00 00 00 00 00 00 00'

The set of values following this list is

X'00 01 02 03 04 05 06 07 08 09 0A 0B'

For Table K.4 (for chrominance DC coefficients), the 16 bytes which specify the list of code lengths for the table are

X'00 03 01 01 01 01 01 01 01 01 01 00 00 00 00 00'

The set of values following this list is

X'00 01 02 03 04 05 06 07 08 09 0A 0B'

K.3.3.2 Specification of typical tables for AC coefficient coding

A set of typical tables for AC component coding is given in K.3.2. The specification of these tables is as follows:

For Table K.5 (for luminance AC coefficients), the 16 bytes which specify the list of code lengths for the table are

X'00 02 01 03 03 02 04 03 05 05 04 04 00 00 01 7D'

The set of values which follows this list is

X'01 02 03 00 04 11 05 12 21 31 41 06 13 51 61 07
22 71 14 32 81 91 A1 08 23 42 B1 C1 15 52 D1 F0
24 33 62 72 82 09 0A 16 17 18 19 1A 25 26 27 28
29 2A 34 35 36 37 38 39 3A 43 44 45 46 47 48 49
4A 53 54 55 56 57 58 59 5A 63 64 65 66 67 68 69
6A 73 74 75 76 77 78 79 7A 83 84 85 86 87 88 89
8A 92 93 94 95 96 97 98 99 9A A2 A3 A4 A5 A6 A7
A8 A9 AA B2 B3 B4 B5 B6 B7 B8 B9 BA C2 C3 C4 C5
C6 C7 C8 C9 CA D2 D3 D4 D5 D6 D7 D8 D9 DA E1 E2
E3 E4 E5 E6 E7 E8 E9 EA F1 F2 F3 F4 F5 F6 F7 F8
F9 FA'

For Table K.6 (for chrominance AC coefficients), the 16 bytes which specify the list of code lengths for the table are

X'00 02 01 02 04 04 03 04 07 05 04 04 00 01 02 77'

The set of values which follows this list is:

X'00 01 02 03 11 04 05 21 31 06 12 41 51 07 61 71
13 22 32 81 08 14 42 91 A1 B1 C1 09 23 33 52 F0
15 62 72 D1 0A 16 24 34 E1 25 F1 17 18 19 1A 26
27 28 29 2A 35 36 37 38 39 3A 43 44 45 46 47 48
49 4A 53 54 55 56 57 58 59 5A 63 64 65 66 67 68
69 6A 73 74 75 76 77 78 79 7A 82 83 84 85 86 87
88 89 8A 92 93 94 95 96 97 98 99 9A A2 A3 A4 A5
A6 A7 A8 A9 AA B2 B3 B4 B5 B6 B7 B8 B9 BA C2 C3
C4 C5 C6 C7 C8 C9 CA D2 D3 D4 D5 D6 D7 D8 D9 DA
E2 E3 E4 E5 E6 E7 E8 E9 EA F2 F3 F4 F5 F6 F7 F8
F9 FA'

K.4 Additional information on arithmetic coding

K.4.1 Test sequence for a small data set for the arithmetic coder

The following 256-bit test sequence (in hexadecimal form) is structured to test many of the encoder and decoder paths:

X'00020051 000000C0 0352872A AAAAAAAA 82C02000 FCD79EF6 74EAABF7 697EE74C'

Tables K.7 and K.8 provide a symbol-by-symbol list of the arithmetic encoder and decoder operation. In these tables the event count, EC, is listed first, followed by the value of Qe used in encoding and decoding that event. The decision D to be encoded (and decoded) is listed next. The column labeled MPS contains the sense of the MPS, and if it is followed by a CE (in the "CX" column), the conditional MPS/LPS exchange occurs when encoding and decoding the decision (see Figures D.3, D.4 and D.17). The contents of the A and C registers are the values before the event is encoded and decoded. ST is the number of X'FF' bytes stacked in the encoder waiting for a resolution of the carry-over. Note that the A register is always greater than X'7FFF'. (The starting value has an implied value of X'10000'.)

In the encoder test, the code bytes (B) are listed if they were completed during the coding of the preceding event. If additional bytes follow, they were also completed during the coding of the preceding event. If a byte is listed in the Bx column, the preceding byte in column B was modified by a carry-over.

In the decoder the code bytes are listed if they were placed in the code register just prior to the event EC.

For this file the coded bit count is 240, including the overhead to flush the final data from the C register. When the marker X'FFD9' is appended, a total of 256 bits are output. The actual compressed data sequence for the encoder is (in hexadecimal form)

X'655B5144 F7969D51 7855BFFF 00FC5184 C7CEF939 00287D46 708ECBC0 F6FFD900'

Table K.7 – Encoder test sequence (sheet 1 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
1	0	0		5A1D	0000	00000000	11	0		
2	0	0	CE	5A1D	A5E3	00000000	11	0		
3	0	0		2586	B43A	0000978C	10	0		
4	0	0		2586	8EB4	0000978C	10	0		
5	0	0		1114	D25C	00012F18	9	0		
6	0	0		1114	C148	00012F18	9	0		
7	0	0		1114	B034	00012F18	9	0		
8	0	0		1114	9F20	00012F18	9	0		
9	0	0		1114	8E0C	00012F18	9	0		
10	0	0		080B	F9F0	00025E30	8	0		
11	0	0		080B	F1E5	00025E30	8	0		
12	0	0		080B	E9DA	00025E30	8	0		
13	0	0		080B	E1CF	00025E30	8	0		
14	0	0		080B	D9C4	00025E30	8	0		
15	1	0		080B	D1B9	00025E30	8	0		
16	0	0		17B9	80B0	00327DE0	4	0		
17	0	0		1182	D1EE	0064FBC0	3	0		
18	0	0		1182	C06C	0064FBC0	3	0		
19	0	0		1182	AEEA	0064FBC0	3	0		
20	0	0		1182	9D68	0064FBC0	3	0		
21	0	0		1182	8BE6	0064FBC0	3	0		
22	0	0		0CEF	F4C8	00C9F780	2	0		
23	0	0		0CEF	E7D9	00C9F780	2	0		
24	0	0		0CEF	DAEA	00C9F780	2	0		
25	0	0		0CEF	CDFB	00C9F780	2	0		
26	1	0		0CEF	C10C	00C9F780	2	0		
27	0	0		1518	CEF0	000AB9D0	6	0		65
28	1	0		1518	B9D8	000AB9D0	6	0		
29	0	0		1AA9	A8C0	005AF480	3	0		
30	0	0		1AA9	8E17	005AF480	3	0		
31	0	0		174E	E6DC	00B5E900	2	0		
32	1	0		174E	CF8E	00B5E900	2	0		
33	0	0		1AA9	BA70	00050A00	7	0		5B
34	0	0		1AA9	9FC7	00050A00	7	0		
35	0	0		1AA9	851E	00050A00	7	0		
36	0	0		174E	D4EA	000A1400	6	0		

Table K.7 – Encoder test sequence (sheet 2 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
37	0	0		174E	BD9C	000A1400	6	0		
38	0	0		174E	A64E	000A1400	6	0		
39	0	0		174E	8F00	000A1400	6	0		
40	0	0		1424	EF64	00142800	5	0		
41	0	0		1424	DB40	00142800	5	0		
42	0	0		1424	C71C	00142800	5	0		
43	0	0		1424	B2F8	00142800	5	0		
44	0	0		1424	9ED4	00142800	5	0		
45	0	0		1424	8AB0	00142800	5	0		
46	0	0		119C	ED18	00285000	4	0		
47	0	0		119C	DB7C	00285000	4	0		
48	0	0		119C	C9E0	00285000	4	0		
49	0	0		119C	B844	00285000	4	0		
50	0	0		119C	A6A8	00285000	4	0		
51	0	0		119C	950C	00285000	4	0		
52	0	0		119C	8370	00285000	4	0		
53	0	0		0F6B	E3A8	0050A000	3	0		
54	0	0		0F6B	D43D	0050A000	3	0		
55	0	0		0F6B	C4D2	0050A000	3	0		
56	0	0		0F6B	B567	0050A000	3	0		
57	1	0		0F6B	A5FC	0050A000	3	0		
58	1	0		1424	F6B0	00036910	7	0		51
59	0	0		1AA9	A120	00225CE0	4	0		
60	0	0		1AA9	8677	00225CE0	4	0		
61	0	0		174E	D79C	0044B9C0	3	0		
62	0	0		174E	C04E	0044B9C0	3	0		
63	0	0		174E	A900	0044B9C0	3	0		
64	0	0		174E	91B2	0044B9C0	3	0		
65	0	0		1424	F4C8	00897380	2	0		
66	0	0		1424	E0A4	00897380	2	0		
67	0	0		1424	CC80	00897380	2	0		
68	0	0		1424	B85C	00897380	2	0		
69	0	0		1424	A438	00897380	2	0		
70	0	0		1424	9014	00897380	2	0		
71	1	0		119C	F7E0	0112E700	1	0		
72	1	0		1424	8CE0	001E6A20	6	0		44
73	0	0		1AA9	A120	00F716E0	3	0		

Table K.7 – Encoder test sequence (sheet 3 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
74	1	0		1AA9	8677	00F716E0	3	0		
75	0	0		2516	D548	00041570	8	0		F7
76	1	0		2516	B032	00041570	8	0		
77	0	0		299A	9458	00128230	6	0		
78	0	0		2516	D57C	00250460	5	0		
79	1	0		2516	B066	00250460	5	0		
80	0	0		299A	9458	00963EC0	3	0		
81	1	0		2516	D57C	012C7D80	2	0		
82	0	0		299A	9458	0004B798	8	0		96
83	0	0		2516	D57C	00096F30	7	0		
84	0	0		2516	B066	00096F30	7	0		
85	0	0		2516	8B50	00096F30	7	0		
86	1	0		1EDF	CC74	0012DE60	6	0		
87	1	0		2516	F6F8	009C5FA8	3	0		
88	1	0		299A	9458	0274C628	1	0		
89	0	0		32B4	A668	0004C398	7	0		9D
90	0	0		2E17	E768	00098730	6	0		
91	1	0		2E17	B951	00098730	6	0		
92	0	0		32B4	B85C	002849A8	4	0		
93	1	0		32B4	85A8	002849A8	4	0		
94	0	0		3C3D	CAD0	00A27270	2	0		
95	1	0		3C3D	8E93	00A27270	2	0		
96	0	0		415E	F0F4	00031318	8	0		51
97	1	0		415E	AF96	00031318	8	0		
98	0	0	CE	4639	82BC	000702A0	7	0		
99	1	0		415E	8C72	000E7E46	6	0		
100	0	0	CE	4639	82BC	001D92B4	5	0		
101	1	0		415E	8C72	003B9E6E	4	0		
102	0	0	CE	4639	82BC	0077D304	3	0		
103	1	0		415E	8C72	00F01F0E	2	0		
104	0	0	CE	4639	82BC	01E0D444	1	0		
105	1	0		415E	8C72	0002218E	8	0		78
106	0	0	CE	4639	82BC	0004D944	7	0		
107	1	0		415E	8C72	000A2B8E	6	0		
108	0	0	CE	4639	82BC	0014ED44	5	0		
109	1	0		415E	8C72	002A538E	4	0		
110	0	0	CE	4639	82BC	00553D44	3	0		

Table K.7 – Encoder test sequence (sheet 4 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
111	1	0		415E	8C72	00AAF38E	2	0		
112	0	0	CE	4639	82BC	01567D44	1	0		
113	1	0		415E	8C72	0005738E	8	0		55
114	0	0	CE	4639	82BC	000B7D44	7	0		
115	1	0		415E	8C72	0017738E	6	0		
116	0	0	CE	4639	82BC	002F7D44	5	0		
117	1	0		415E	8C72	005F738E	4	0		
118	0	0	CE	4639	82BC	00BF7D44	3	0		
119	1	0		415E	8C72	017F738E	2	0		
120	0	0	CE	4639	82BC	02FF7D44	1	0		
121	1	0		415E	8C72	0007738E	8	0		BF
122	0	0	CE	4639	82BC	000F7D44	7	0		
123	1	0		415E	8C72	001F738E	6	0		
124	0	0	CE	4639	82BC	003F7D44	5	0		
125	1	0		415E	8C72	007F738E	4	0		
126	0	0	CE	4639	82BC	00FF7D44	3	0		
127	1	0		415E	8C72	01FF738E	2	0		
128	0	0	CE	4639	82BC	03FF7D44	1	0		
129	1	0		415E	8C72	0007738E	8	1		
130	0	0	CE	4639	82BC	000F7D44	7	1		
131	0	0		415E	8C72	001F738E	6	1		
132	0	0		3C3D	9628	003EE71C	5	1		
133	0	0		375E	B3D6	007DCE38	4	1		
134	0	0		32B4	F8F0	00FB9C70	3	1		
135	1	0		32B4	C63C	00FB9C70	3	1		
136	0	0		3C3D	CAD0	03F0BFE0	1	1		
137	1	0		3C3D	8E93	03F0BFE0	1	1		
138	1	0		415E	F0F4	000448D8	7	0		FF00FC
139	0	0	CE	4639	82BC	0009F0DC	6	0		
140	0	0		415E	8C72	00145ABE	5	0		
141	0	0		3C3D	9628	0028B57C	4	0		
142	0	0		375E	B3D6	00516AF8	3	0		
143	0	0		32B4	F8F0	00A2D5F0	2	0		
144	0	0		32B4	C63C	00A2D5F0	2	0		
145	0	0		32B4	9388	00A2D5F0	2	0		
146	0	0		2E17	C1A8	0145ABE0	1	0		

Table K.7 – Encoder test sequence (sheet 5 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
147	1	0		2E17	9391	0145ABE0	1	0		
148	0	0		32B4	B85C	00084568	7	0		51
149	0	0		32B4	85A8	00084568	7	0		
150	0	0		2E17	A5E8	00108AD0	6	0		
151	0	0		299A	EFA2	002115A0	5	0		
152	0	0		299A	C608	002115A0	5	0		
153	0	0		299A	9C6E	002115A0	5	0		
154	0	0		2516	E5A8	00422B40	4	0		
155	0	0		2516	C092	00422B40	4	0		
156	0	0		2516	9B7C	00422B40	4	0		
157	0	0		1EDF	ECCC	00845680	3	0		
158	0	0		1EDF	CDED	00845680	3	0		
159	0	0		1EDF	AF0E	00845680	3	0		
160	0	0		1EDF	902F	00845680	3	0		
161	1	0		1AA9	E2A0	0108AD00	2	0		
162	1	0		2516	D548	000BA7B8	7	0		84
163	1	0		299A	9458	00315FA8	5	0		
164	1	0		32B4	A668	00C72998	3	0		
165	1	0		3C3D	CAD0	031E7530	1	0		
166	1	0		415E	F0F4	000C0F0C	7	0		C7
167	0	0	CE	4639	82BC	00197D44	6	0		
168	0	0		415E	8C72	0033738E	5	0		
169	1	0		3C3D	9628	0066E71C	4	0		
170	1	0		415E	F0F4	019D041C	2	0		
171	0	0	CE	4639	82BC	033B6764	1	0		
172	1	0		415E	8C72	000747CE	8	0		CE
173	0	0	CE	4639	82BC	000F25C4	7	0		
174	1	0		415E	8C72	001EC48E	6	0		
175	1	0	CE	4639	82BC	003E1F44	5	0		
176	1	0		4B85	F20C	00F87D10	3	0		
177	1	0	CE	504F	970A	01F2472E	2	0		
178	0	0	CE	5522	8D76	03E48E5C	1	0		
179	0	0		504F	AA44	00018D60	8	0		F9
180	1	0		4B85	B3EA	00031AC0	7	0		
181	1	0	CE	504F	970A	0007064A	6	0		
182	1	0	CE	5522	8D76	000E0C94	5	0		
183	1	0		59EB	E150	00383250	3	0		

Table K.7 – Encoder test sequence (sheet 6 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
184	0	1		59EB	B3D6	0071736A	2	0		
185	1	0		59EB	B3D6	00E39AAA	1	0		
186	1	1		59EB	B3D6	0007E92A	8	0		38
187	1	1		5522	B3D6	000FD254	7	0		
188	1	1		504F	BD68	001FA4A8	6	0		
189	0	1		4B85	DA32	003F4950	5	0		
190	1	1	CE	504F	970A	007FAFFA	4	0		
191	1	1		4B85	A09E	00FFED6A	3	0		
192	0	1		4639	AA32	01FFDAD4	2	0		
193	0	1	CE	4B85	8C72	04007D9A	1	0		
194	1	1	CE	504F	81DA	0000FB34	8	0	39	00
195	1	1		4B85	A09E	0002597E	7	0		
196	1	1		4639	AA32	0004B2FC	6	0		
197	0	1		415E	C7F2	000965F8	5	0		
198	1	1	CE	4639	82BC	0013D918	4	0		
199	0	1		415E	8C72	00282B36	3	0		
200	0	1	CE	4639	82BC	0050EC94	2	0		
201	1	1		4B85	F20C	0003B250	8	0		28
202	1	1		4B85	A687	0003B250	8	0		
203	1	1		4639	B604	000764A0	7	0		
204	0	1		415E	DF96	000EC940	6	0		
205	1	1	CE	4639	82BC	001ECEF0	5	0		
206	0	1		415E	8C72	003E16E6	4	0		
207	1	1	CE	4639	82BC	007CC3F4	3	0		
208	0	1		415E	8C72	00FA00EE	2	0		
209	1	1	CE	4639	82BC	01F49804	1	0		
210	0	1		415E	8C72	0001A90E	8	0		7D
211	1	1	CE	4639	82BC	0003E844	7	0		
212	0	1		415E	8C72	0008498E	6	0		
213	1	1	CE	4639	82BC	00112944	5	0		
214	0	1		415E	8C72	0022CB8E	4	0		
215	1	1	CE	4639	82BC	00462D44	3	0		
216	1	1		415E	8C72	008CD38E	2	0		
217	1	1		3C3D	9628	0119A71C	1	0		
218	1	1		375E	B3D6	00034E38	8	0		46
219	1	1		32B4	F8F0	00069C70	7	0		
220	1	1		32B4	C63C	00069C70	7	0		

Table K.7 – Encoder test sequence (sheet 7 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
221	0	1		32B4	9388	00069C70	7	0		
222	1	1		3C3D	CAD0	001BF510	5	0		
223	1	1		3C3D	8E93	001BF510	5	0		
224	1	1		375E	A4AC	0037EA20	4	0		
225	0	1		32B4	DA9C	006FD440	3	0		
226	1	1		3C3D	CAD0	01C1F0A0	1	0		
227	1	1		3C3D	8E93	01C1F0A0	1	0		
228	0	1		375E	A4AC	0003E140	8	0		70
229	1	1		3C3D	DD78	00113A38	6	0		
230	0	1		3C3D	A13B	00113A38	6	0		
231	0	1		415E	F0F4	00467CD8	4	0		
232	1	1	CE	4639	82BC	008E58DC	3	0		
233	0	1		415E	8C72	011D2ABE	2	0		
234	1	1	CE	4639	82BC	023AEBA4	1	0		
235	1	1		415E	8C72	0006504E	8	0		8E
236	1	1		3C3D	9628	000CA09C	7	0		
237	1	1		375E	B3D6	00194138	6	0		
238	1	1		32B4	F8F0	00328270	5	0		
239	1	1		32B4	C63C	00328270	5	0		
240	0	1		32B4	9388	00328270	5	0		
241	1	1		3C3D	CAD0	00CB8D10	3	0		
242	1	1		3C3D	8E93	00CB8D10	3	0		
243	1	1		375E	A4AC	01971A20	2	0		
244	0	1		32B4	DA9C	032E3440	1	0		
245	0	1		3C3D	CAD0	000B70A0	7	0		CB
246	1	1		415E	F0F4	002FFCCC	5	0		
247	1	1		415E	AF96	002FFCCC	5	0		
248	1	1		3C3D	DC70	005FF998	4	0		
249	0	1		3C3D	A033	005FF998	4	0		
250	1	1		415E	F0F4	01817638	2	0		
251	0	1		415E	AF96	01817638	2	0		
252	0	1	CE	4639	82BC	0303C8E0	1	0		
253	1	1		4B85	F20C	000F2380	7	0		C0
254	1	1		4B85	A687	000F2380	7	0		
255	0	1		4639	B604	001E4700	6	0		
256	0	1	CE	4B85	8C72	003D6D96	5	0		
Flush:					81DA	007ADB2C	4	0		F6
										FFD9

Table K.8 – Decoder test sequence (sheet 1 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
1	0	0		5A1D	0000	655B0000	0	65 5B
2	0	0	CE	5A1D	A5E3	655B0000	0	
3	0	0		2586	B43A	332AA200	7	51
4	0	0		2586	8EB4	332AA200	7	
5	0	0		1114	D25C	66554400	6	
6	0	0		1114	C148	66554400	6	
7	0	0		1114	B034	66554400	6	
8	0	0		1114	9F20	66554400	6	
9	0	0		1114	8E0C	66554400	6	
10	0	0		080B	F9F0	CCAA8800	5	
11	0	0		080B	F1E5	CCAA8800	5	
12	0	0		080B	E9DA	CCAA8800	5	
13	0	0		080B	E1CF	CCAA8800	5	
14	0	0		080B	D9C4	CCAA8800	5	
15	1	0		080B	D1B9	CCAA8800	5	
16	0	0		17B9	80B0	2FC88000	1	
17	0	0		1182	D1EE	5F910000	0	
18	0	0		1182	C06C	5F910000	0	
19	0	0		1182	AEEA	5F910000	0	
20	0	0		1182	9D68	5F910000	0	
21	0	0		1182	8BE6	5F910000	0	
22	0	0		0CEF	F4C8	BF228800	7	44
23	0	0		0CEF	E7D9	BF228800	7	
24	0	0		0CEF	DAEA	BF228800	7	
25	0	0		0CEF	CDFB	BF228800	7	
26	1	0		0CEF	C10C	BF228800	7	
27	0	0		1518	CEF0	B0588000	3	
28	1	0		1518	B9D8	B0588000	3	
29	0	0		1AA9	A8C0	5CC40000	0	
30	0	0		1AA9	8E17	5CC40000	0	
31	0	0		174E	E6DC	B989EE00	7	F7
32	1	0		174E	CF8E	B989EE00	7	
33	0	0		1AA9	BA70	0A4F7000	4	
34	0	0		1AA9	9FC7	0A4F7000	4	
35	0	0		1AA9	851E	0A4F7000	4	
36	0	0		174E	D4EA	149EE000	3	
37	0	0		174E	BD9C	149EE000	3	
38	0	0		174E	A64E	149EE000	3	
39	0	0		174E	8F00	149EE000	3	
40	0	0		1424	EF64	293DC000	2	

Table K.8 – Decoder test sequence (sheet 2 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
41	0	0		1424	DB40	293DC000	2	
42	0	0		1424	C71C	293DC000	2	
43	0	0		1424	B2F8	293DC000	2	
44	0	0		1424	9ED4	293DC000	2	
45	0	0		1424	8AB0	293DC000	2	
46	0	0		119C	ED18	527B8000	1	
47	0	0		119C	DB7C	527B8000	1	
48	0	0		119C	C9E0	527B8000	1	
49	0	0		119C	B844	527B8000	1	
50	0	0		119C	A6A8	527B8000	1	
51	0	0		119C	950C	527B8000	1	
52	0	0		119C	8370	527B8000	1	
53	0	0		0F6B	E3A8	A4F70000	0	
54	0	0		0F6B	D43D	A4F70000	0	
55	0	0		0F6B	C4D2	A4F70000	0	
56	0	0		0F6B	B567	A4F70000	0	
57	1	0		0F6B	A5FC	A4F70000	0	
58	1	0		1424	F6B0	E6696000	4	96
59	0	0		1AA9	A120	1EEB0000	1	
60	0	0		1AA9	8677	1EEB0000	1	
61	0	0		174E	D79C	3DD60000	0	
62	0	0		174E	C04E	3DD60000	0	
63	0	0		174E	A900	3DD60000	0	
64	0	0		174E	91B2	3DD60000	0	
65	0	0		1424	F4C8	7BAD3A00	7	9D
66	0	0		1424	E0A4	7BAD3A00	7	
67	0	0		1424	CC80	7BAD3A00	7	
68	0	0		1424	B85C	7BAD3A00	7	
69	0	0		1424	A438	7BAD3A00	7	
70	0	0		1424	9014	7BAD3A00	7	
71	1	0		119C	F7E0	F75A7400	6	
72	1	0		1424	8CE0	88B3A000	3	
73	0	0		1AA9	A120	7FBD0000	0	
74	1	0		1AA9	8677	7FBD0000	0	
75	0	0		2516	D548	9F7A8800	5	51
76	1	0		2516	B032	9F7A8800	5	
77	0	0		299A	9458	517A2000	3	
78	0	0		2516	D57C	A2F44000	2	
79	1	0		2516	B066	A2F44000	2	
80	0	0		299A	9458	5E910000	0	

Table K.8 – Decoder test sequence (sheet 3 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
81	1	0		2516	D57C	BD22F000	7	78
82	0	0		299A	9458	32F3C000	5	
83	0	0		2516	D57C	65E78000	4	
84	0	0		2516	B066	65E78000	4	
85	0	0		2516	8B50	65E78000	4	
86	1	0		1EDF	CC74	CBCF0000	3	
87	1	0		2516	F6F8	F1D00000	0	
88	1	0		299A	9458	7FB95400	6	55
89	0	0		32B4	A668	53ED5000	4	
90	0	0		2E17	E768	A7DAA000	3	
91	1	0		2E17	B951	A7DAA000	3	
92	0	0		32B4	B85C	72828000	1	
93	1	0		32B4	85A8	72828000	1	
94	0	0		3C3D	CAD0	7E3B7E00	7	BF
95	1	0		3C3D	8E93	7E3B7E00	7	
96	0	0		415E	F0F4	AF95F800	5	
97	1	0		415E	AF96	AF95F800	5	
98	0	0	CE	4639	82BC	82BBF000	4	
99	1	0		415E	8C72	8C71E000	3	
100	0	0	CE	4639	82BC	82BBC000	2	
101	1	0		415E	8C72	8C718000	1	
102	0	0	CE	4639	82BC	82BB0000	0	
103	1	0		415E	8C72	8C71FE00	7	FF 00
104	0	0	CE	4639	82BC	82BBFC00	6	
105	1	0		415E	8C72	8C71F800	5	
106	0	0	CE	4639	82BC	82BBF000	4	
107	1	0		415E	8C72	8C71E000	3	
108	0	0	CE	4639	82BC	82BBC000	2	
109	1	0		415E	8C72	8C718000	1	
110	0	0	CE	4639	82BC	82BB0000	0	
111	1	0		415E	8C72	8C71F800	7	FC
112	0	0	CE	4639	82BC	82BBF000	6	
113	1	0		415E	8C72	8C71E000	5	
114	0	0	CE	4639	82BC	82BBC000	4	
115	1	0		415E	8C72	8C718000	3	
116	0	0	CE	4639	82BC	82BB0000	2	
117	1	0		415E	8C72	8C700000	1	
118	0	0	CE	4639	82BC	82B80000	0	
119	1	0		415E	8C72	8C6AA200	7	51
120	0	0	CE	4639	82BC	82AD4400	6	

Table K.8 – Decoder test sequence (sheet 4 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
121	1	0		415E	8C72	8C548800	5	
122	0	0	CE	4639	82BC	82811000	4	
123	1	0		415E	8C72	8BFC2000	3	
124	0	0	CE	4639	82BC	81D04000	2	
125	1	0		415E	8C72	8A9A8000	1	
126	0	0	CE	4639	82BC	7F0D0000	0	
127	1	0		415E	8C72	85150800	7	84
128	0	0	CE	4639	82BC	74021000	6	
129	1	0		415E	8C72	6EFE2000	5	
130	0	0	CE	4639	82BC	47D44000	4	
131	0	0		415E	8C72	16A28000	3	
132	0	0		3C3D	9628	2D450000	2	
133	0	0		375E	B3D6	5A8A0000	1	
134	0	0		32B4	F8F0	B5140000	0	
135	1	0		32B4	C63C	B5140000	0	
136	0	0		3C3D	CAD0	86331C00	6	C7
137	1	0		3C3D	8E93	86331C00	6	
138	1	0		415E	F0F4	CF747000	4	
139	0	0	CE	4639	82BC	3FBCE000	3	
140	0	0		415E	8C72	0673C000	2	
141	0	0		3C3D	9628	0CE78000	1	
142	0	0		375E	B3D6	19CF0000	0	
143	0	0		32B4	F8F0	339F9C00	7	CE
144	0	0		32B4	C63C	339F9C00	7	
145	0	0		32B4	9388	339F9C00	7	
146	0	0		2E17	C1A8	673F3800	6	
147	1	0		2E17	9391	673F3800	6	
148	0	0		32B4	B85C	0714E000	4	
149	0	0		32B4	85A8	0714E000	4	
150	0	0		2E17	A5E8	0E29C000	3	
151	0	0		299A	EFA2	1C538000	2	
152	0	0		299A	C608	1C538000	2	
153	0	0		299A	9C6E	1C538000	2	
154	0	0		2516	E5A8	38A70000	1	
155	0	0		2516	C092	38A70000	1	
156	0	0		2516	9B7C	38A70000	1	
157	0	0		1EDF	ECCC	714E0000	0	
158	0	0		1EDF	CDED	714E0000	0	
159	0	0		1EDF	AF0E	714E0000	0	
160	0	0		1EDF	902F	714E0000	0	

Table K.8 – Decoder test sequence (sheet 5 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
161	1	0		1AA9	E2A0	E29DF200	7	F9
162	1	0		2516	D548	D5379000	4	
163	1	0		299A	9458	94164000	2	
164	1	0		32B4	A668	A5610000	0	
165	1	0		3C3D	CAD0	C6B4E400	6	39
166	1	0		415E	F0F4	E0879000	4	
167	0	0	CE	4639	82BC	61E32000	3	
168	0	0		415E	8C72	4AC04000	2	
169	1	0		3C3D	9628	95808000	1	
170	1	0		415E	F0F4	EE560000	7	00
171	0	0	CE	4639	82BC	7D800000	6	
172	1	0		415E	8C72	81FA0000	5	
173	0	0	CE	4639	82BC	6DCC0000	4	
174	1	0		415E	8C72	62920000	3	
175	1	0	CE	4639	82BC	2EFC0000	2	
176	1	0		4B85	F20C	BBF00000	0	
177	1	0	CE	504F	970A	2AD25000	7	28
178	0	0	CE	5522	8D76	55A4A000	6	
179	0	0		504F	AA44	3AA14000	5	
180	1	0		4B85	B3EA	75428000	4	
181	1	0	CE	504F	970A	19BB0000	3	
182	1	0	CE	5522	8D76	33760000	2	
183	1	0		59EB	E150	CDD80000	0	
184	0	1		59EB	B3D6	8CE6FA00	7	7D
185	1	0		59EB	B3D6	65F7F400	6	
186	1	1		59EB	B3D6	1819E800	5	
187	1	1		5522	B3D6	3033D000	4	
188	1	1		504F	BD68	6067A000	3	
189	0	1		4B85	DA32	C0CF4000	2	
190	1	1	CE	504F	970A	64448000	1	
191	1	1		4B85	A09E	3B130000	0	
192	0	1		4639	AA32	76268C00	7	46
193	0	1	CE	4B85	8C72	245B1800	6	
194	1	1	CE	504F	81DA	48B63000	5	
195	1	1		4B85	A09E	2E566000	4	
196	1	1		4639	AA32	5CACC000	3	
197	0	1		415E	C7F2	B9598000	2	
198	1	1	CE	4639	82BC	658B0000	1	
199	0	1		415E	8C72	52100000	0	
200	0	1	CE	4639	82BC	0DF8E000	7	70

Table K.8 – Decoder test sequence (sheet 6 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
201	1	1		4B85	F20C	37E38000	5	
202	1	1		4B85	A687	37E38000	5	
203	1	1		4639	B604	6FC70000	4	
204	0	1		415E	DF96	DF8E0000	3	
205	1	1	CE	4639	82BC	82AC0000	2	
206	0	1		415E	8C72	8C520000	1	
207	1	1	CE	4639	82BC	827C0000	0	
208	0	1		415E	8C72	8BF31C00	7	8E
209	1	1	CE	4639	82BC	81BE3800	6	
210	0	1		415E	8C72	8A767000	5	
211	1	1	CE	4639	82BC	7EC4E000	4	
212	0	1		415E	8C72	8483C000	3	
213	1	1	CE	4639	82BC	72DF8000	2	
214	0	1		415E	8C72	6CB90000	1	
215	1	1	CE	4639	82BC	434A0000	0	
216	1	1		415E	8C72	0D8F9600	7	CB
217	1	1		3C3D	9628	1B1F2C00	6	
218	1	1		375E	B3D6	363E5800	5	
219	1	1		32B4	F8F0	6C7CB000	4	
220	1	1		32B4	C63C	6C7CB000	4	
221	0	1		32B4	9388	6C7CB000	4	
222	1	1		3C3D	CAD0	2EA2C000	2	
223	1	1		3C3D	8E93	2EA2C000	2	
224	1	1		375E	A4AC	5D458000	1	
225	0	1		32B4	DA9C	BA8B0000	0	
226	1	1		3C3D	CAD0	4A8F0000	6	C0
227	1	1		3C3D	8E93	4A8F0000	6	
228	0	1		375E	A4AC	951E0000	5	
229	1	1		3C3D	DD78	9F400000	3	
230	0	1		3C3D	A13B	9F400000	3	
231	0	1		415E	F0F4	E9080000	1	
232	1	1	CE	4639	82BC	72E40000	0	
233	0	1		415E	8C72	6CC3EC00	7	F6
234	1	1	CE	4639	82BC	435FD800	6	
235	1	1		415E	8C72	0DB9B000	5	
236	1	1		3C3D	9628	1B736000	4	
237	1	1		375E	B3D6	36E6C000	3	
238	1	1		32B4	F8F0	6DCD8000	2	
239	1	1		32B4	C63C	6DCD8000	2	
240	0	1		32B4	9388	6DCD8000	2	

Table K.8 – Decoder test sequence (sheet 7 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
241	1	1		3C3D	CAD0	33E60000	0	
242	1	1		3C3D	8E93	33E60000	0	
Marker detected: zero byte fed to decoder								
243	1	1		375E	A4AC	67CC0000	7	
244	0	1		32B4	DA9C	CF980000	6	
245	0	1		3C3D	CAD0	9EC00000	4	
246	1	1		415E	F0F4	40B40000	2	
247	1	1		415E	AF96	40B40000	2	
248	1	1		3C3D	DC70	81680000	1	
249	0	1		3C3D	A033	81680000	1	
Marker detected: zero byte fed to decoder								
250	1	1		415E	F0F4	75C80000	7	
251	0	1		415E	AF96	75C80000	7	
252	0	1	CE	4639	82BC	0F200000	6	
253	1	1		4B85	F20C	3C800000	4	
254	1	1		4B85	A687	3C800000	4	
255	0	1		4639	B604	79000000	3	
256	0	1	CE	4B85	8C72	126A0000	2	

K.5 Low-pass downsampling filters for hierarchical coding

In this section simple examples are given of downsampling filters which are compatible with the upsampling filter defined in J.1.1.2.

Figure K.5 shows the weighting of neighbouring samples for simple one-dimensional horizontal and vertical low-pass filters. The output of the filter must be normalized by the sum of the neighbourhood weights.

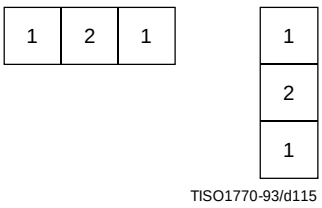


Figure K.5 – Low-pass filter example

The centre sample in Figure K.5 should be aligned with the left column or top line of the high resolution image when calculating the left column or top line of the low resolution image. Sample values which are situated outside of the image boundary are replicated from the sample values at the boundary to provide missing edge values.

If the image being downsampled has an odd width or length, the odd dimension is increased by 1 by sample replication on the right edge or bottom line before downsampling.

K.6 Domain of applicability of DCT and spatial coding techniques

The DCT coder is intended for lossy coding in a range from quite visible loss to distortion well below the threshold for visibility. However in general, DCT-based processes cannot be used for true lossless coding.

The lossless coder is intended for completely lossless coding. The lossless coding process is significantly less effective than the DCT-based processes for distortions near and above the threshold of visibility.

The point transform of the input to the lossless coder permits a very restricted form of lossy coding with the "lossless" coder. (The coder is still lossless after the input point transform.) Since the DCT is intended for lossy coding, there may be some confusion about when this alternative lossy technique should be used.

Lossless coding with a point transformed input is intended for applications which cannot be addressed by DCT coding techniques. Among these are

- true lossless coding to a specified precision;
- lossy coding with precisely defined error bounds;
- hierarchical progression to a truly lossless final stage.

If lossless coding with a point transformed input is used in applications which can be met effectively by DCT coding, the results will be significantly less satisfactory. For example, distortion in the form of visible contours usually appears when precision of the luminance component is reduced to about six bits. For normal image data, this occurs at bit rates well above those for which the DCT gives outputs which are visually indistinguishable from the source.

K.7 Domain of applicability of the progressive coding modes of operation

Two very different progressive coding modes of operation have been defined, progressive coding of the DCT coefficients and hierarchical progression. Progressive coding of the DCT coefficients has two complementary procedures, spectral selection and successive approximation. Because of this diversity of choices, there may be some confusion as to which method of progression to use for a given application.

K.7.1 Progressive coding of the DCT

In progressive coding of the DCT coefficients two complementary procedures are defined for decomposing the 8×8 DCT coefficient array, spectral selection and successive approximation. Spectral selection partitions zig-zag array of DCT coefficients into "bands", one band being coded in each scan. Successive approximation codes the coefficients with reduced precision in the first scan; in each subsequent scan the precision is increased by one bit.

A single forward DCT is calculated for these procedures. When all coefficients are coded to full precision, the DCT is the same as in the sequential mode. Therefore, like the sequential DCT coding, progressive coding of DCT coefficients is intended for applications which need very good compression for a given level of visual distortion.

The simplest progressive coding technique is spectral selection; indeed, because of this simplicity, some applications may choose – despite the limited progression that can be achieved – to use only spectral selection. Note, however, that the absence of high frequency bands typically leads – for a given bit rate – to a significantly lower image quality in the intermediate stages than can be achieved with the more general progressions. The net coding efficiency at the completion of the final stage is typically comparable to or slightly less than that achieved with the sequential DCT.

A much more flexible progressive system is attained at some increase in complexity when successive approximation is added to the spectral selection progression. For a given bit rate, this system typically provides significantly better image quality than spectral selection alone. The net coding efficiency at the completion of the final stage is typically comparable to or slightly better than that achieved with the sequential DCT.

K.7.2 Hierarchical progression

Hierarchical progression permits a sequence of outputs of increasing spatial resolution, and also allows refinement of image quality at a given spatial resolution. Both DCT and spatial versions of the hierarchical progression are allowed, and progressive coding of DCT coefficients may be used in a frame of the DCT hierarchical progression.

The DCT hierarchical progression is intended for applications which need very good compression for a given level of visual distortion; the spatial hierarchical progression is intended for applications which need a simple progression with a truly lossless final stage. Figure K.6 illustrates examples of these two basic hierarchical progressions.

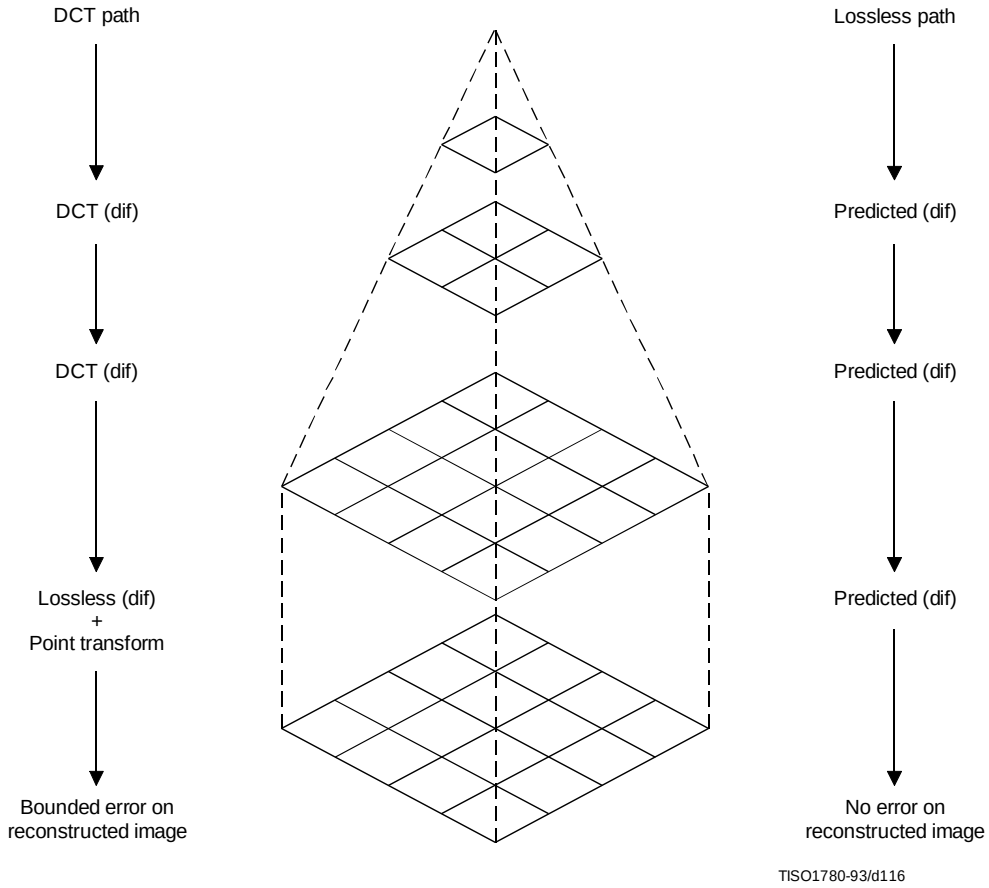


Figure K.6 – Sketch of the basic operations of the hierarchical mode

K.7.2.1 DCT Hierarchical progression

If a DCT hierarchical progression uses reduced spatial resolution, the early stages of the progression can have better image quality for a given bit rate than the early stages of non-hierarchical progressive coding of the DCT coefficients. However, at the point where the distortion between source and output becomes indistinguishable, the coding efficiency achieved with a DCT hierarchical progression is typically significantly lower than the coding efficiency achieved with a non-hierarchical progressive coding of the DCT coefficients.

While the hierarchical DCT progression is intended for lossy progressive coding, a final spatial differential coding stage can be used. When this final stage is used, the output can be almost lossless, limited only by the difference between the encoder and decoder IDCT implementations. Since IDCT implementations can differ significantly, truly lossless coding after a DCT hierarchical progression cannot be guaranteed. An important alternative, therefore, is to use the input point transform of the final lossless differential coding stage to reduce the precision of the differential input. This allows a bounding of the difference between source and output at a significantly lower cost in coded bits than coding of the full precision spatial difference would require.

K.7.2.2 Spatial hierarchical progression

If lossless progression is required, a very simple hierarchical progression may be used in which the spatial lossless coder with point transformed input is used as a first stage. This first stage is followed by one or more spatial differential coding stages. The first stage should be nearly lossless, such that the low order bits which are truncated by the point transform are essentially random – otherwise the compression efficiency will be degraded relative to non-progressive lossless coding.

K.8 Suppression of block-to-block discontinuities in decoded images

A simple technique is available for suppressing the block-to-block discontinuities which can occur in images compressed by DCT techniques.

The first few (five in this example) low frequency DCT coefficients are predicted from the nine DC values of the block and the eight nearest-neighbour blocks, and the predicted values are used to suppress blocking artifacts in smooth areas of the image.

The prediction equations for the first five AC coefficients in the zig-zag sequence are obtained as follows:

K.8.1 AC prediction

The sample field in a 3 by 3 array of blocks (each block containing an 8×8 array of samples) is modeled by a two-dimensional second degree polynomial of the form:

$$P(x,y) = A1(x^2y^2) + A2(x^2y) + A3(xy^2) + A4(x^2) + A5(xy) + A6(y^2) + A7(x) + A8(y) + A9$$

The nine coefficients A1 through A9 are uniquely determined by imposing the constraint that the mean of P(x,y) over each of the nine blocks must yield the correct DC-values.

Applying the DCT to the quadratic field predicting the samples in the central block gives a prediction of the low frequency AC coefficients depicted in Figure K.7.

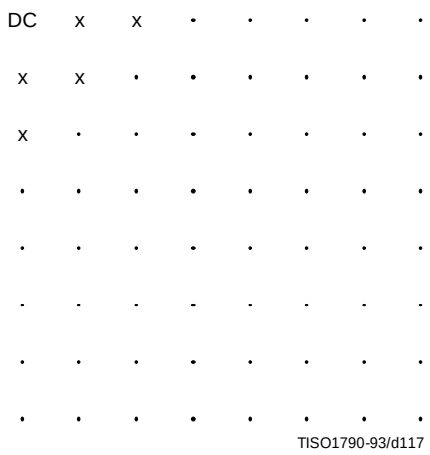


Figure K.7 – DCT array positions of predicted AC coefficients

The prediction equations derived in this manner are as follows:

For the two dimensional array of DC values shown

DC_1 DC_2 DC_3
DC_4 DC_5 DC_6
DC_7 DC_8 DC_9

The unquantized prediction equations are

$AC_{01} = 1{,}13885\ (DC_4 - DC_6)$
$AC_{10} = 1{,}13885\ (DC_2 - DC_8)$
$AC_{20} = 0{,}27881\ (DC_2 + DC_8 - 2 \times DC_5)$
$AC_{11} = 0{,}16213\ ((DC_1 - DC_3) - (DC_7 - DC_9))$
$AC_{02} = 0{,}27881\ (DC_4 + DC_6 - 2 \times DC_5)$

The scaling of the predicted AC coefficients is consistent with the DCT normalization defined in A.3.3.

K.8.2 Quantized AC prediction

The prediction equations can be mapped to a form which uses quantized values of the DC coefficients and which computes quantized AC coefficients using integer arithmetic. The quantized DC coefficients need to be scaled, however, such that the predicted coefficients have fractional bit precision.

First, the prediction equation coefficients are scaled by 32 and rounded to the nearest integer. Thus,

$$1{,}13885 \times 32 = 36$$

$$0{,}27881 \times 32 = 9$$

$$0{,}16213 \times 32 = 5$$

The multiplicative factors are then scaled by the ratio of the DC and AC quantization factors and rounded appropriately. The normalization defined for the DCT introduces another factor of 8 in the unquantized DC values. Therefore, in terms of the quantized DC values, the predicted quantized AC coefficients are given by the equations below. Note that if (for example) the DC values are scaled by a factor of 4, the AC predictions will have 2 fractional bits of precision relative to the quantized DCT coefficients.

$$QAC_{01} = ((R_d \times Q_{01}) + (36 \times Q_{00} \times (QDC_4 - QDC_6)))/(256 \times Q_{01})$$
$$QAC_{10} = ((R_d \times Q_{10}) + (36 \times Q_{00} \times (QDC_2 - QDC_8)))/(256 \times Q_{10})$$
$$QAC_{20} = ((R_d \times Q_{20}) + (9 \times Q_{00} \times (QDC_2 + QDC_8 - 2 \times QDC_5)))/(256 \times Q_{20})$$
$$QAC_{11} = ((R_d \times Q_{11}) + (5 \times Q_{00} \times ((QDC_1 - QDC_3) - (QDC_7 - QDC_9))))/(256 \times Q_{11})$$
$$QAC_{02} = ((R_d \times Q_{02}) + (9 \times Q_{00} \times (QDC_4 + QDC_6 - 2 \times QDC_5)))/(256 \times Q_{02})$$

where QDC_x and QAC_{xy} are the quantized and scaled DC and AC coefficient values. The constant Rd is added to get a correct rounding in the division. Rd is 128 for positive numerators, and –128 for negative numerators.

Predicted values should not override coded values. Therefore, predicted values for coefficients which are already non-zero should be set to zero. Predictions should be clamped if they exceed a value which would be quantized to a non-zero value for the current precision in the successive approximation.

K.9 Modification of dequantization to improve displayed image quality

For a progression where the first stage successive approximation bit, Al, is set to 3, uniform quantization of the DCT gives the following quantization and dequantization levels for a sequence of successive approximation scans, as shown in Figure K.8:

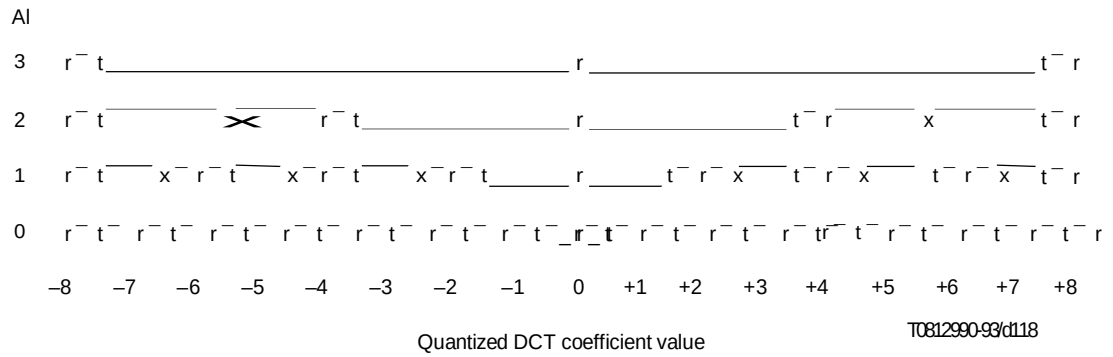


Figure K.8 – Illustration of two reconstruction strategies

The column to the left labelled “Al” gives the bit position specified in the scan header. The quantized DCT coefficient magnitudes are therefore divided by 2^{Al} during that scan.

Referring to the final scan (Al = 0), the points marked with "t" are the threshold values, while the points marked with "r" are the reconstruction values. The unquantized output is obtained by multiplying the horizontal scale in Figure K.8 by the quantization value.

The quantization interval for a coefficient value of zero is indicated by the depressed interval of the line. As the bit position Al is increased, a "fat zero" quantization interval develops around the zero DCT coefficient value. In the limit where the scaling factor is very large, the zero interval is twice as large as the rest of the quantization intervals.

Two different reconstruction strategies are shown. The points marked "r" are the reconstruction obtained using the normal rounding rules for the DCT for the complete full precision output. This rule seems to give better image quality when high bandwidth displays are used. The points marked "x" are an alternative reconstruction which tends to give better images on lower bandwidth displays. "x" and "r" are the same for slice 0. The system designer must determine which strategy is best for the display system being used.

K.10 Example of point transform

The difference between the arithmetic-shift-right by Pt and divide by 2^{Pt} can be seen from the following:

After the level shift the DC has values from +127 to –128. Consider values near zero (after the level shift), and the case where Pt = 1:

Before level shift	Before point transform	After divide by 2	After shift-right-arithmetic 1
131	+3	+1	+1
130	+2	+1	+1
129	+1	0	0
128	0	0	0
127	–1	0	–1
126	–2	–1	–1
125	–3	–1	–2
124	–4	–2	–2
123	–5	–2	–3

The key difference is in the truncation of precision. The divide truncates the magnitude; the arithmetic shift truncates the LSB. With a divide by 2 we would get non-uniform quantization of the DC values; therefore we use the shift-right-arithmetic operation.

For positive values, the divide by 2 and the shift-right-arithmetic by 1 operations are the same. Therefore, the shift-right-arithmetic by 1 operation effectively is a divide by 2 when the point transform is done before the level shift.

Annex L

Patents

(This annex does not form an integral part of this Recommendation | International Standard)

L.1 Introductory remarks

The user's attention is called to the possibility that – for some of the coding processes specified in Annexes F, G, H, and J – compliance with this Specification may require use of an invention covered by patent rights.

By publication of this Specification, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. However, for each patent listed in this annex, the patent holder has filed with the Information Technology Task Force (ITTF) and the Telecommunication Standardization Bureau (TSB) a statement of willingness to grant a license under these rights on reasonable and non-discriminatory terms and conditions to applicants desiring to obtain such a license.

The criteria for including patents in this annex are:

a) the patent has been identified by someone who is familiar with the technical fields relevant to this Specification, and who believes use of the invention covered by the patent is *required* for implementation of one or more of the coding processes specified in Annexes F, G, H, or J;

b) the patent-holder has written a letter to the ITTF and TSB, stating willingness to grant a license to an unlimited number of applicants throughout the world under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

This list of patents shall be updated, if necessary, upon publication of any revisions to the Recommendation | International Standard.

L.2 List of patents

The following patents may be required for implementation of any one of the processes specified in Annexes F, G, H, and J which uses arithmetic coding:

US 4,633,490, December 30, 1986, IBM, MITCHELL (J.L.) and GOERTZEL (G.): *Symmetrical Adaptive Data Compression/Decompression System.*

US 4,652,856, February 4, 1986, IBM, MOHIUDDIN (K.M.) and RISSANEN (J.J.): *A Multiplication-free Multi-Alphabet Arithmetic Code.*

US 4,369,463, January 18, 1983, IBM, ANASTASSIOU (D.) and MITCHELL (J.L.): *Grey Scale Image Compression with Code Words a Function of Image History.*

US 4,749,983, June 7, 1988, IBM, LANGDON (G.): *Compression of Multilevel Signals.*

US 4,935,882, June 19, 1990, IBM, PENNEBAKER (W.B.) and MITCHELL (J.L.): *Probability Adaptation for Arithmetic Coders*.

US 4,905,297, February 27, 1990, IBM, LANGDON (G.G.), Jr., MITCHELL (J.L.), PENNEBAKER (W.B.), and RISSANEN (J.J.): *Arithmetic Coding Encoder and Decoder System*.

US 4,973,961, November 27, 1990, AT&T, CHAMZAS (C.), DUTTWEILER (D.L.): *Method and Apparatus for Carry-over Control in Arithmetic Entropy Coding.*

US 5,025,258, June 18, 1991, AT&T, DUTTWEILER (D.L): *Adaptive Probability Estimator for Entropy Encoding/Decoding*.

US 5,099,440, March 24, 1992, IBM, PENNEBAKER (W.B.) and MITCHELL (J.L.): *Probability Adaptation for Arithmetic Coders*.

Japanese Patent Application 2-46275, February 26, 1990, MEL ONO (F.), KIMURA (T.), YOSHIDA (M.), and KINO (S.): *Coding System*.

The following patent may be required for implementation of any one of the hierarchical processes specified in Annex H when used with a lossless final frame:

US 4,665,436, May 12, 1987, EI OSBORNE (J.A.) and SEIFFERT (C.): *Narrow Bandwidth Signal Transmission*.

No other patents required for implementation of any of the other processes specified in Annexes F, G, H, or J had been identified at the time of publication of this Specification.

L.3 Contact addresses for patent information

Director, Telecommunication Standardization Bureau (formerly CCITT)
International Telecommunication Union
Place des Nations
CH-1211 Genève 20, Switzerland
Tel. +41 (22) 730 5111
Fax: +41 (22) 730 5853

Information Technology Task Force
International Organization for Standardization
1, rue de Varembé
CH-1211 Genève 20, Switzerland
Tel: +41 (22) 734 0150
Fax: +41 (22) 733 3843

Program Manager, Licensing
Intellectual Property and Licensing Services
IBM Corporation
208 Harbor Drive
P.O. Box 10501
Stamford, Connecticut 08904-2501, USA
Tel: +1 (203) 973 7935
Fax: +1 (203) 973 7981 or +1 (203) 973 7982

Mitsubishi Electric Corp.
Intellectual Property License Department
1-2-3 Morunouchi, Chiyoda-ku
Tokyo 100, Japan
Tel: +81 (3) 3218 3465
Fax: +81 (3) 3215 3842

AT&T Intellectual Property Division Manager
Room 3A21
10 Independence Blvd.
Warren, NJ 07059, USA
Tel: +1 (908) 580 5392
Fax: +1 (908) 580 6355

Senior General Manager
Corporate Intellectual Property and Legal Headquarters
Canon Inc.
30-2 Shimomaruko 3-chome
Ohta-ku Tokyo 146 Japan
Tel: +81 (3) 3758 2111
Fax: +81 (3) 3756 0947

Chief Executive Officer
Electronic Imagery, Inc.
1100 Park Central Boulevard South
Suite 3400
Pompano Beach, FL 33064, USA
Tel: +1 (305) 968 7100
Fax: +1 (305) 968 7319

Annex M

Bibliography

(This annex does not form an integral part of this Recommendation | International Standard)

M.1 General references

LEGER (A.), OMACHI (T.), and WALLACE (G.K.): JPEG Still Picture Compression Algorithm, *Optical Engineering*, Vol. 30, No. 7, pp. 947-954, 1991.

RABBANI (M.) and JONES (P.): Digital Image Compression Techniques, *Tutorial Texts in Optical Engineering*, Vol. TT7, SPIE Press, 1991.

HUDSON (G.), YASUDA (H.) and SEBESTYEN (I.): The International Standardization of a Still Picture Compression Technique, *Proc. of IEEE Global Telecommunications Conference*, pp. 1016-1021, 1988.

LEGER (A.), MITCHELL (J.) and YAMAZAKI (Y.): Still Picture Compression Algorithm Evaluated for International Standardization, *Proc. of the IEEE Global Telecommunications Conference*, pp. 1028-1032, 1988.

WALLACE (G.), VIVIAN (R.) and POULSEN (H.): Subjective Testing Results for Still Picture Compression Algorithms for International Standardization, *Proc. of the IEEE Global Telecommunications Conference*, pp. 1022-1027, 1988.

MITCHELL (J.L.) and PENNEBAKER (W.B.): Evolving JPEG Colour Data Compression Standard, *Standards for Electronic Imaging Systems*, M. Nier, M.E. Courtot, Editors, SPIE, Vol. CR37, pp. 68-97, 1991.

WALLACE (G.K.): The JPEG Still Picture Compression Standard, *Communications of the ACM*, Vol. 34, No. 4, pp. 31-44, 1991.

NETRAVALI (A.N.) and HASKELL (B.G.): *Digital Pictures: Representation and Compression*, Plenum Press, New York 1988.

PENNEBAKER (W.B.) and MITCHELL (J.L.): *JPEG: Still Image Data Compression Standard*, Van Nostrand Reinhold, New York 1993.

M.2 DCT references

CHEN (W.), SMITH (C.H.) and FRALICK (S.C.): A Fast Computational Algorithm for the Discrete Cosine Transform, *IEEE Trans. on Communications*, Vol. COM-25, pp. 1004-1009, 1977.

AHMED (N.), NATARAJAN (T.) and RAO (K.R.): Discrete Cosine Transform, *IEEE Trans. on Computers*, Vol. C-23, pp. 90-93, 1974.

NARASINHA (N.J.) and PETERSON (A.M.): On the Computation of the Discrete Cosine Transform, *IEEE Trans. on Communications*, Vol. COM-26, No. 6, pp. 966-968, 1978.

DUHAMEL (P.) and GUILLEMOT (C.): Polynomial Transform Computation of the 2-D DCT, *Proc. IEEE ICASSP-90*, pp. 1515-1518, Albuquerque, New Mexico 1990.

FEIG (E.): A Fast Scaled DCT Algorithm, in *Image Processing Algorithms and Techniques*, Proc. SPIE, Vol. 1244, K.S. Pennington and R. J. Moorhead II, Editors, pp. 2-13, Santa Clara, California, 1990.

HOU (H.S.): A Fast Recursive Algorithm for Computing the Discrete Cosine Transform, *IEEE Trans. Acoust. Speech and Signal Processing*, Vol. ASSP-35, No. 10, pp. 1455-1461.

LEE (B.G.): A New Algorithm to Compute the Discrete Cosine Transform, *IEEE Trans. on Acoust., Speech and Signal Processing*, Vol. ASSP-32, No. 6, pp. 1243-1245, 1984.

LINZER (E.N.) and FEIG (E.): New DCT and Scaled DCT Algorithms for Fused Multiply/Add Architectures, *Proc. IEEE ICASSP-91*, pp. 2201-2204, Toronto, Canada, 1991.

VETTERLI (M.) and NUSSBAUMER (H.J.): Simple FFT and DCT Algorithms with Reduced Number of Operations, *Signal Processing*, 1984.

VETTERLI (M.): Fast 2-D Discrete Cosine Transform, *Proc. IEEE ICASSP-85*, pp. 1538-1541, Tampa, Florida, 1985.

ARAI (Y.), AGUI (T.), and NAKAJIMA (M.): A Fast DCT-SQ Scheme for Images, *Trans. of IEICE*, Vol. E.71, No. 11, pp. 1095-1097, 1988.

SUEHIRO (N.) and HATORI (M.): Fast Algorithms for the DFT and other Sinusoidal Transforms, *IEEE Trans. on Acoust., Speech and Signal Processing*, Vol ASSP-34, No. 3, pp. 642-644, 1986.

M.3 Quantization and human visual model references

CHEN (W.H.) and PRATT (W.K.): Scene adaptive coder, *IEEE Trans. on Communications*, Vol. COM-32, pp. 225-232, 1984.

GRANRATH (D.J.): The role of human visual models in image processing, *Proceedings of the IEEE*, Vol. 67, pp. 552-561, 1981.

LOHSCHELLER (H.): Vision adapted progressive image transmission, *Proceedings of EUSIPCO*, Vol. 83, pp. 191-194, 1983.

LOHSCHELLER (H.) and FRANKE (U.): Colour picture coding – Algorithm optimization and technical realization, *Frequenze*, Vol. 41, pp. 291-299, 1987.

LOHSCHELLER (H.): A subjectively adapted image communication system, *IEEE Trans. on Communications*, Vol. COM-32, pp. 1316-1322, 1984.

PETERSON (H.A.) *et al*: Quantization of colour image components in the DCT domain, *SPIE/IS&T 1991 Symposium on Electronic Imaging Science and Technology*, 1991.

M.4 Arithmetic coding references

LANGDON (G.): An Introduction to Arithmetic Coding, *IBM J. Res. Develop.*, Vol. 28, pp. 135-149, 1984.

PENNEBAKER (W.B.), MITCHELL (J.L.), LANGDON (G.) Jr., and ARPS (R.B.): An Overview of the Basic Principles of the Q-Coder Binary Arithmetic Coder, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 717-726, 1988.

MITCHELL (J.L.) and PENNEBAKER (W.B.): Optimal Hardware and Software Arithmetic Coding Procedures for the Q-Coder Binary Arithmetic Coder, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 727-736, 1988.

PENNEBAKER (W.B.) and MITCHELL (J.L.): Probability Estimation for the Q-Coder, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 737-752, 1988.

MITCHELL (J.L.) and PENNEBAKER (W.B.): Software Implementations of the Q-Coder, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 753-774, 1988.

ARPS (R.B.), TRUONG (T.K.), LU (D.J.), PASCO (R.C.) and FRIEDMAN (T.D.): A Multi-Purpose VLSI Chip for Adaptive Data Compression of Bilevel Images, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 775-795, 1988.

ONO (F.), YOSHIDA (M.), KIMURA (T.) and KINO (S.): Subtraction-type Arithmetic Coding with MPS/LPS Conditional Exchange, *Annual Spring Conference of IECED*, Japan, D-288, 1990.

DUTTWEILER (D.) and CHAMZAS (C.): Probability Estimation in Arithmetic and Adaptive-Huffman Entropy Coders, submitted to *IEEE Trans. on Image Processing*.

JONES (C.B.): An Efficient Coding System for Long Source Sequences, *IEEE Trans. Inf. Theory*,Vol. IT-27, pp. 280-291, 1981.

LANGDON (G.): Method for Carry-over Control in a Fifo Arithmetic Code String, *IBM Technical Disclosure Bulletin*, Vol. 23, No.1, pp. 310-312, 1980.

M.5 Huffman coding references

HUFFMAN (D.A.): A Method for the Construction of Minimum Redundancy codes, *Proc. IRE*, Vol. 40, pp. 1098-1101, 1952.

Image Compression and the Discrete Cosine Transform

Ken Cabeen and Peter Gent

Math 45
College of the Redwoods

Abstract. The mathematical equations of the DCT and its uses with image compression are explained.

Introduction

As our use of and reliance on computers continues to grow, so too does our need for efficient ways of storing large amounts of data. For example, someone with a web page or online catalog – that uses dozens or perhaps hundreds of images – will more than likely need to use some form of image compression to store those images. This is because the amount of space required to hold unadulterated images can be prohibitively large in terms of cost. Fortunately, there are several methods of image compression available today. These fall into two general categories: lossless and lossy image compression. The JPEG process is a widely used form of lossy image compression that centers around the Discrete Cosine Transform. The DCT works by separating images into parts of differing frequencies. During a step called quantization, where part of compression actually occurs, the less important frequencies are discarded, hence the use of the term ”lossy.” Then, only the most important frequencies that remain are used retrieve the image in the decompression process. As a result, reconstructed images contain some distortion; but as we shall soon see, these levels of distortion can be adjusted during the compression stage. The JPEG method is used for both color and black-and-white images, but the focus of this article will be on compression of the latter.

The Process

The following is a general overview of the JPEG process. Later, we will take the reader through a detailed tour of JPEG's method so that a more comprehensive understanding of the process may be acquired.

1. The image is broken into 8x8 blocks of pixels.
2. Working from left to right, top to bottom, the DCT is applied to each block.
3. Each block is compressed through quantization.
4. The array of compressed blocks that constitute the image is stored in a drastically reduced amount of space.
5. When desired, the image is reconstructed through decompression, a process that uses the Inverse Discrete Cosine Transform (IDCT).

The DCT Equation

The DCT equation (Eq. 1) computes the i,jth entry of the DCT of an image.

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N} \right] \cos\left[\frac{(2y+1)j\pi}{2N} \right] \qquad 1$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{cases} \qquad 2$$

$p(x,y)$ is the x,yth element of the image represented by the matrix p. N is the size of the block that the DCT is done on. The equation calculates one entry (i,jth) of the transformed image from the pixel values of the original image matrix. For the standard 8x8 block that JPEG compression uses, N equals 8 and x and y range from 0 to 7. Therefore $D(i,j)$ would be as in Equation (3).

$$D(i,j) = \frac{1}{4}C(i)C(j)\sum_{x=0}^{7}\sum_{y=0}^{7}p(x,y)\cos\left[\frac{(2x+1)i\pi}{16}\right]\cos\left[\frac{(2y+1)j\pi}{16}\right] \qquad 3$$

Because the DCT uses cosine functions, the resulting matrix depends on the horizontal, diagonal, and vertical frequencies. Therefore an image black with a lot of change in frequency has a very random looking resulting matrix, while an image matrix of just one color, has a resulting matrix of a large value for the first element and zeroes for the other elements.

The DCT Matrix

To get the matrix form of Equation (1), we will use the following equation

$$T_{i,j} = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } i = 0 \\ \sqrt{\frac{2}{N}}\cos\left[\frac{(2j+1)i\pi}{2N}\right] & \text{if } i > 0 \end{cases} \qquad 4$$

For an 8x8 block it results in this matrix:

$$T = \begin{bmatrix} .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 \\ .4904 & .4157 & .2778 & .0975 & -.0975 & -.2778 & -.4157 & -.4904 \\ .4619 & .1913 & -.1913 & -.4619 & -.4619 & -.1913 & .1913 & .4619 \\ .4157 & -.0975 & -.4904 & -.2778 & .2778 & .4904 & .0975 & -.4157 \\ .3536 & -.3536 & -.3536 & .3536 & .3536 & -.3536 & -.3536 & .3536 \\ .2778 & -.4904 & .0975 & .4157 & -.4157 & -.0975 & .4904 & -.2778 \\ .1913 & -.4619 & .4619 & -.1913 & -.1913 & .4619 & -.4619 & .1913 \\ .0975 & -.2778 & .4157 & -.4904 & .4904 & -.4157 & .2778 & -.0975 \end{bmatrix}$$

The first row ($i = 1$) of the matrix has all the entries equal to $1/\sqrt{8}$ as expected from Equation (4).

The columns of T form an orthonormal set, so T is an orthogonal matrix. When doing the inverse DCT the orthogonality of T is important, as the inverse of T is T' which is easy to calculate.

Doing the DCT on an 8x8 Block

Before we begin, it should be noted that the pixel values of a black-and-white image range from 0 to 255 in steps of 1, where pure black is represented by 0, and pure white by 255. Thus it can be seen how a photo, illustration, etc. can be accurately represented by these 256 shades of gray.

Since an image comprises hundreds or even thousands of 8x8 blocks of pixels, the following description of what happens to one 8x8 block is a microcosm of the JPEG process;

what is done to one block of image pixels is done to all of them, in the order earlier specified.

Now, let's start with a block of image-pixel values. This particular block was chosen from the very upper- left-hand corner of an image.

$$Original = \begin{bmatrix} 154 & 123 & 123 & 123 & 123 & 123 & 123 & 136 \\ 192 & 180 & 136 & 154 & 154 & 154 & 136 & 110 \\ 254 & 198 & 154 & 154 & 180 & 154 & 123 & 123 \\ 239 & 180 & 136 & 180 & 180 & 166 & 123 & 123 \\ 180 & 154 & 136 & 167 & 166 & 149 & 136 & 136 \\ 128 & 136 & 123 & 136 & 154 & 180 & 198 & 154 \\ 123 & 105 & 110 & 149 & 136 & 136 & 180 & 166 \\ 110 & 136 & 123 & 123 & 123 & 136 & 154 & 136 \end{bmatrix}$$

Because the DCT is designed to work on pixel values ranging from -128 to 127, the original block is "leveled off" by subtracting 128 from each entry. This results in the following matrix.

$$M = \begin{bmatrix} 26 & -5 & -5 & -5 & -5 & -5 & -5 & 8 \\ 64 & 52 & 8 & 26 & 26 & 26 & 8 & -18 \\ 126 & 70 & 26 & 26 & 52 & 26 & -5 & -5 \\ 111 & 52 & 8 & 52 & 52 & 38 & -5 & -5 \\ 52 & 26 & 8 & 39 & 38 & 21 & 8 & 8 \\ 0 & 8 & -5 & 8 & 26 & 52 & 70 & 26 \\ -5 & -23 & -18 & 21 & 8 & 8 & 52 & 38 \\ -18 & 8 & -5 & -5 & -5 & 8 & 26 & 8 \end{bmatrix}$$

We are now ready to perform the Discrete Cosine Transform, which is accomplished by matrix multiplication.

$$D = TMT' \qquad 5$$

In Equation (5) matrix M is first multiplied on the left by the DCT matrix T from the previous section; this transforms the rows. The columns are then transformed by multiplying on the right by the transpose of the DCT matrix. This yields the following matrix.

$$D = \begin{bmatrix} 162.3 & 40.6 & 20.0 & 72.3 & 30.3 & 12.5 & -19.7 & -11.5 \\ 30.5 & 108.4 & 10.5 & 32.3 & 27.7 & -15.5 & 18.4 & -2.0 \\ -94.1 & -60.1 & 12.3 & -43.4 & -31.3 & 6.1 & -3.3 & 7.1 \\ -38.6 & -83.4 & -5.4 & -22.2 & -13.5 & 15.5 & -1.3 & 3.5 \\ -31.3 & 17.9 & -5.5 & -12.4 & 14.3 & -6.0 & 11.5 & -6.0 \\ -0.9 & -11.8 & 12.8 & 0.2 & 28.1 & 12.6 & 8.4 & 2.9 \\ 4.6 & -2.4 & 12.2 & 6.6 & -18.7 & -12.8 & 7.7 & 12.0 \\ -10.0 & 11.2 & 7.8 & -16.3 & 21.5 & 0.0 & 5.9 & 10.7 \end{bmatrix}$$

This block matrix now consists of 64 DCT coefficients, c_{ij}, where i and j range from 0 to 7. The top-left coefficient, c_{00}, correlates to the low frequencies of the original image block. As we move away from c_{00} in all directions, the DCT coefficients correlate to higher and higher frequencies of the image block, where c_{77} corresponds to the highest frequency. It is important to note that the human eye is most sensitive to low frequencies, and results from the quantization step will reflect this fact.

Quantization

Our 8x8 block of DCT coefficients is now ready for compression by quantization. A remarkable and highly useful feature of the JPEG process is that in this step, varying levels of image compression and quality are obtainable through selection of specific quantization matrices. This enables the user to decide on quality levels ranging from 1 to 100, where 1 gives the poorest image quality and highest compression, while 100 gives the best quality and lowest compression. As a result, the quality/compression ratio can be tailored to suit different needs.

Subjective experiments involving the human visual system have resulted in the JPEG standard quantization matrix. With a quality level of 50, this matrix renders both high compression and excellent decompressed image quality.

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

If, however, another level of quality and compression is desired, scalar multiples of the JPEG standard quantization matrix may be used. For a quality level greater than 50 (less compression, higher image quality), the standard quantization matrix is multiplied by (100-quality level)/50. For a quality level less than 50 (more compression, lower image quality), the standard quantization matrix is multiplied by 50/quality level. The scaled

quantization matrix is then rounded and clipped to have positive integer values ranging from 1 to 255. For example, the following quantization matrices yield quality levels of 10 and 90.

$$Q_{10} = \begin{bmatrix} 80 & 60 & 50 & 80 & 120 & 200 & 255 & 255 \\ 55 & 60 & 70 & 95 & 130 & 255 & 255 & 255 \\ 70 & 65 & 80 & 120 & 200 & 255 & 255 & 255 \\ 70 & 85 & 110 & 145 & 255 & 255 & 255 & 255 \\ 90 & 110 & 185 & 255 & 255 & 255 & 255 & 255 \\ 120 & 175 & 255 & 255 & 255 & 255 & 255 & 255 \\ 245 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \end{bmatrix}$$

$$Q_{90} = \begin{bmatrix} 3 & 2 & 2 & 3 & 5 & 8 & 10 & 12 \\ 2 & 2 & 3 & 4 & 5 & 12 & 12 & 11 \\ 3 & 3 & 3 & 5 & 8 & 11 & 14 & 11 \\ 3 & 3 & 4 & 6 & 10 & 17 & 16 & 12 \\ 4 & 4 & 7 & 11 & 14 & 22 & 21 & 15 \\ 5 & 7 & 11 & 13 & 16 & 12 & 23 & 18 \\ 10 & 13 & 16 & 17 & 21 & 24 & 24 & 21 \\ 14 & 18 & 19 & 20 & 22 & 20 & 20 & 20 \end{bmatrix}$$

Quantization is achieved by dividing each element in the transformed image matrix D by the corresponding element in the quantization matrix, and then rounding to the nearest integer value. For the following step, quantization matrix Q_{50} is used.

$$C_{i,j} = round\left(\frac{D_{i,j}}{Q_{i,j}}\right) \qquad 6$$

$$C = \begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 1 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 1 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Recall that the coefficients situated near the upper-left corner correspond to the lower frequencies – to which the human eye is most sensitive – of the image block. In addition, the zeros represent the less important, higher frequencies that have been discarded, giving rise to

the lossy part of compression. As mentioned earlier, only the remaining nonzero coefficients will be used to reconstruct the image. It is also interesting to note the effect of different quantization matrices; use of Q_{10} would give C significantly more zeros, while Q_{90} would result in very few zeros.

Coding

The quantized matrix C is now ready for the final step of compression. Before storage, all coefficients of C are converted by an encoder to a stream of binary data (01101011...). In-depth coverage of the coding process is beyond the scope of this article. However, we can point out one key aspect that the reader is sure to appreciate. After quantization, it is quite common for most of the coefficients to equal zero. JPEG takes advantage of this by encoding quantized coefficients in the zig-zag sequence shown in Figure 1. The advantage lies in the consolidation of relatively large runs of zeros, which compress very well. The sequence in Figure 1 (4x4) continues for the entire 8x8 block.

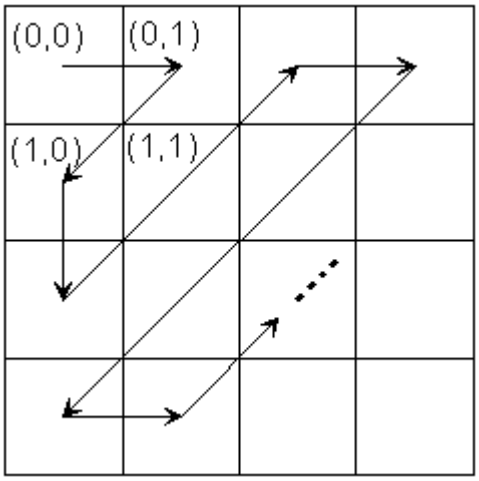


Figure 1

Decompression

Reconstruction of our image begins by decoding the bit stream representing the quantized matrix C. Each element of C is then multiplied by the corresponding element of the quantization matrix originally used.

$$R_{i,j} = Q_{i,j} \times C_{i,j} \qquad 7$$

$$R = \begin{bmatrix} 160 & 44 & 20 & 80 & 24 & 0 & 0 & 0 \\ 36 & 108 & 14 & 38 & 26 & 0 & 0 & 0 \\ -98 & -65 & 16 & -48 & -40 & 0 & 0 & 0 \\ -42 & -85 & 0 & -29 & 0 & 0 & 0 & 0 \\ -36 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The IDCT is next applied to matrix *R*, which is rounded to the nearest integer. Finally, 128 is added to each element of that result, giving us the decompressed JPEG version *N* of our original 8x8 image block *M*.

$$N = round(T' \; R \; T) + 128 \qquad 8$$

Comparison of Matrices

Let us now see how the JPEG version of our original pixel block compares.

$$Original = \begin{bmatrix} 154 & 123 & 123 & 123 & 123 & 123 & 123 & 136 \\ 192 & 180 & 136 & 154 & 154 & 154 & 136 & 110 \\ 254 & 198 & 154 & 154 & 180 & 154 & 123 & 123 \\ 239 & 180 & 136 & 180 & 180 & 166 & 123 & 123 \\ 180 & 154 & 136 & 167 & 166 & 149 & 136 & 136 \\ 128 & 136 & 123 & 136 & 154 & 180 & 198 & 154 \\ 123 & 105 & 110 & 149 & 136 & 136 & 180 & 166 \\ 110 & 136 & 123 & 123 & 123 & 136 & 154 & 136 \end{bmatrix}$$

$$Decompressed = \begin{bmatrix} 149 & 134 & 119 & 116 & 121 & 126 & 127 & 128 \\ 204 & 168 & 140 & 144 & 155 & 150 & 135 & 125 \\ 253 & 195 & 155 & 166 & 183 & 165 & 131 & 111 \\ 245 & 185 & 148 & 166 & 184 & 160 & 124 & 107 \\ 188 & 149 & 132 & 155 & 172 & 159 & 141 & 136 \\ 132 & 123 & 125 & 143 & 160 & 166 & 168 & 171 \\ 109 & 119 & 126 & 128 & 139 & 158 & 168 & 166 \\ 111 & 127 & 127 & 114 & 118 & 141 & 147 & 135 \end{bmatrix}$$

This is a remarkable result, considering that nearly 70% of the DCT coefficients were discarded prior to image block decompression/reconstruction. Given that similar results will occur with the rest of the blocks that constitute the entire image, it should be no surprise that

the JPEG image will be scarcely distinguishable from the original. Remember, there are 256 possible shades of gray in a black-and-white picture, and a difference of, say, 10, is barely noticeable to the human eye.

Pepper Example

We can do the DCT and quantization process on the peppers image.

Figure 2 – Peppers

Each eight by eight block is hit with the DCT, resulting in the image shown in Figure 3.

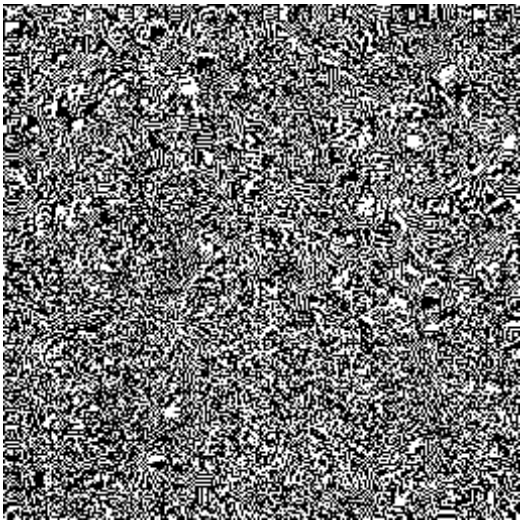


Figure 3 – DCT of Peppers

Each element in each block of the image is then quantized using a quantization matrix of quality level 50. At this point many of the elements become zeroed out, and the image takes up much less space to store.

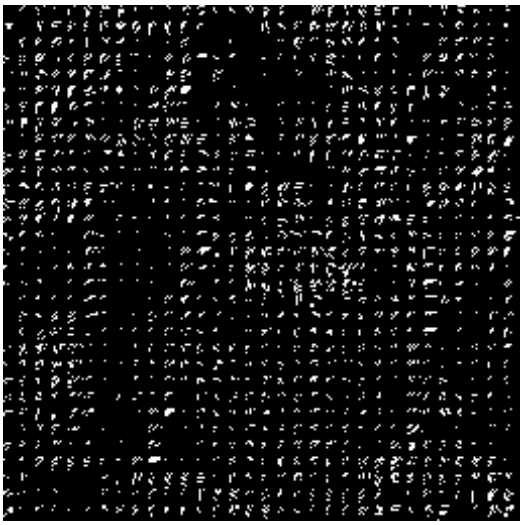


Figure 4 – Quantized DCT of Peppers

The image can now be decompressed using the inverse discrete cosine transform. At quality level 50 there is almost no visible loss in this image, but there is high compression. At lower quality levels, the quality goes down by a lot, but the compression does not increase very much.

Figure 5 – Original Peppers

Figure 6 – Quality 50 – 84% Zeros

Figure 7 – Quality 20 – 91% Zeros

Figure 8 – Quality 10 – 94% Zeros

More Examples

We can see what the compression does to other images. High contrast images, or images with a lot of high frequencies do not compress as well as smooth, low frequency images.

Figure 9 – Original

Figure 10 – Quality 15 – 90% Zeros

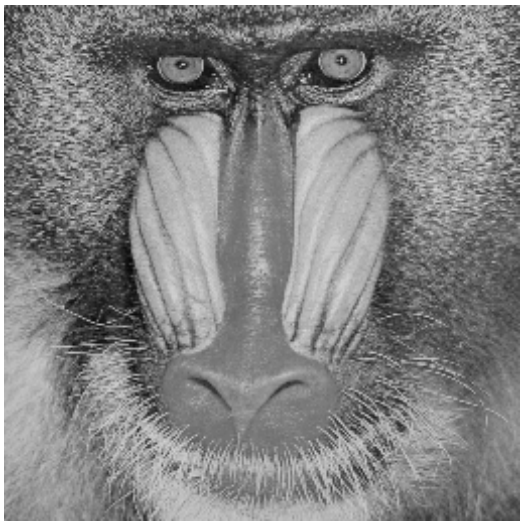


Figure 11 – Original

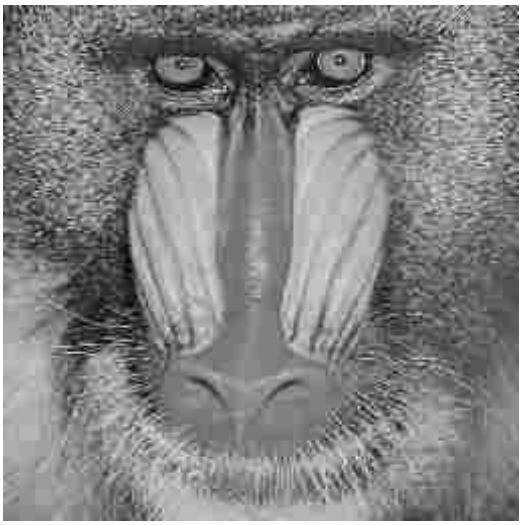


Figure 12 – Quality 15 – 88% Zeros

Bibliography

- Kesavan, Hareesh. *Choosing a DCT Quantization Matrix for JPEG Encoding*. Web page. http://www-ise.Stanford.EDU/class/ee392c/demos/kesavan/
- McGowan, John. *The Discrete Cosine Transform*. Web page. http://www.rahul.net/jfm/dct.html
- Wallace, Gregory K. *The JPEG Still Picture Compression Standard*. Paper submitted in December 1991 for publication in *IEEE Transactions on Consumer Electronics*.
- Wolfgang, Ray. *JPEG Tutorial*. Web page. http://www.imaging.org/tutorial/jpegtut1.html
- Our special thanks to David Arnold, math instructor extraordinaire.

IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 1, NO. 2, APRIL 1992

Image Coding Using Wavelet Transform

Marc Antonini, Michel Barlaud, *Member, IEEE,* Pierre Mathieu, and Ingrid Daubechies, *Member, IEEE*

***Abstract*—Image compression is now essential for applications such as transmission and storage in data bases. This paper proposes a new scheme for image compression taking into account psychovisual features both in the space and frequency domains; this new method involves two steps. First, we use a *wavelet transform* in order to obtain a set of biorthogonal subclasses of images; the original image is decomposed at different scales using a pyramidal algorithm architecture. The decomposition is along the vertical and horizontal directions and maintains constant the number of pixels required to describe the image. Second, according to Shannon's rate distortion theory, the wavelet coefficients are *vector quantized* using a multiresolution codebook. Furthermore, to encode the wavelet coefficients, we propose a noise shaping bit allocation procedure which assumes that details at high resolution are less visible to the human eye. Finally, in order to allow the receiver to recognize a picture as quickly as possible at minimum cost, we present a progressive transmission scheme. It is shown that the wavelet transform is particularly well adapted to progressive transmission.**

***Keywords*—Wavelet, biorthogonal wavelet, multiscale pyramidal algorithm, vector quantization, noise shaping, progressive transmission.**

I. Introduction

In many different fields, *digitized images* are replacing conventional analog images as photograph or x-rays. The volume of data required to describe such images greatly slow transmission and makes storage prohibitively costly. The information contained in the images must, therefore, be compressed by extracting only the visible elements, which are then encoded. The quantity of data involved is thus reduced substantially.

A fundamental goal of data compression is to reduce the bit rate for transmission or storage while maintaining an acceptable fidelity or image quality. Compression can be achieved by transforming the data, projecting it on a basis of functions, and then encoding this transform. Because of the nature of the image signal and the mechanisms of human vision, the transform used must accept nonstationarity and be well localized in both the space and frequency domains. To avoid redundancy, which hinders compression, the transform must be at least biorthogonal and lastly, in order to save CPU time, the corresponding algorithm must be fast. The two-dimensional wavelet transform defined by Meyer and Lemarié [31], [24], [25], together with its implementation as described by Mallat [27], satisfies each of these conditions.

The compression method we have developed associates a wavelet transform and a vector quantization coding scheme. The wavelet coefficients are coded considering a noise shaping bit allocation procedure. This technique exploits the psychovisual as well as statistical redundancies in the image data, enabling bit rate reduction.

Section II describes the wavelet transforms used in this paper. After a quick review of wavelets in general, we explain in more detail the properties and construction of regular biorthogonal wavelet bases. We then extend this one-dimensional construction to a two-dimensional scheme with separable filters. The new coding scheme is next presented in Section III. We focus particularly in this section on the statistical properties of wavelet coefficients, on the asymptotic coding gain that can be achieved using vector quantization in the subimages, and on the optimal allocation across the subimages. Experimental results are given in Section IV for images taken within and outside of the training set.

II. Wavelets

A. A Short Review of Wavelet Analysis

Wavelets are functions generated from one single function ψ by dilations and translations

$$\psi^{a,b}(t) = |a|^{-1/2}\psi\left(\frac{t-b}{a}\right).$$

(For this introduction we assume t is a one-dimensional variable). The *mother wavelet* ψ has to satisfy $\int dx\, \psi(x) = 0$, which implies at least some oscillations. (Technically speaking, the condition on ψ should be $\int d\omega\, |\Psi(\omega)|^2\, |\omega|^{-1} < \infty$, where Ψ is the Fourier transform of ψ; if $\psi(t)$ decays faster than $|t|^{-1}$ for $t \to \infty$, then this condition is equivalent to the one above). The definition of wavelets as dilates of one function means that high frequency wavelets correspond to $a < 1$ or narrow width, while low frequency wavelets have $a > 1$ or wider width.

The basic idea of the wavelet transform is to represent any arbitrary function f as a superposition of wavelets. Any such superposition decomposes f into different scale levels, where each level is then further decomposed with a resolution adapted to the level. One way to achieve such a decomposition writes f as an integral over a and b of $\psi^{a,b}$ with appropriate weighting coefficients [22]. In practice, one prefers to write f as a discrete superposition (sum rather than integral). Therefore, one introduces a discre-

Manuscript received February 7, 1990; revised March 26, 1991.

M. Antonini, M. Barlaud, and P. Mathieu are with LASSY 13S CNRS, Universite de Nice-Sophia Antipolis, 06560 Valbonne, France.

I. Daubechies is with AT&T Bell Laboratories, Murray Hill, NJ 07974.

IEEE Log Number 9106073.

tization, $a = a_0^m$, $b = nb_0a_0^m$, with $m, n \in \mathbb{Z}$, and $a_0 > 1$, $b_0 > 0$ fixed. The wavelet decomposition is then

$$f = \sum c_{m,n}(f)\psi_{m,n} \tag{1}$$

with $\psi_{m,n}(t) = \psi^{a_0^m, nb_0a_0^m}(t) = a_0^{-m/2}\psi(a_0^{-m}t - nb_0)$. Decompositions of this type were studied in [14], [15]. For $a_0 = 2$, $b_0 = 1$ there exist very special choices of ψ such that the $\psi_{m,n}$ constitute an orthonormal basis, so that

$$c_{m,n}(f) = \langle \psi_{m,n}, f \rangle = \int dx\, \psi_{m,n}(x) f(x)$$

in this case. Different bases of this nature were constructed by Stromberg [36], Meyer [31], Lemarié [24], Battle [7], and Daubechies [16]. All these examples correspond to a multiresolution analysis, a mathematical tool invented by Mallat [27], which is particularly well adapted to the use of wavelet bases in image analysis, and which gives rise to a fast computation algorithm.

In a multiresolution analysis, one really has *two* functions: the mother wavelet ψ and a *scaling function* ϕ. One also introduces dilated and translated versions of the scaling function, $\phi_{m,n}(x) = 2^{-m/2}\phi(2^{-m}x - n)$. For fixed m, the $\phi_{m,n}$ are orthonormal. We denote by V_m the space spanned by the $\phi_{m,n}$; these spaces V_m describe successive approximation spaces, $\cdots V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \cdots$, each with resolution 2^m. For each m, the $\psi_{m,n}$ span a space W_m which is exactly the orthogonal complement in V_{m-1} of V_m; the coefficients $\langle \psi_{m,n}, f \rangle$, therefore, describe the information lost when going from an approximation of f with resolution 2^{m-1} to the coarser approximation with resolution 2^m. All this is translated into the following algorithm for the computation of the $c_{m,n}(f) = \langle \psi_{m,n}, f \rangle$ (for more details, see [27]):

$$c_{m,n}(f) = \sum_k g_{2n-k}a_{m-1,k}(f)$$

$$a_{m,n}(f) = \sum_k h_{2n-k}a_{m-1,k}(f) \tag{2}$$

where $g_l = (-1)^l h_{-l+1}$ and $h_n = 2^{1/2}\int dx\, \phi(x-n)\phi(2x)$. In fact the $a_{m,n}(f)$ are coefficients characterizing the projection of f onto V_m. If the function f is given in sampled form, then one can take these samples for the highest order resolution approximation coefficients $a_{0,n}$, and (2) describes a subband coding algorithm on these sampled values, with low-pass filter h and high-pass filter g. Because of their association with orthonormal wavelet bases, these filters give exact reconstruction, i.e.:

$$a_{m-1,l}(f) = \sum_n [h_{2n-l}a_{m,n}(f) + g_{2n-l}c_{m,n}(f)]. \tag{3}$$

Most of the orthonormal wavelet bases have infinitely supported ψ, corresponding to filters h and g with infinitely many taps. The construction in [16] gives ψ with finite support, and therefore, corresponds to FIR filters. It follows that the orthonormal bases in [16] correspond to a subband coding scheme with exact reconstruction property, using the same FIR filters for reconstruction as for decomposition. Such filters are well known since the work of Smith and Barnwell [35] and of Vetterli [37]. The extra ingredient in the orthonormal wavelet decomposition is that it writes the signal to be decomposed as a superposition of reasonably *smooth* elementary building blocks. The filters must satisfy the additional condition:

$$\prod_{k=1}^{\infty} H(2^{-k}\xi)$$

decay faster than $C(1 + |\xi|)^{-\epsilon - 0.5}$ as $|\xi| \to \infty$, for some $\epsilon > 0$, where

$$H(\xi) = 2^{-1/2}\sum_n h_n e^{-jn\xi}.$$

This extra regularity requirement is usually not satisfied by the exact reconstruction filters in the ASSP literature.

B. Applications of Wavelet Bases to Image Analysis

1) Biorthogonal Wavelet Bases: Since images are mostly smooth (except for occasional edges) it seems appropriate that an exact reconstruction subband coding scheme for image analysis should correspond to an orthonormal basis with a reasonably smooth mother wavelet. In order to have fast computation, the filters should be short (short filters lead to less smoothness, however, so they cannot be too short). On the other hand it is desirable that the FIR filters used be linear phase, since such filters can be easily cascaded in pyramidal filter structures without the need for phase compensation. Unfortunately, there are no nontrivial orthonormal linear phase FIR filters with the exact reconstruction property [35], regardless of any regularity considerations. The only symmetric exact reconstruction filters are those corresponding to the Haar basis, i.e., $h_0 = h_1 = 2^{1/2}$ and $g_0 = -g_1 = 2^{1/2}$, with all other h_n, $g_n = 0$.

One can preserve linear phase (corresponding to symmetry for the wavelet) by relaxing the orthonormality requirement, and using biorthogonal bases. It is then still possible to construct examples where the mother wavelets have arbitrarily high regularity.

In such a scheme, we still decompose as in (2), but reconstruction becomes

$$a_{m-1,l}(f) = \sum_n [\tilde{h}_{2n-l}a_{m,n}(f) + \tilde{g}_{2n-l}c_{m,n}(f)] \tag{4}$$

where the filters $\tilde{h}$, $\tilde{g}$ may be different from h, g. In order to have exact reconstruction, we impose:

$$\begin{aligned} \tilde{g}_n &= (-1)^n h_{-n+1} \\ g_n &= (-1)^n \tilde{h}_{-n+1} \end{aligned} \qquad \sum_n h_n\tilde{h}_{n+2k} = \delta_{k,0}. \tag{5}$$

So far, we have not performed anything differently from the usual exact reconstruction subband coding schemes with synthesis filters different from the decomposition filters. If the filters satisfy the additional condition that:

$$\prod_{k=1}^{\infty} \tilde{H}(2^{-k}\xi) \quad \text{and} \quad \prod_{k=1}^{\infty} H(2^{-k}\xi) \tag{6a}$$

decay faster than $C(1 + |\xi|)^{-\epsilon - 0.5}$ as $|\xi| \to \infty$, for some $\epsilon > 0$, where

$$\tilde{H}(\xi) = 2^{-1/2} \sum_n \tilde{h}_n e^{-jn\xi} \quad H(\xi) = 2^{-1/2} \sum_n h_n e^{-jn\xi} \tag{6b}$$

then we can give the following interpretation to (2) and (4). Define functions ϕ and $\tilde{\phi}$ by

$$\phi(x) = \sum_n h_n \phi(2x - n) \quad \text{and} \quad \tilde{\phi}(x) = \sum_n \tilde{h}_n \tilde{\phi}(2x - n).$$

Their Fourier transforms are exactly the infinite products (6a), and they are, therefore, well-defined square integrable functions, compactly supported if the filters h and $\tilde{h}$ are FIR. Define also

$$\psi(x) = \sum_n g_n \phi(2x - n) \quad \text{and} \quad \tilde{\psi}(x) = \sum_n \tilde{g}_n \tilde{\phi}(2x - n).$$

Then, the $a_{m,n}(f)$ and $c_{m,n}(f)$ in (2) can be rewritten as:

$$a_{m,n}(f) = \langle \phi_{m,n}, f \rangle = 2^{-m/2} \int dx\, \phi_{m,n}(x) f(x)$$

$$c_{m,n}(f) = \langle \psi_{m,n}, f \rangle = 2^{-m/2} \int dx\, \psi_{m,n}(x) f(x)$$

and reconstruction is simply:

$$f = \sum_{m,n} \langle \psi_{m,n}, f \rangle \tilde{\psi}_{m,n}. \tag{7}$$

The filter bank structure with the associating wavelets and scaling functions is depicted on the following subband coding scheme (Fig. 1).

If the infinite products in (6a) decay even faster than imposed above, then ϕ and $\tilde{\phi}$ and consequently ψ and $\tilde{\psi}$ will be reasonably smooth. Note that (7) is very similar to the orthonormal decomposition described in Section II-A; the only difference is that the expansion of f with respect to the basis $\tilde{\psi}_{m,n}$ uses coefficients computed via the dual basis $\psi_{m,n}$ with $\tilde{\psi}$ different from ψ. This interpretation is not possible for all exact reconstruction subband coding schemes; in particular, convergence of the infinite products (6a) is only possible if

$$\sum_n h_n = 2^{1/2} \quad \text{and} \quad \sum_n \tilde{h}_n = 2^{1/2}.$$

Moreover, (7) can only hold if

$$\sum_n (-1)^n h_n = 0 \quad \text{and} \quad \sum_n (-1)^n \tilde{h}_n = 0.$$

Most exact reconstruction subband coding schemes do not satisfy these conditions.

Biorthogonal bases of wavelets have recently been constructed, with regularity simultaneously but independently, by Cohen, Daubechies and Feauveau [12] and by Herley and Vetterli [38]. Reference [12] contains a detailed mathematical study, with proofs that, under the conditions stated above, the wavelets do indeed constitute numerically stable bases (Riesz bases) and a discussion of necessary and sufficient conditions for regularity. In [18] Feauveau explores the construction from the point of view of multiresolution spaces rather than from the filters. Basically one has two hierarchies of spaces in the biorthogonal case, each corresponding to one pair of filters.

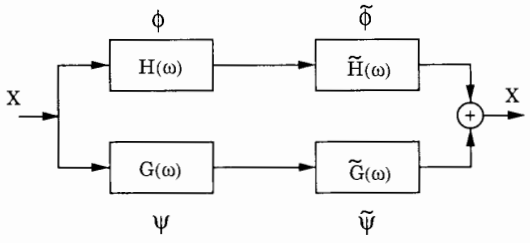


Fig. 1. Filter bank structure and the associating wavelets.

It is shown in [12] that arbitrarily high regularity can be achieved by both ψ and $\tilde{\psi}$, provided one chooses sufficiently long filters. In particular, if the functions ψ and $\tilde{\psi}$ are, respectively, $(k - 1)$ and $(\tilde{k} - 1)$ times continuously differentiable, then the trigonometric polynomials $H(\xi)$ and $\tilde{H}(\xi)$ have to be divisible by $(1 + e^{-j\xi})^k$ and $(1 + e^{-j\xi})^{\tilde{k}}$, respectively, so that the length of the corresponding filters h, $\tilde{h}$ has to exceed k, $\tilde{k}$.

By (5), divisibility of $\tilde{H}(\xi)$ by $(1 + e^{-j\xi})^{\tilde{k}}$ means that ψ will have $\tilde{k}$ consecutive moments zero:

$$\int dx\, x^l \psi(x) = 0, \qquad \text{for} \quad l = 0, 1, \cdots, \tilde{k} - 1.$$

For more details concerning this discussion, see [12].

It is well known (and it can easily be checked by using Taylor expansions) that if ψ has $\tilde{k}$ moments zero, then the coefficients $\langle \psi_{m,n}, f \rangle$ will represent functions f, which are $\tilde{k}$ times differentiable, with a high compression potential (many coefficients will be negligibly small).

Many examples of biorthogonal wavelet bases with reasonably regular ψ and $\tilde{\psi}$ can be constructed; for our applications, the regularity of the elementary building blocks $\tilde{\psi}_{m,n}$, which is linked to the number of zero moments of ψ, is more important than the regularity of the $\psi_{m,n}$ or the number of zero moments of $\tilde{\psi}$. Within the limits imposed by the support widths, we will, therefore, try to choose $\tilde{k}$ as large as possible.

In terms of trigonometric polynomials $H(\xi)$ and $\tilde{H}(\xi)$, the exact reconstruction requirement condition on h and $\tilde{h}$ given in (5) reduces to (for symmetric filters)

$$H(\xi)\tilde{H}(\xi) + H(\xi + \pi)\tilde{H}(\xi + \pi) = 1. \tag{8}$$

Together with divisibility of H and $\tilde{H}$, respectively, by $(1 + e^{-j\xi})^k$ and $(1 + e^{-j\xi})^{\tilde{k}}$, this leads to (see [12])

$$H(\xi)\tilde{H}(\xi) = \cos(\xi/2)^{2l} \left[\sum_{p=0}^{l-1} \binom{l - 1 + p}{p} \cdot \sin(\xi/2)^{2p} + \sin(\xi/2)^{2l} R(\xi) \right] \tag{9}$$

where $R(\xi)$ is an odd polynomial in $\cos(\xi)$, and where $2l = k + \tilde{k}$ (symmetry of h and $\tilde{h}$ forces $k + \tilde{k}$ to be even).

TABLE I
FILTER COEFFICIENTS FOR THE SPLINE FILTERS WITH $l = 3$, $k = 4$, $\tilde{k} = 2$

n	0	±1	±2	±3	±4
$2^{-1/2}h_n$	45/64	19/64	−1/8	−3/64	3/128
$2^{-1/2}\tilde{h}_n$	1/2	1/4	0	0	0

Many examples are possible. We have studied in particular the following three examples, which belong to three different families.

2) Spline Filters: One can choose, e.g., $R \equiv 0$, with $\tilde{H}(\xi) = \cos(\xi/2)^{\tilde{k}} e^{-j\kappa\xi/2}$ where $\kappa = 0$ if $\tilde{k}$ is even, $\kappa = 1$ if $\tilde{k}$ is odd. This corresponds to the filters called "spline filters" in [12] (because the corresponding function $\tilde{\phi}$ is a B-spline function) or "binomial filters" in [38] (because the $\tilde{h}$ are simply binomial coefficients). It then follows that:

$$H(\xi) = \cos(\xi/2)^{2l-\tilde{k}} e^{j\kappa\xi/2} \cdot \left[\sum_{p=0}^{l-1} \binom{l-1+p}{p} \sin(\xi/2)^{2p}\right]. \quad (10)$$

We have looked at one example from this family; it corresponds to $l = 3$, $\tilde{k} = 2$. The coefficients h_n and $\tilde{h}_n$ are listed in Table I; the corresponding scaling functions and wavelets are plotted in Fig. 2.

It is clear that the two filters in the first example have very uneven length. This is typical for all the examples in this family of "spline filters."

3) A Spline Variant with Less Dissimilar Lengths: This family still uses $R \equiv 0$ in (9), but factorizes the right-hand side of (9), breaking up the polynomial of degree $l-1$ in $\sin(\xi/2)$ into a product of two polynomials in $\sin(\xi/2)$ with real coefficients, one to be allocated to H, the other to $\tilde{H}$, so as to make the lengths of h and $\tilde{h}$ as close as possible.

The example presented here is the "smallest" one in this family (shortest h and $\tilde{h}$); it corresponds to $l = 4$ and $k = 4$. The filter coefficients are listed in Table II; the corresponding scaling functions and wavelets are plotted in Fig. 3.

Note that, unlike examples 1 and 3 where the $2^{-1/2}h_n$, $2^{-1/2}\tilde{h}_n$ are rational, the entries in Table II are truncated decimal expansions of irrational numbers. The functions ϕ in examples 1 and 2 look very similar (compare Figs. 2(a) and 3(a)); a more detailed analysis shows that the one in example 2 is more regular, however. Both correspond to 4 vanishing moments for $\tilde{\psi}$.

4) Filters Close to Orthonormal Filters: Finally, there exist many examples for which $R \neq 0$. In particular there exists a special choice of R for which the two filters are very close to each other, and both very close to an orthonormal wavelet filter.

Surprisingly, for the first example of this series, one of the two filters is a Laplacian pyramid filter proposed in [9]. It corresponds to $l = 2$, $k = 2$ and $R(\xi) = 48\cos(\xi)/175$. The filter coefficients are listed in Table III; the corresponding scaling functions and wavelets are plotted in Fig. 4. It is clear that the scaling functions ϕ and $\tilde{\phi}$ are very similar, corresponding to very similar ψ and $\tilde{\psi}$. Note that in this case, the filter coefficients are again rational.

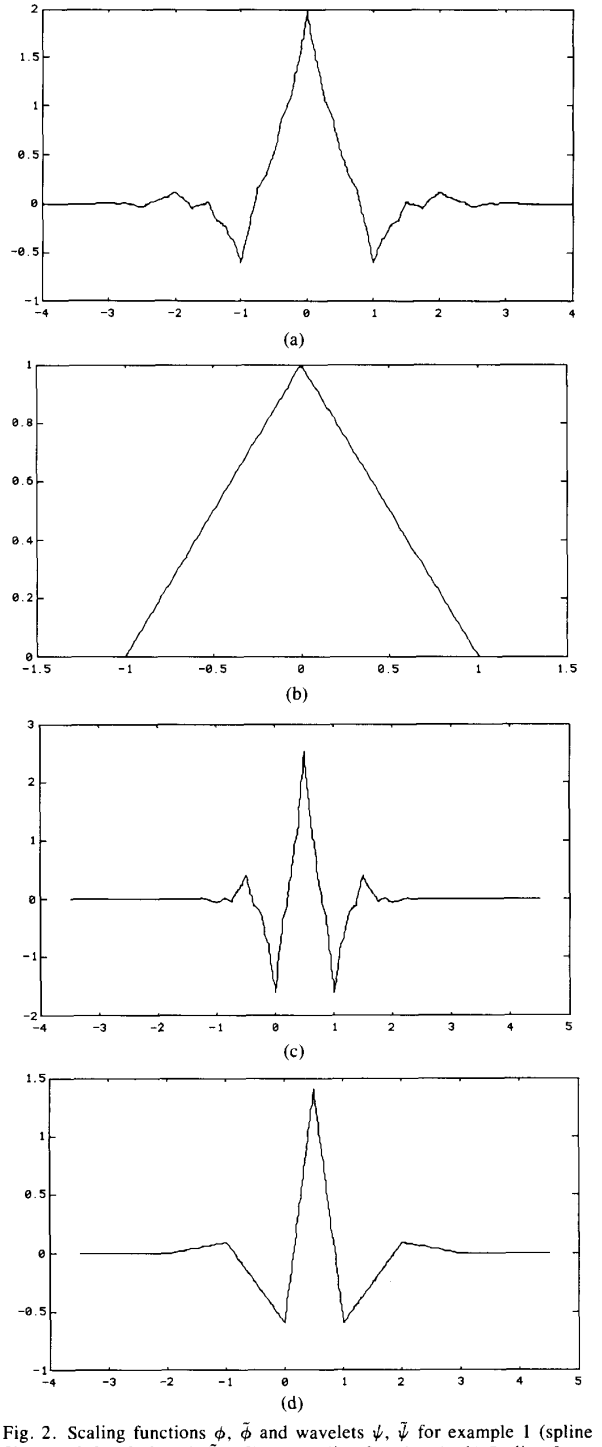


Fig. 2. Scaling functions ϕ, $\tilde{\phi}$ and wavelets ψ, $\tilde{\psi}$ for example 1 (spline filters with $l = 3$, $k = 4$, $\tilde{k} = 2$). (a) Scaling function ϕ. (b) Scaling function $\tilde{\phi}$. (c) Wavelet ψ. (d) Wavelet $\tilde{\psi}$.

TABLE II
FILTER COEFFICIENTS FOR THE SPLINE VARIANT WITH LESS DISSIMILAR LENGTHS, WITH $l = 4 = k$, $\tilde{k} = 4$

n	0	± 1	± 2	± 3	± 4
$2^{-1/2}h_n$	0.602 949	0.266 864	−0.078 223	−0.016 864	0.026 749
$2^{-1/2}\tilde{h}_n$	0.557 543	0.295 636	−0.028 772	−0.045 636	0

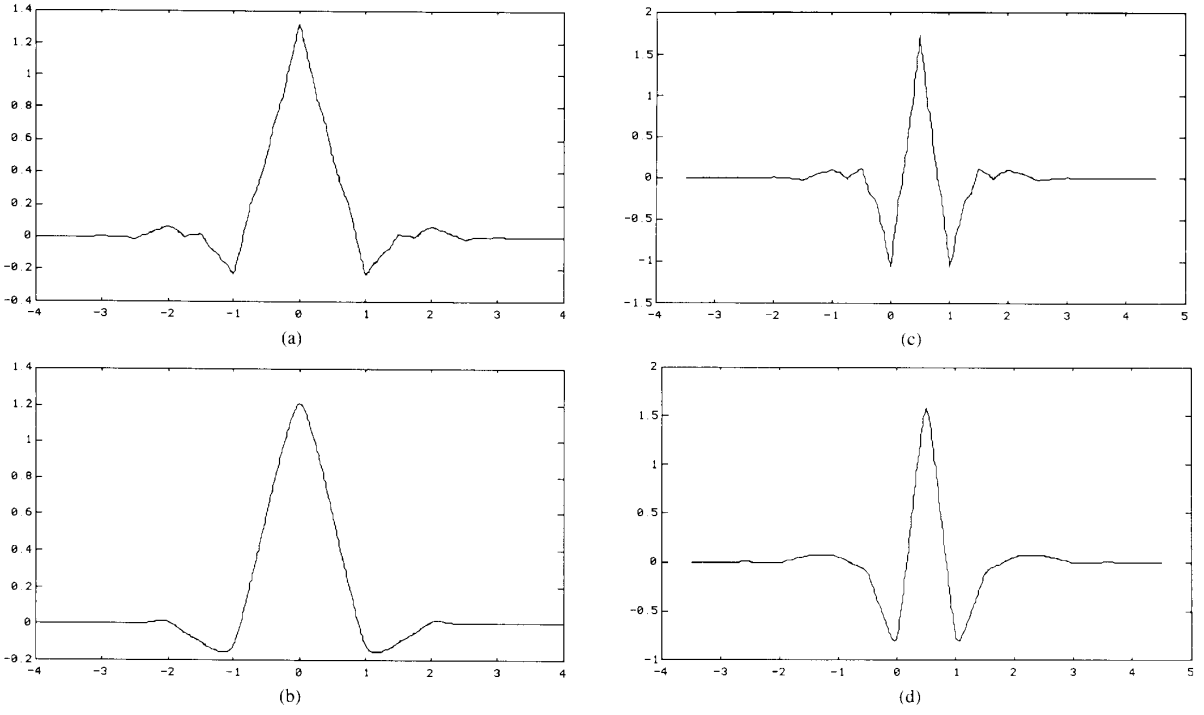


Fig. 3. Scaling functions ϕ, $\tilde{\phi}$ and wavelets ψ, $\tilde{\psi}$ for example 2 (spline variant with less dissimilar lengths; $l = 4 = k$, $\tilde{k} = 4$). (a) Scaling function ϕ. (b) Scaling function $\tilde{\phi}$. (c) Wavelet ψ. (d) Wavelet $\tilde{\psi}$.

TABLE III
FILTER COEFFICIENTS FOR EXAMPLE 3. THE ENTRIES ARE RATIONAL, AND THE TWO FILTERS ARE VERY CLOSE. THE h-FILTER COINCIDES WITH A LAPLACIAN PYRAMID FILTER PROPOSED IN [9]. IN THIS CASE $l = 2 = k$, $\tilde{k} = 2$

n	0	± 1	± 2	± 3	± 4
$2^{-1/2}h_n$	0.6	0.25	−0.05	0	0
$2^{-1/2}\tilde{h}_n$	17/28	73/280	−3/56	−3/280	0

The two biorthogonal filters in this example are both close to an orthonormal wavelet filter of length 6 constructed in [17], where it was called a "coiflet." Being an orthonormal wavelet filter, the coiflet is nonsymmetric. The filters in this example are shorter than in examples 1 and 2, but k is also smaller. The next example in this family corresponds to $k = 4$ (and $l = 4$); the filters h and $\tilde{h}$ then have length 9 and 15; they are both close to a coiflet of length 12.

5) Extension to the Two-Dimensional Case: There exist various extensions of the one-dimensional wavelet transform to higher dimensions. We follow Mallat [27] and use a two-dimensional wavelet transform in which horizontal and vertical orientations are considered preferential.

In two-dimensional wavelet analysis one introduces, like in the one-dimensional case, a scaling function $\phi(x, y)$ such that:

$$\phi(x, y) = \phi(x)\phi(y) \tag{11}$$

where $\phi(x)$ is a one-dimensional scaling function.

Let $\psi(x)$ be the one-dimensional wavelet associated with the scaling function $\phi(x)$. Then, the three two-dimensional wavelets are defined as:

$$\begin{aligned} \psi^H(x, y) &= \phi(x)\psi(y) \\ \psi^V(x, y) &= \psi(x)\phi(y) \\ \psi^D(x, y) &= \psi(x)\psi(y). \end{aligned} \tag{12}$$

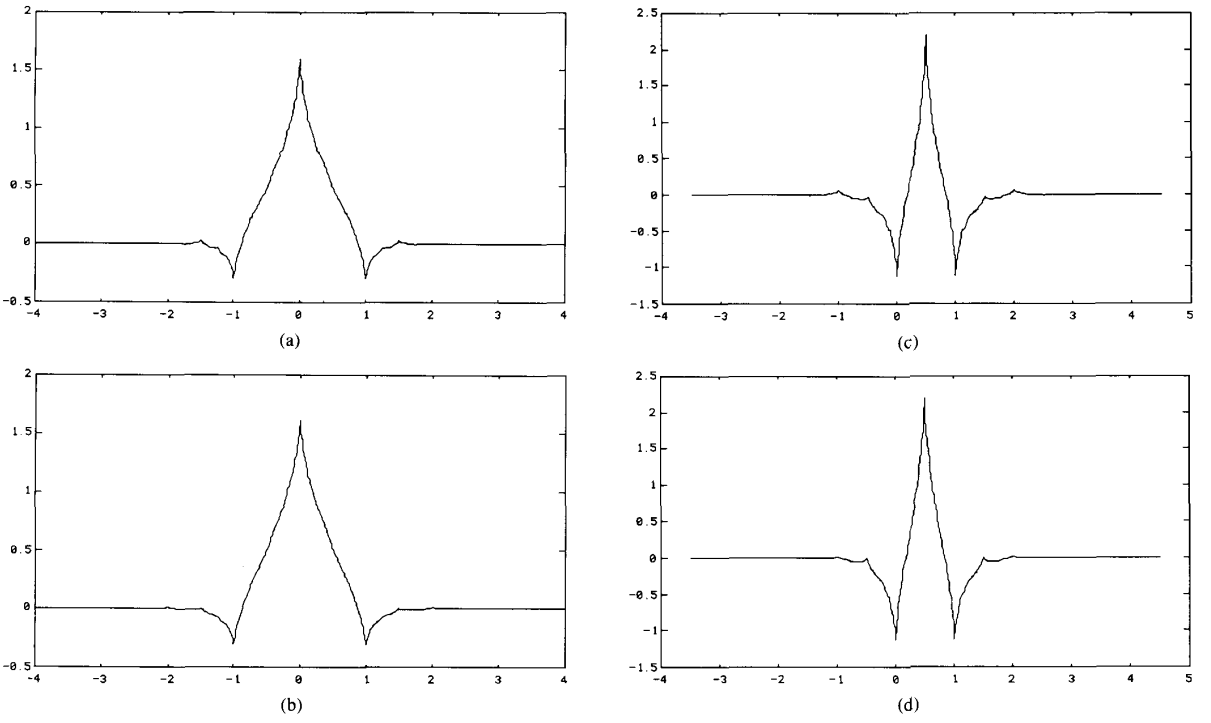


Fig. 4. Scaling functions ϕ, $\tilde{\phi}$ and wavelets ψ, $\tilde{\psi}$ for example 3 (biorthogonal filters close to an orthonormal wavelet filter, $l = 2 = k$, $\tilde{k} = 2$). (a) Scaling function ϕ. (b) Scaling function $\tilde{\phi}$. (c) Wavelet ψ. (d) Wavelet $\tilde{\psi}$.

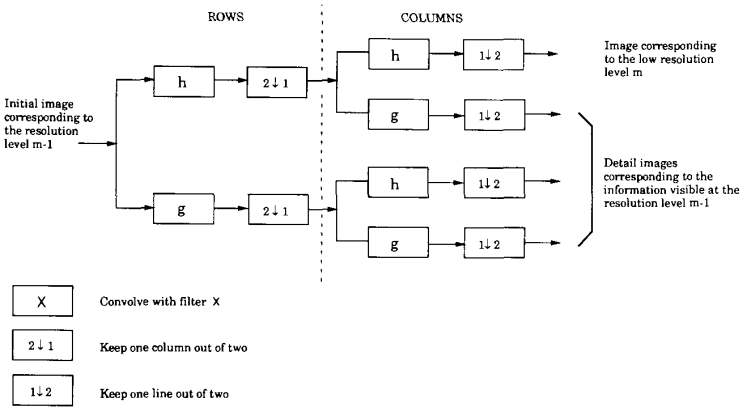


Fig. 5. One stage in a multiscale image decomposition.

Fig. 5 represents one stage in a *multiscale pyramidal* decomposition of an image: wavelet coefficients of the image are computed, as in the one-dimensional case (Sections II-A and II-B.1), using a subband coding algorithm. The filters h and g are one-dimensional filters. This decomposition provides subimages corresponding to different resolution levels and orientations (see Fig. 6). The reconstruction scheme of the image is presented Fig. 7.

To compare the three different filters presented in this paper, we have decomposed the image Lena (Fig. 16) with each of these filters. The results are presented in Fig. 8. In Fig. 8(a) we can see the normalized detail subimages at different resolution levels $m = 1$, $m = 2$, and $m = 3$ (wavelet coefficients) and in Fig. 8(b) the low resolution level subimages.

III. Image Coding Application

A. Statistical Properties of Wavelet Coefficients

The performance of a coder used for a given resolution and direction can be determined by the statistics of the corresponding subimage, i.e., its probability density function (PDF).

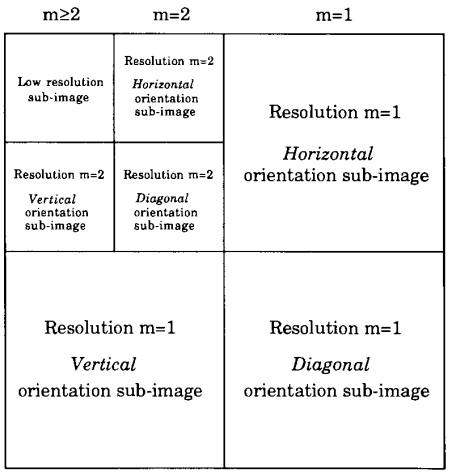


Fig. 6. Image decomposition.

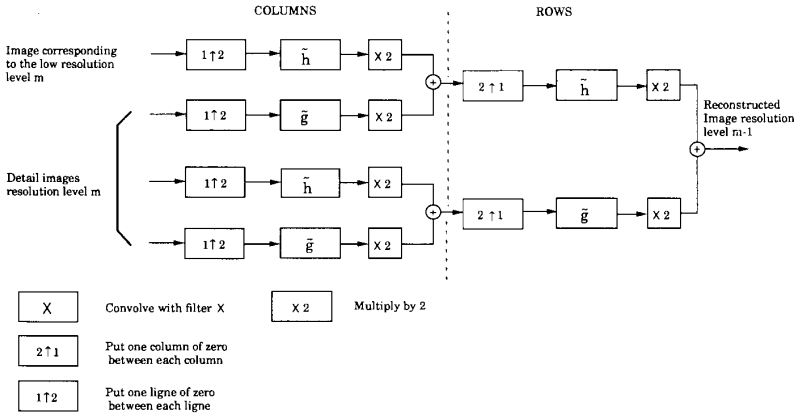


Fig. 7. One stage in a multiscale image reconstruction.

A typical PDF and different approximations are given in Fig. 9, where we plot the true PDF for resolution level $m = 1$ and direction d = vertical together with three model functions: a Gaussian, a Laplacian, and an intermediate function, the so-called generalized Gaussian [2].

This generalized Gaussian law is given explicitly by

$$p_{m,d}(x) = a_{m,d} \exp\left(-|b_{m,d}x|^{r_{m,d}}\right)$$

with

$$a_{md} = \frac{b_{m,d} r_{m,d}}{2\Gamma\left(\frac{1}{r_{m,d}}\right)} \quad \text{and} \quad b_{m,d} = \frac{1}{\sigma_{m,d}} \frac{\Gamma\left(\frac{3}{r_{m,d}}\right)^{1/2}}{\Gamma\left(\frac{1}{r_{m,d}}\right)^{1/2}} \tag{13}$$

where $\sigma_{m,d}$ is the standard deviation of the subimage (m, d), and $\Gamma(\cdot)$ is the usual Gamma function.

The general formula (13) contains the other two examples as particular cases:

- $r_{m,d} = 2$ leads to the well-known Gaussian PDF;
- $r_{m,d} = 1$ leads to a Laplacian PDF.

The variance of this approximation model is set equal to the variance of the corresponding subimage. Thus the parameter $r_{m,d}$ is computed in order to match the real PDF using the well-known chi-squared test. In this case the optimum parameter was 0.7. Other experiments for other resolutions (except the lowest resolution) lead to very similar results.

We can see in Fig. 9 that the real PDF (scale $m = 1$ and vertical orientation) is closely approximated by a generalized Gaussian law with parameter $r_{1,t} = 0.7$.

B. Encoding of Wavelet Coefficients Using Vector Quantization

Different techniques involving vector or scalar quantization can be used to encode wavelet coefficients.

According to Shannon's rate distortion theory, better results are always obtained when vectors rather than sca-

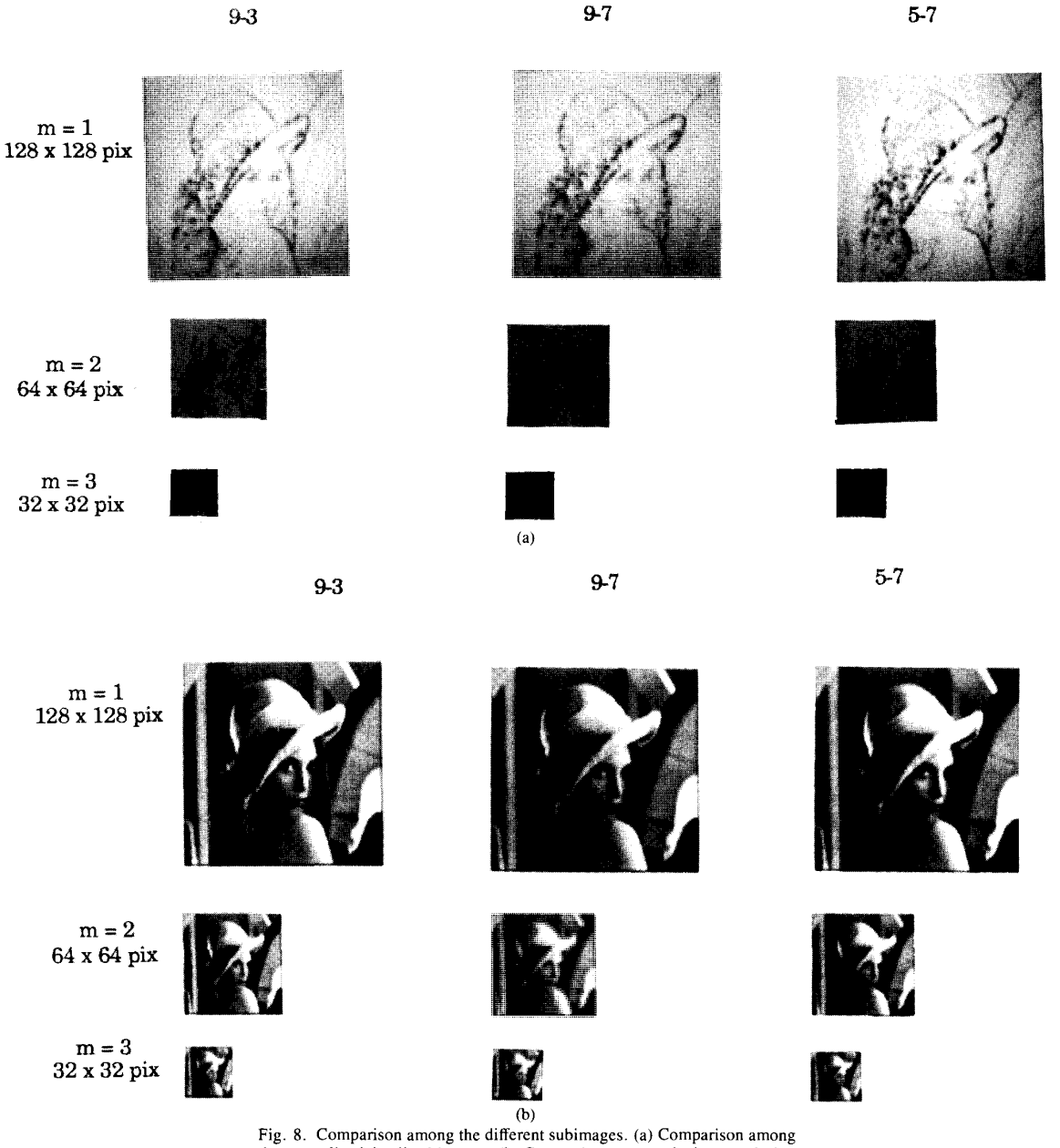


Fig. 8. Comparison among the different subimages. (a) Comparison among the normalized detail subimages. (b) Comparison among the low resolution level subimages.

lars are encoded. Therefore, the present application uses vector quantization.

1. Principle of Vector Quantization: Developed recently by Gersho and Gray (1980) [20], [21], vector quantization has proven to be a powerful tool for digital image compression [4], [29], [30], [32], [39]. The principle involves encoding a sequence of samples (vector) rather than encoding each sample individually. Encoding is performed by approximating the sequence to be coded by a vector belonging to a catalogue of shapes, usually known as a codebook.

The codebook is created and optimized using the well-known Linde–Buzo–Gray (LBG) [26] classification algorithm with a mean squared error (MSE) criterion. This algorithm is designed to perform a classification based on a training set comprised of vectors belonging to different images; it converges iteratively toward a locally optimal codebook.

Each of the vectors in the codebook is indexed. At the encoding stage, the index of the vector in the codebook most closely describing (in terms of MSE criterion) the sample set to be encoded is selected to represent this set. Of course, in order to reconstruct the sample set, the decoder must have the same codebook as the coder.

The encoding/decoding scheme depicted in Fig. 10 was proposed in [29] and [30] for orthonormal wavelets.

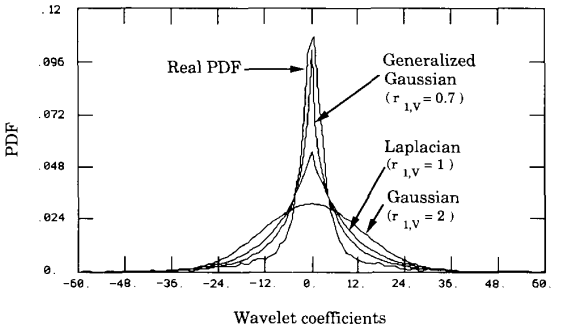


Fig. 9. Real PDF of subimage at scale $m = 1$ for vertical orientation, and its different approximations.

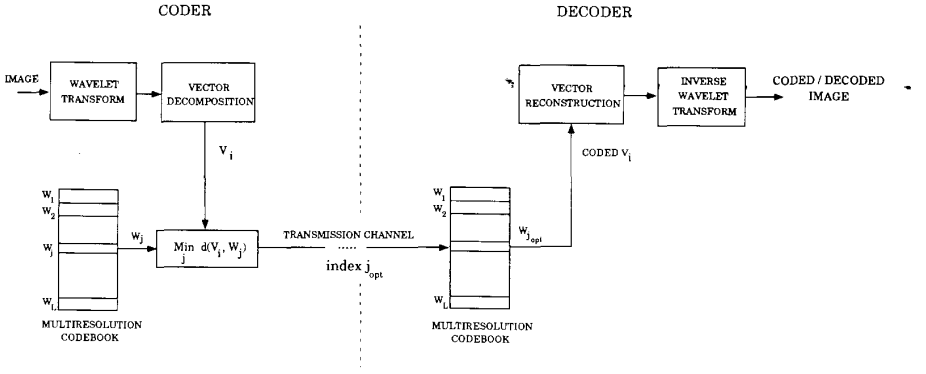


Fig. 10. Encoding/decoding scheme.

2) Comparative Performances of Vector Quantization (VQ) and Scalar Quantization (SQ): According to [3], [13], [19], [43], [30] the asymptotic lower bound distortion gain obtained when VQ, rather than SQ, is applied to a subimage is expressed as:

$$G_{m,d}^{VQ} \geq \frac{2^{-c}}{(c+1)A(k_{m,d}, c)} \times \frac{\left[\int [p_{m,d}(x)]^{1/(c+1)}\, dx\right]^{(c+1)}}{\left[\int [p_{m,d}(x)]^{k_{m,d}/(c+k_{m,d})}\, dx\right]^{(c+k_{m,d})}} \tag{14}$$

for a subimage corresponding to resolution m and direction d. $p_{m,d}(x)$ is the PDF of wavelet coefficients of the subimage with resolution m and direction d.

Here, the MSE criterion is used as a distortion measure ($c = 2$). The values of $A(k_{m,d}, 2)$ used are the upper bounds of the MSE computed and tabulated by Conway and Sloane for vector size $k_{m,d}$ [13]. This formula gives an indication of the minimum theoretical gain that can be obtained.

However, this approximation is valid only for small quantization errors, i.e., for a high bit rate $R_{m,d}$. Thus the gain $G_{m,d}^{VQ}$ only gives here an asymptotic indication.

In Fig. 11, the curves of $G_{m,d}^{VQ}$ are plotted as a function of the vector dimension $k_{m,d}$ for the Laplacian, Gaussian,

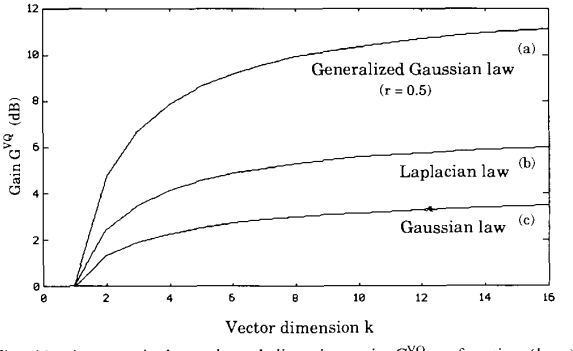


Fig. 11. Asymptotic lower bound distortion gain $G_{m,d}^{VQ}$ = function $(k_{m,d})$.

and generalized Gaussian approximation laws, and for a subimage at scale $m = 1$ and vertical orientation. Experimental results are closely matched by the theoretical results for a generalized Gaussian law with $r_{m,d} = 0.7$ except for the lower subband. Therefore, all computations based on this approximation law show that, in each subband, VQ outperforms SQ (see Fig. 11).

In summary VQ performs better for coding wavelet coefficients.

3) Generation of a Multiresolution Codebook: The preceding paragraph explained why VQ outperforms other methods. Nonetheless, major problems are encountered in the VQ of images.

- It is impossible to create a universal codebook (efficient for each image to be encoded).

- The LBG algorithm smooths high frequencies (loss of resolution).
- There is a trade-off between low distortion and high compression rate (computational cost).
- It is not easy to take into account the properties of the human visual system [28], [33].

The use of the wavelet transform (i.e., multiresolution) is one way of overcoming these different problems.

The wavelet decomposition of an image enables the generation of a codebook containing two-dimensional vectors for *each resolution level and preferential direction* (horizontal, vertical, and diagonal). Each of these subcodebooks (see Fig. 12) is generated using the LBG algorithm.

- The training set is comprised of vectors belonging to different images corresponding to the resolution and orientation under consideration.
- The initial codebook is generated by splitting the centroid (center of gravity) of this training set [21].

A multiresolution codebook can thus be obtained by assembling all of these resulting subcodebooks. Each subcodebook has a low distortion level and contains few words, which clearly facilitates the search for the best coding vector; the coding computational load is reduced, because only the appropriate subcodebook (resolution direction) of the multiresolution codebook is checked for each input vector. In addition, the quality of the coded image is better. The multiresolution codebook is depicted in Fig. 12.

Global codebook design has drawbacks in that it results in edge smoothing while the proposed method preserves edges. In fact, each subcodebook contains the shape of the wavelet coefficients which are most highly representative in terms of the MSE criterion.

Since the spatial and frequency aspects of the image are taken into account in the wavelet decomposition, the classification and search during the encoding of a subimage vector can be achieved using a simple criterion such as least mean squares. This frees us from using distortion measurements such as weighted least mean squares or other measurements involving perceptual factors. These algorithms are indeed costly in computation time.

C. Optimal Bit Allocation

Multiresolution exploits the eye's masking effects, and therefore, enables us to refine and select the type of coding according to the resolution level and the contour orientation. Although a flat noise shape minimizes the MSE criterion, it is generally not optimal for a subjective quality of image. To apply *noise shaping* across the VQ subimages, we define a total weighted MSE distortion $D_T^*(R_T)$ ((17)) for a total bit rate R_T ((18)).

Let us define $D_{m,d}(R_{m,d})$ the average distortion in the coding of the subimage (m, d) for $R_{m,d}$ bits per pixel:

$$D_{m,d}(R_{m,d}) = E(|x - q(x)|^c) = d(x, q(x)) \qquad c \geq 1 \tag{15}$$

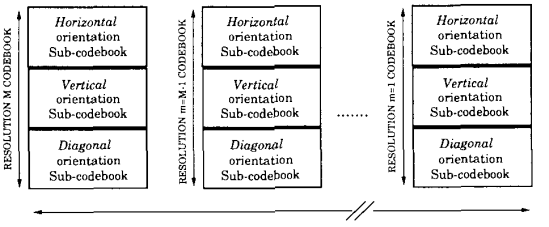


Fig. 12. Multiresolution codebook.

for all coefficients x belonging to the subimage, $q(x)$ being the quantization of x.

Total distortion of the image for a total rate of R_T bits per pixel is then given by:

$$D_T(R_T) = \frac{1}{2^{2M}} D_M^{SQ}(R_M^{SQ}) + \sum_{m=1}^{M} \frac{1}{2^{2m}} \sum_{d=1}^{3} D_{m,d}(R_{m,d}) \tag{16}$$

where $D_M^{SQ}(R_M^{SQ})$ corresponds to the distortion in the subimage of lowest resolution M (texture subimage).

The problem of finding an optimal bit assignment (in bits per pixel) for each subimage vector quantizer is then formulated as:

$$\min_{R_{m,d}} \left[D_T^*(R_T) = \frac{1}{2^{2M}} D_M^{SQ}(R_M^{SQ}) + \sum_{m=1}^{M} \frac{1}{2^{2m}} \sum_{d=1}^{3} D_{m,d}(R_{m,d}) \times B_{m,d} \right] \tag{17}$$

$$\textbf{subject to: } R_T = \frac{1}{2^{2M}} R_M^{SQ} + \sum_{m=1}^{M} \frac{1}{2^{2m}} \sum_{d=1}^{3} R_{m,d} \tag{18}$$

where R_M^{SQ} corresponds to the bit allocation, in bits per pixel, of lowest resolution M subimage.

Assignment of the weights is based on the fact that the human eye is not equally sensitive to signals at all spatial frequencies. On the basis of contrast sensitivity data collected by Campbell and Robson [10], and to obtain a controlled degree of noise shaping across the subimages, we consider a function $B_{m,d}$ such that:

$$B_{m,d} = \gamma^m \log(\sigma_{m,d}^{2\beta_{m,d}}) \tag{19}$$

where $\sigma_{m,d}$ is the standard deviation corresponding to subimage (m, d) and the values of γ and $\beta_{m,d}$ are chosen experimentally in order to match human vision.

$D_T^*(R_T)$ is the total weighted encoding distortion function, and M is the lowest resolution considered.

The expression of $D_{m,d}(R_{m,d})$ is given by [19]

$$D_{m,d}(R_{m,d}) = 2^{-cR_{m,d}} \times \alpha_{m,d}(p, c), \qquad c \geq 1$$

with

$$\alpha_{m,d}(p, c) = A(k_{m,d}, c) \times \left[\int [p_{m,d}(x)]^{k_{m,d}/(c + k_{m,d})} \, dx \right]^{(c + k_{m,d})}. \tag{20}$$

This minimization problem can be solved by using Lagrangian multipliers. Using this technique, we must solve the following equation:

$$\frac{\partial}{\partial R_{m,d}}\left[D_T^*(R_T) - \lambda\left(R_T - \frac{1}{2^{2M}} R_M^{SQ} - \sum_{m=1}^{M} \frac{1}{2^{2m}} \sum_{d=1}^{3} R_{m,d}\right)\right] = 0 \tag{21}$$

where λ is a Lagrangian multiplier.

Using (17) and (20), this equation becomes:

$$\frac{\partial}{\partial R_{m,d}}\left[\frac{1}{2^{2m}} D_M^{SQ}(R_M^{SQ}) + \sum_{m=1}^{M} \frac{1}{2^{2m}} \sum_{d=1}^{3} (2^{-cR_{m,d}} \alpha_{m,d}(p, c) B_{m,d}) - \lambda\left(R_T - \frac{1}{2^{2M}} R_M^{SQ} - \sum_{m=1}^{M} \frac{1}{2^{2m}} \sum_{d=1}^{3} R_{m,d}\right)\right] = 0. \tag{22}$$

Taking the partial derivative with respect to $R_{m,d}$ yields an expression for $R_{m,d}$ in terms of λ:

$$R_{m,d} = \frac{1}{c} \log_2 \left[\frac{(c \ln 2)\alpha_{m,d}(p, c) B_{m,d}}{\lambda}\right]. \tag{23}$$

By substituting (23) into the constraint (18) of the minimization problem we obtain an expression of the Lagrangian multiplier λ

$$\lambda = c \ln 2 \left[2^{-c(R_T - (1/4^M) R_M^{SQ})} \prod_{m=1}^{M} \prod_{d=1}^{3} \cdot [\alpha_{m,d}(p, c) B_{m,d}]^{1/4^m}\right]^{4^M/4^M - 1}. \tag{24}$$

Finally, substituting λ into (23) results in an expression of the optimal bit assignment $R_{m,d_{opt}}$ (in bits per pixel (bpp)) to the vector quantizer of subimage (m, d):

$$R_{m,d_{opt}} = \frac{4^M R_T - R_M^{SQ}}{4^M - 1} + \frac{1}{c} \log_2 \left[\frac{\alpha_{m,d}(p, c) B_{m,d}}{\left[\prod_{m'=1}^{M} \prod_{d'=1}^{3} [\alpha_{m',d'}(p, c) B_{m',d'}]^{1/4^{m'}}\right]^{4^M/4^M - 1}}\right]. \tag{25}$$

This expression requires the knowledge of the subimage's PDF's.

The optimal distortion of the quantizer, $D_{T_{opt}}^*(R_T)$, is then computed by combining (25) and (17). We find:

$$D_{T_{opt}}^*(R_T) = \frac{1}{2^{2M}} D_M^{SQ}(R_M^{SQ}) + \frac{4^M - 1}{4^M} 2^{-c(4^M R_T - R_M^{SQ})/4^M - 1} \cdot \left[\prod_{m=1}^{M} \sum_{d=1}^{3} [\alpha_{m,d}(p, c) B_{m,d}]^{1/4^m}\right]^{4^M/4^M - 1}. \tag{26}$$

Finally, bit allocation which is a function of the image will be transmitted as side information requiring only a few bits.

IV. Experimental Results

The images used are sampled 256 by 256 black and white images. The intensity of each pixel is coded on 256 grey levels (8 bpp).

The numerical evaluation of the coder's performance is achieved by computing the peak signal-to-noise ratio (PSNR) between the original image and the coded image.

For each coded image, we can use a variable length code. We also give the corresponding $\mathcal{R}_T$ if an optimal entropy coding was performed, defined as follows.

To the L codewords w_j; $j = 1, 2, \cdots, L$ of the vector quantizer corresponds to L regions (clusters) of $\mathbb{R}^k$, $\mathcal{P}_j$; $j = 1, 2, \cdots, L$. The jth region is defined by

$$\mathcal{P}_j = \{x \in \mathbb{R}^k / Q(x) = w_j\}$$

and represents the subset of vectors of $\mathbb{R}^k$ which are well matched by the codeword w_j of the codebook.

Thus for each resolution and direction, we can introduce the average information of the codebook, called the entropy measure:

$$\mathcal{R}_{m,d} = -\frac{1}{k_{m,d}} \times \sum_{j=1}^{L} p(w_j) \log_2 p(w_j) \text{ bpp}$$

where $p(w_j)$ is the probability of selecting the source vector w_j, belonging to the codebook at scale m and corresponding to the orientation d, during the coding of the image (m, d).

Then, as in (18), $\mathcal{R}_T$ is the sum of the estimated entropy in each subimage as follows:

$$\mathcal{R}_T = \frac{1}{2^{2M}} \mathcal{R}_M^{SQ} + \sum_{m=1}^{M} \frac{1}{2^{2m}} \sum_{d=1}^{3} \mathcal{R}_{m,d} \text{ bpp}.$$

The vector quantizer used is a *full search* quantizer, i.e., during the coding, all of the vectors in the subcodebook corresponding to the resolution and direction to be encoded are searched. The selection criterion used is the MSE criterion.

A. Comparison Between the Different Wavelets

In the following, we present results obtained with the Lena image (image within the training set) for a real bit rate of 1 bpp and using the three different filters proposed in Section II-B. (Fig. 13 corresponds to filters 9-3 presented in example 1, Fig. 14 corresponds to filters 9-7 presented in example 2, and Fig. 15 corresponds to filters 5-7 presented in example 3.) Here, the Lena image is taken as part of the training set in order to minimize the effects of quantization noise: this enables the influence of the filters to be taken into account.

For a given set of filters, separate codebooks are trained for each resolution-orientation subimage, and bit alloca-

Fig. 13. Filters no. 1 9-3, PSNR = 31.82 dB, $\mathcal{R}_T = 0.80$ bpp.

Fig. 15. Filters no. 3, 5-7, PSNR = 31.46 dB, $\mathcal{R}_T = 0.80$ bpp.

Fig. 14. Filters no. 2, 9-7, PSNR = 32.10 dB, $\mathcal{R}_T = 0.78$ bpp.

Fig. 16. Original 256 by 256 Lena, 8 bpp.

tion is carried out according to (25). For the Lena image, the bit assignment is represented in Fig. 17. Resolution 1 (diagonal orientation) is discarded. Resolution 1 (horizontal and vertical orientations) and resolution 2 (diagonal orientation) are coded using 256-vector codebooks (codeword size 4 by 4) resulting in a 0.5-b/pixel rate, while resolution 2 (horizontal and vertical orientations) is coded at a 2-b/pixel rate using 256-vector codebooks (codeword size 2 by 2). Finally, the lowest resolution is coded at 8 b/pixel.

B. Results as a Function of Regularity and Vanishing Moments

In Section II-B, we mentioned our belief that both the regularity of the reconstruction wavelet $\tilde{\psi}$ and the number

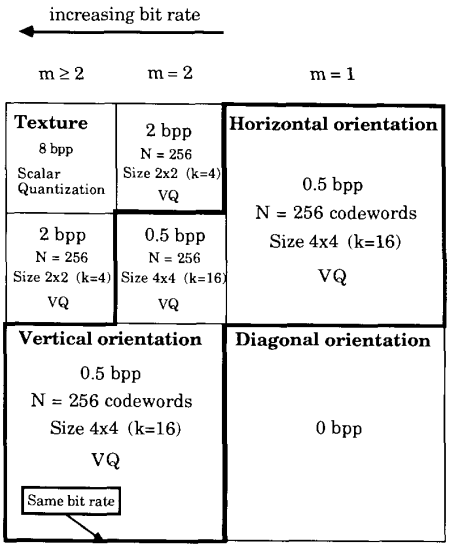


Fig. 17. Subimages bit rate allocation: example of a bit allocation for a total bit rate of 1 bpp and for the 256 by 256 Lena image.

of vanishing moments of the analyzing wavelet ψ are important in applications. To illustrate this we carried out the following experiments. For a given pair, h, $\tilde{h}$, we analyzed the same image twice: once as described above, and a second time after exchanging the roles of the filters h and $\tilde{h}$.

The filter pairs in example 2 both have the same number of vanishing moments, $k = \tilde{k} = 4$. However, $\tilde{\psi}$ is considerably more regular than ψ (see Fig. 3). With this filter pair, our experiment on the Lena image led to a PSNR of 32.10 dB in the first case, and to a PSNR of 31.51 dB if the roles of h and $\tilde{h}$ are inverted. The case where the reconstruction wavelet has the highest regularity, therefore, performs best.

In example 1 the functions ψ and $\tilde{\psi}$ have comparable regularity: both are continuous and neither has a continuous derivative. In fact $\tilde{\psi}$ is a bit more regular than ψ: $\tilde{\psi}$ is differentiable almost everywhere, and is Hölder continuous with exponent 1, while ψ is Hölder continuous with the exponent only at 0.83. On the other hand, ψ has 2 vanishing moments, while $\tilde{\psi}$ has 4 ($k = 4$, $\tilde{k} = 2$). The same experiment, again with the Lena image, now leads to a PSNR of 31.82 dB if h, $\tilde{h}$ are taken as in Table I, and to a PSNR of 31.13 dB when the roles of h and $\tilde{h}$ are reversed. The situation where $\tilde{\psi}$ is most regular but ψ has fewer vanishing moments, therefore, performs better (gain of 0.69 dB) than the case where ψ has more vanishing moments but $\tilde{\psi}$ is less regular. This seems to suggest that the regularity of $\tilde{\psi}$ has a larger effect than the number of vanishing moments of ψ. However, in this example the difference in overall regularity, as measured by the differences between Hölder exponents, is much smaller here than in example 2 (0.17 as compared to 0.63 in example 2), and it seems hard to explain how this smaller difference in Hölder exponent could account for a comparable gain in PSNR. In fact, the Hölder exponent is not a very good measure for the regularity of $\tilde{\psi}$ in this case: it is completely determined by the discontinuity of the derivative of $\tilde{\psi}$ in only a few points, and it is insensitive to the fact that $\tilde{\psi}$ is infinitely differentiable in all other points. If this is taken into account, then $\tilde{\psi}$ looks much more regular than ψ (the Hölder exponent of which is determined by its behavior near a dense set of points), which might explain the gain in PSNR.

We conclude from all this that: 1) for the same number of vanishing moments for ψ, the scheme with most regular $\tilde{\psi}$ is likely to perform best; and 2) increasing the regularity of $\tilde{\psi}$, even at the expense of the number of vanishing moments for ψ, may lead to better results.

Based on theoretical arguments (Taylor expansions) and results from numerical analysis [8], we also expect: 3) for comparable regularity of $\tilde{\psi}$, the scheme with largest vanishing moments for ψ is likely to perform best.

C. Comparison with Other Coders

If the PSNR is chosen as a criterion of comparison, these results are close to those obtained by Woods and O'Neil [42] and Westerink *et al.* [40]. However, in their subband coding algorithm, they use 32-taps Johnston filters, while only 9 or 7 taps are necessary for our method. According to Westerink's results in [41], the PSNR decreases by about 2 dB when using 8-taps Johnston filters. However, some others new QMF designs can also lead to good results with about 9 taps for image coding [1].

In this section, we present both numerical and qualitative comparison between our coding scheme and other previously published results. Since the most popular image in the recent literature has been the 512 by 512 Lena image, the comparison is made using this image taken outside the training set.

Among the different methods published, we consider the three following well-known methods: Ho and Gersho obtained a 30.93-dB PSNR at 0.36 bpp, result using "variable-rate multi stage VQ" [23]. Riskin and Gray improved on the full search VQ (PSNR = 29.29 dB, 0.32 bpp) using pruned tree structured VQ (PSNR = 30.92 dB, 0.32 bpp) [34]. High PSNR values were obtained by Woods and Cohen using entropy coded and predictive VQ (PSNR = 32.5 dB, 0.45 bpp) [11].

Our aim is not to optimize the PSNR but rather a weighted function of the MSE in order to match human vision. We give two examples at low bit rate using wavelet VQ.

Our initial result at 0.37 bpp presented Fig. 18 with a 30.85-dB PSNR is very close to those of Ho and Gersho [23] and Riskin *et al.* [34]. The perceptual quality of our coded images is better than indicated by the PSNR value

Fig. 18. 512 by 512 Lena image. Filters no. 2 9-7, PSNR = 30.85 dB, $\mathcal{R}_T = 0.37$ bpp.

Fig. 19. 512 by 512 Lena image. Filters no. 2 9-7, PSNR = 29.11 dB, $\mathcal{P}_T = 0.21$ bpp.

mainly due to the regularity of the wavelet and the bit allocation. These images do not suffer from the blocking effects obtained when using VQ in the spatial domain. No ringing effects can be observed.

The second result at 0.21 bpp presented in Fig. 19 with a 29.11-dB PSNR shows that a very low bit rate can be achieved with our method, without severe degradation.

Our method using a new class of filters derived from wavelet theory using full search VQ can be improved by any of the three above-mentioned methods.

In fact the LBG clustering algorithm is a very simple algorithm but not optimal for variable length code. The PSNR of the method could be improved by about 3 dB, for example, using ECVQ [34] but CPU time becomes prohibitively expensive.

D. Progressive Transmission Scheme

The main objective of progressive transmission is to allow the receiver to recognize a picture as quickly as possible at minimum cost, by sending a low resolution level picture first. Then, it can be decided to receiver further picture details or to abort the transmission. Further details of the picture are obtained by sequentially receiving the encoded wavelet coefficients at different resolution levels and directions.

Following the example of [40], we will display each picture level during the progressive transmission with a size that matches the resolution of that particular level.

To test the efficiency of the vector quantizer, the image to be coded is taken outside the training set.

Fig. 20 represents 5 stages in the progressive transmission of a 256 by 256 image using filters 9–7 given in example 2. According to the bit allocation procedure (Section III-C) with a generalized Gaussian PDF approximation law, only the wavelet coefficients corresponding to the $m = 1$ and $m = 2$ high resolution levels are vector quantized, while the low level subimages ($m \geq 2$) are scalar quantized.

V. Conclusion

This paper describes a new image coding scheme combining the wavelet transform and VQ.

A new family of filters has been derived from the wavelet theory. We have shown the importance of regularity and vanishing moments for image coding. Furthermore, these filters require few taps, unlike standard QMF methods.

The wavelet transform used here attempts to exploit the masking effect of the human eye, yielding encouraging results. Indeed, the proposed method enables high compression bit rates while maintaining good visual quality through the use of bit allocation in the subimages. The blocking effects seen when spatial VQ is performed are avoided.

This method is well adapted to progressive transmission as well as very low bit rate compression. Furthermore, using a simple full-search VQ provides good results, comparable to the best results published currently.

Further research should include some new derivation such as entropy constraint and predictive VQ. We would improve this coding scheme, if we accept a heavier computational load.

Resolution	*m=4*	*m=3*	*m=2*	*m=1*
Size	**16 × 16 pix**	**32 × 32 pix**	**64 × 64 pix**	**128 × 128 pix**
R_T	0.031 bpp	0.125 bpp	0.5 bpp	0.781 bpp
$\mathcal{R}_T$	0.0264 bpp	0.0919 bpp	0.3354 bpp	0.5039 bpp

Resolution m=0

256 × 256 pix

R_T = 1 bpp $\mathcal{R}_T$ = 0.6297 bpp

PSNR = 31.28 dB

Fig. 20. Progressive transmission-filters no. 2 9-7.

REFERENCES

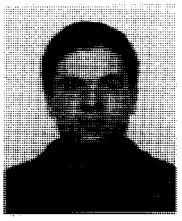
[1] E. H. Adelson and E. Simoncelli, "Non-separable extensions of quadrature mirror filters to multiple dimensions," *Proc. IEEE*, vol. 78, Apr. 1990.

[2] M. Abramowitz, I. A. Stegun, *Handbook of Mathematical Functions*. New York: Dover, 1965.

[3] V. R. Algazi, "Useful approximation to optimum quantization," *IEEE Trans. Commun.*, vol. COM-14, pp. 297-301, June 1966.

[4] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image

coding using vector quantization in the wavelet transform domain," in *Proc. IEEE ICASSP*, April 1990, pp. 2297-2300.
[5] M. Barlaud, L. Blanc-Féraud, P. Mathieu, J. Menez, and M. Antonini, "2D linear predictive image coding with vector quantization," in *Proc. EUSIPCO*, Grenoble, France, Sept. 5-8, 1988, pp. 1637-1640.
[6] M. Barlaud, P. Mathieu, and M. Antonini, "Wavelet transform image coding using vector quantization," presented at 6th Workshop on MDSP, Monterey, CA, Sept. 1989.
[7] G. Battle, "A block spin construction of wavelets. Part I Lemarié functions," *Comm. Math. Phys.*, vol. 110, pp. 601-615, 1987.
[8] G. Beylkin, R. Coifman, and V. Rokhlin, "Fast wavelet transforms and numerical analysis. I," to be published.
[9] P. Burt and E. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Trans. Commun.*, vol. 31, pp. 482-540, 1983.
[10] F. W. Campbell and J. G. Robson, "Application of Fourier analysis to the visibility of gratings," *J. Phys.*, vol. 197, pp. 551-566, 1968.
[11] R. A. Cohen and J. W. Woods, "Sliding block entropy coding of images," in *Proc. IEEE ICASSP*, Glasgow, Scotland, May 23-26, 1989, pp. 1731-1733.
[12] A. Cohen, I. Daubechies, and J. C. Feauveau, "Biorthogonal bases of compactly supported wavelets," AT&T Bell Lab., Tech. Rep., TM 11217-900529-07, 1990.
[13] J. H. Conway and N. J. A. Sloane, "A lower bound on the average error of vector quantizers," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 106-109, Jan. 1985.
[14] I. Daubechies, A. Grossman, and Y. Meyer, "Painless nonorthogonal expansions," *J. Math. Phys.*, vol. 27, pp. 1271-1283, 1986.
[15] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," to be published.
[16] ——, "Orthonormal bases of compactly supported wavelets," *Comm. Pure Appl. Math.*, vol. 41, pp. 909-996, 1988.
[17] ——, "Orthonormal bases of compactly supported wavelets. II. Variations on a theme," AT&T Bell Lab., Tech. Rep. TM 11217-891116-17, 1990.
[18] J. C. Feauveau, "Analyse multirésolution par ondelettes non orthogonales et bancs de filtres numériques," Ph.D. dissertation, Univ. Paris Sud, France, Jan. 1990.
[19] A. Gersho, "Asymptotically optimal block quantization," *IEEE Trans. Inform. Theory*, vol. IT-25, July 1979.
[20] ——, "On the structure of vector quantizers," *IEEE Trans. Inform. Theory*, vol. IT-28, Mar. 1982.
[21] R. M. Gray, "Vector quantization," *IEEE ASSP Mag.*, pp. 4-29, Apr. 1984.
[22] A. Grossman and J. Morlet, "Decomposition of hardy functions into square integrable wavelets of constant shape," *SIAM J. Math Anal.*, vol. 15, pp. 723-736, 1984.
[23] Y. Ho and A. Gersho, "Variable-rate multi-stage vector quantization for image coding," in *Proc. IEEE ICASSP*, New York, Apr. 1988.
[24] P. G. Lemarié, "Une nouvelle base d'ondelettes de $L^2(\mathbb{R})$," *J. Math. Pures et Appl.*, vol. 67, pp. 227-238, 1988.
[25] P. G. Lemarié and Y. Meyer, "Ondelettes et bases hilbertiennes," *Rev. Mat. Iberoamericana*, vol. 2, pp. 1-18, 1986.
[26] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84-95, Jan. 1980.
[27] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Mach. Intel.*, vol. 11, July 89.
[28] D. Marr, *Vision*. New York: Freeman, 1982.
[29] P. Mathieu, M. Barlaud, and M. Antonini, "Compression d'Images par transformée en ondelette," *12ième colloque GRETSI, Juan les Pins*, June 12-16, 1989.
[30] P. Mathieu, M. Barlaud, and M. Antonini, "Compression d'Image par transformée en ondelette et quantification vectorielle," *Traitement du Signal*, vol. 7, no. 2, 1990.
[31] Y. Meyer, "Principe d'incertitude, bases hilbertiennes et algèbres d'opérateurs," *Seminaire Bourbaki*, no. 662, 1985-1986.
[32] N. M. Nasrabadi and R. A. King, "Image coding using vector quantization: A review," *IEEE Trans. Commun.*, vol. 36, Aug. 1988.
[33] W. K. Pratt, *Digital Image Processing*. New York: Wiley, 1978.
[34] E. Riskin, E. M. Daly, and R. M. Gray, "Pruned tree-structured vector quantization in image coding," in *Proc. IEEE ICASSP*, Glasgow, Scotland, May 1989, pp. 1735-1738.
[35] M. J. Smith and D. P. Barnwell, "Exact reconstruction for tree-structured subband coders," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. ASSP-34, pp. 434-441, 1986.
[36] J. O. Stromberg, "A modified haar system and higher order spline systems," in *Conf. in Harmonic Analysis in Honor of Antoni Zygmund*, Vol. II, pp. 475-493.
[37] M. Vetterli, "Splitting a signal into subsampled channels allowing perfect reconstruction," in *Proc. IASTED Conf. Appl. Signal Processing Digital Filtering*, Paris, France, June 1985.
[38] M. Vetterli and C. Herley, "Wavelets and filter banks: Relationships and new results," in *Proc. IEEE ICASSP*, Albuquerque, Apr. 1990.
[39] P. H. Westerink, D. E. Boekee, J. Biemond, and J. W. Woods, "Subband coding of image using vector quantization," *IEEE Trans. Commun.*, vo. 36, pp. 713-719, 1988.
[40] P. H. Westerink, J. Biemond, and D. E. Boekee, "Progressive transmission of images using subband coding," in *Proc. IEEE ICASSP*, 1989, pp. 1811-1814.
[41] P. H. Westerink, "Subband coding of images," Ph.D. dissertation Delft Univ., 1989.
[42] J. W. Woods and S. D. O'Neil, "Subband coding of images," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. ASSP-34, Oct. 1986.
[43] P. Zador, "Asymptotic quantization error of continuous signals and their quantization dimension," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 139-149, 1982.



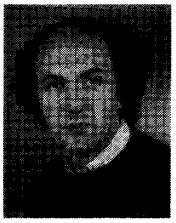
Marc Antonini was born in France on August 29, 1965. He received the DEA degree in signal processing in 1988 from the University of Nice-Sophia Antipolis, France, and the Ph.D. degree from the Laboratory of Signaux et Systèmes, URA I3S, CNRS and the University of Nice-Sophia Antipolis in 1991.

His research interests include multidimensional image processing, wavelet analysis, and image coding.

Michel Barlaud (M'88) was born in France on November 24, 1945. He received the "Doctorat d'Etat" degree from University of Paris XII.

He is currently a Professor and a member of the Laboratory of Signaux et Systèmes, URA I3S both from CNRS and University of Nice-Sophia Antipolis. After some work on non-stationary signal processing, his research interests move towards multidimensional image processing, wavelet analysis, image coding, inverse problems, image restoration, and edge detection.

Dr. Barlaud is member of the IEEE-ASSP MDSP committee.



Pierre Mathieu was born in Alger on May 10, 1956. He received the Ingenieur ENSEEIHT and Ph.D. degrees from INP Toulouse.

He is currently Maître de Conférences in the Laboratory of Signaux et Systèmes, URA I3S both from CNRS and University of Nice-Sophia Antipolis. His research interests include multidimensional image processing, wavelet analysis, image coding, and image restoration.

Ingrid Daubechies (M'89) received the B.S. and Ph.D. degrees from the Vrje Universiteit Brussel, Belgium in 1975 and 1980, both in physics.

She is currently a Member of Technical Staff in the Mathematics Center of AT&T Bell Laboratories, Murray Hill, NJ. Her current research interests include mathematical problems in connection with signal analysis, in particular applications of time-frequency representations.

US005321520A

United States Patent [19]

Inga et al.

[11] Patent Number: 5,321,520

[45] Date of Patent: Jun. 14, 1994

[54] **AUTOMATED HIGH DEFINITION/RESOLUTION IMAGE STORAGE, RETRIEVAL AND TRANSMISSION SYSTEM**

[75] Inventors: **Jorge J. Inga; Thomas V. Saliga,** both of Tampa, Fla.

[73] Assignee: **Automated Medical Access Corporation,** Tampa, Fla.

[21] Appl. No.: **915,298**

[22] Filed: **Jul. 20, 1992**

[51] **Int. Cl.**[5] **H04N 1/21;** H04N 1/41

[52] **U.S. Cl.** **358/403;** 358/426; 358/428

[58] **Field of Search** 358/403, 426, 261.1, 358/444, 428; 395/114

[56] **References Cited**

U.S. PATENT DOCUMENTS

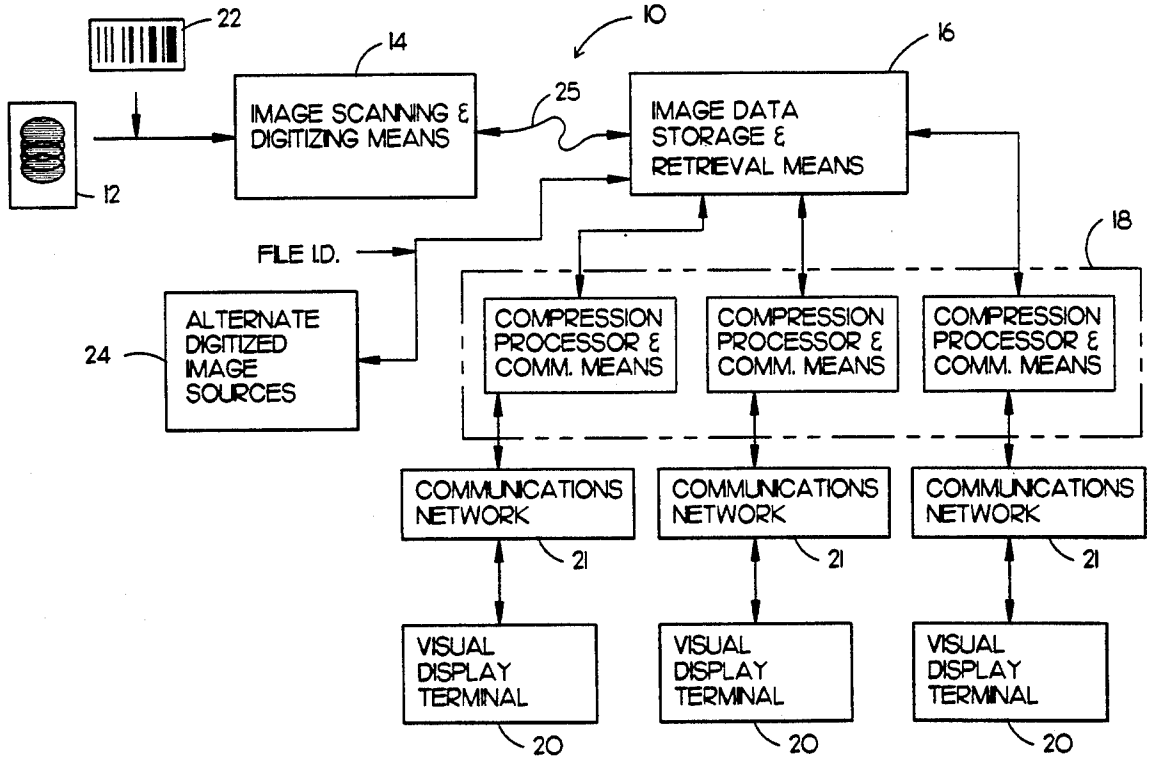
4,719,514	1/1988	Kurahayashi et al.	358/404
4,768,099	8/1988	Mukai	358/426
4,933,025	2/1991	Vesel et al.	370/94.1
4,958,283	9/1990	Tawara et al.	364/413.3
5,068,745	11/1991	Shimura	358/426
5,111,044	5/1992	Agano	250/327.2
5,111,306	5/1992	Kanno et al.	358/426
5,170,266	12/1992	Marsh et al.	358/426

Primary Examiner—Edward L. Coles, Sr.
Assistant Examiner—Scott A. Rogers
Attorney, Agent, or Firm—David Kiewit

[57] **ABSTRACT**

An automated high definition/resolution image storage, retrieval and transmission system for use with medical X-ray film or other documents to provide simultaneous automated access to a common data base by a plurality of remote subscribers upon request, the automated high definition/resolution image storage, retrieval and transmission system comprising an image scanning and digitizing subsystem to scan and digitize visual image information from an image film or the like; an image data storage and retrieval subsystem to receive and store the digitized information and to selectively provide the digitized information upon request from a remote site, a telecommunication subsystem to selectively transmit the requested digitized information from the image data storage and retrieval subsystem to the requesting remote visual display terminal for conversion to a visual image at the remote site to visually display the requested information from the image data storage and retrieval subsystem.

12 Claims, 9 Drawing Sheets



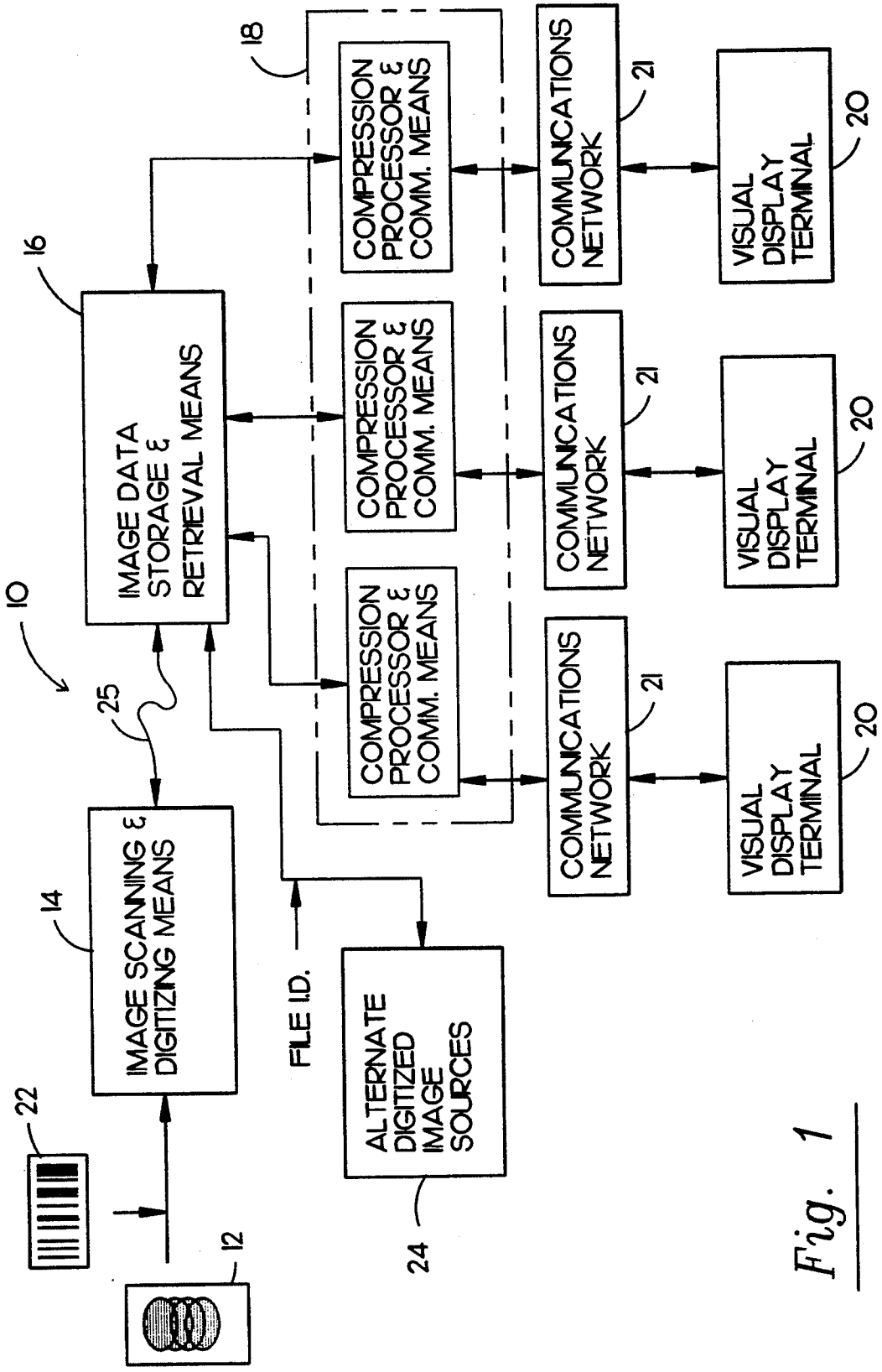

10
12
14
16
18
20
21
22
24
25
IMAGE SCANNING & DIGITIZING MEANS
IMAGE DATA STORAGE & RETRIEVAL MEANS
COMPRESSION PROCESSOR & COMM. MEANS
COMPRESSION PROCESSOR & COMM. MEANS
COMPRESSION PROCESSOR & COMM. MEANS
COMMUNICATIONS NETWORK
COMMUNICATIONS NETWORK
COMMUNICATIONS NETWORK
VISUAL DISPLAY TERMINAL
VISUAL DISPLAY TERMINAL
VISUAL DISPLAY TERMINAL
FILE I.D.
ALTERNATE DIGITIZED IMAGE SOURCES

Fig. 1

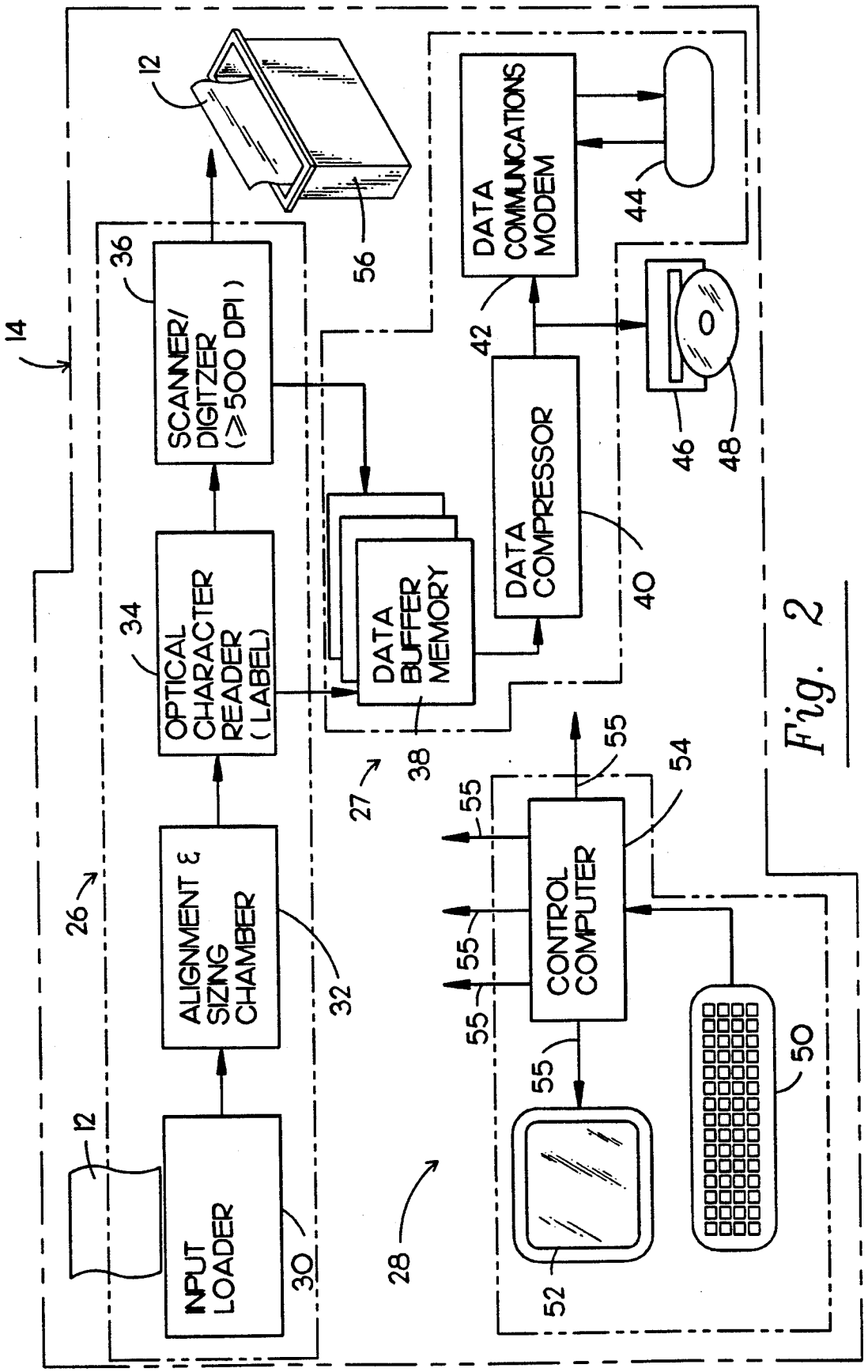

12
56
DATA COMMUNICATIONS MODEM
44
42
SCANNER/ DIGITZER (≥ 500 DPI)
36
14
46
48
DATA COMPRESSOR
40
OPTICAL CHARACTER READER (LABEL)
34
DATA BUFFER MEMORY
38
27
55
54
55
55
55
CONTROL COMPUTER
26
ALIGNMENT & SIZING CHAMBER
32
50
12
INPUT LOADER
30
28
52

Fig. 2

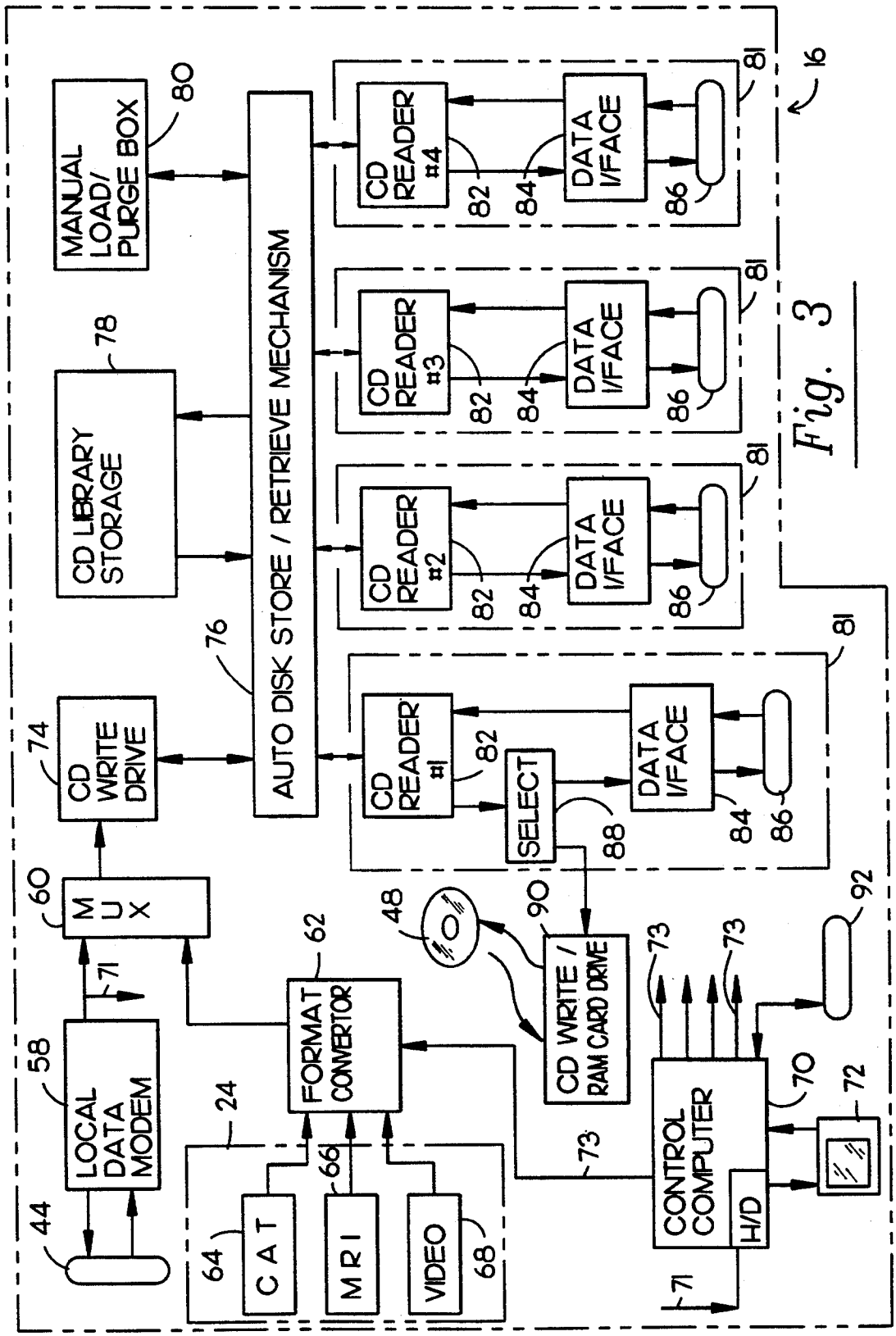

MANUAL LOAD/ PURGE BOX
80
CD LIBRARY STORAGE
78
AUTO DISK STORE / RETRIEVE MECHANISM
76
CD WRITE DRIVE
74
M U X
60
LOCAL DATA MODEM
58
44
71
FORMAT CONVERTOR
62
24
66
CAT
64
MRI
VIDEO
68
48
90
CD WRITE / RAM CARD DRIVE
73
CONTROL COMPUTER
H/D
70
72
92
CD READER #1
CD READER #2
CD READER #3
CD READER #4
SELECT
88
DATA I/FACE
82
84
86
81
16

Fig. 3

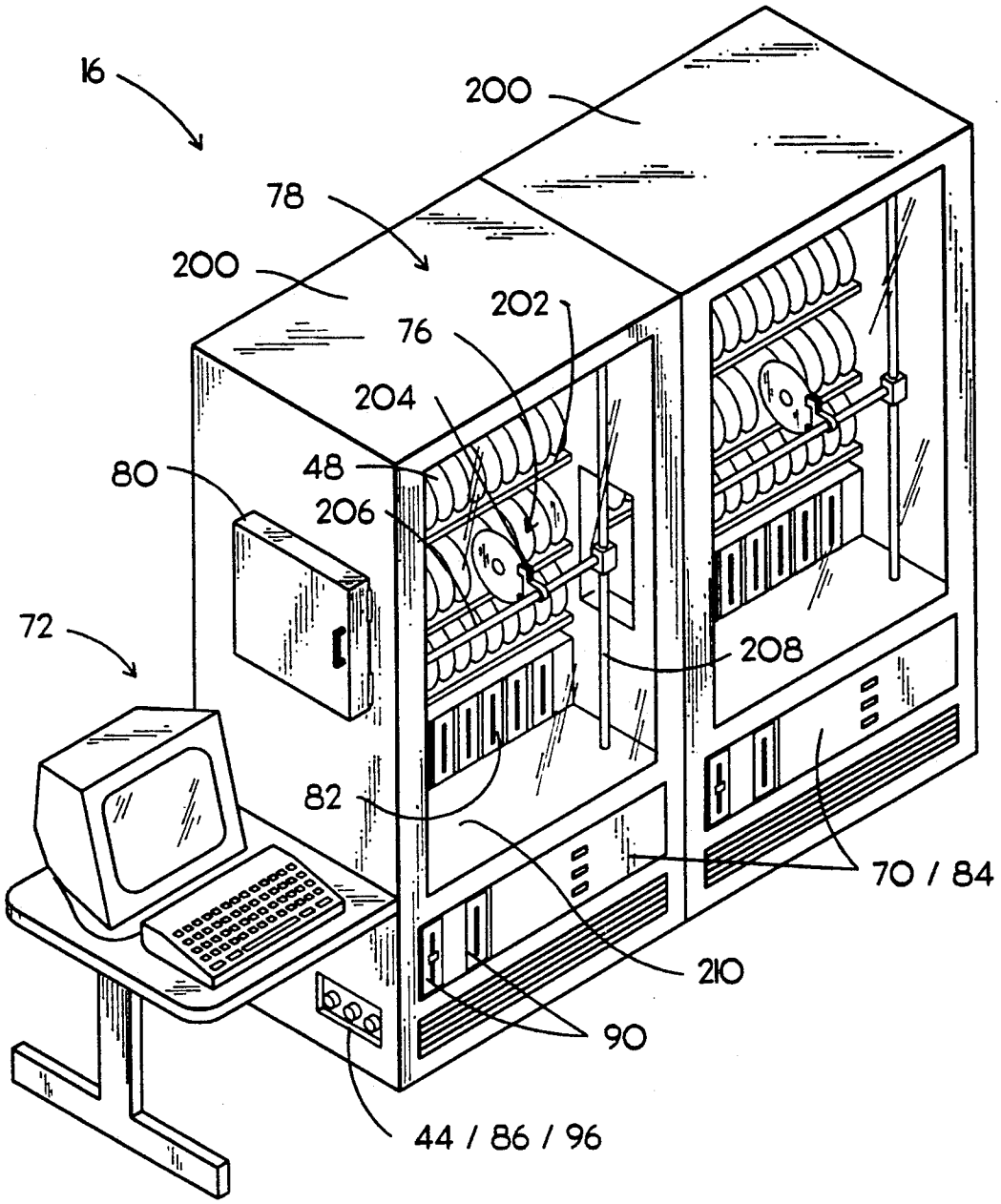


Fig. 4

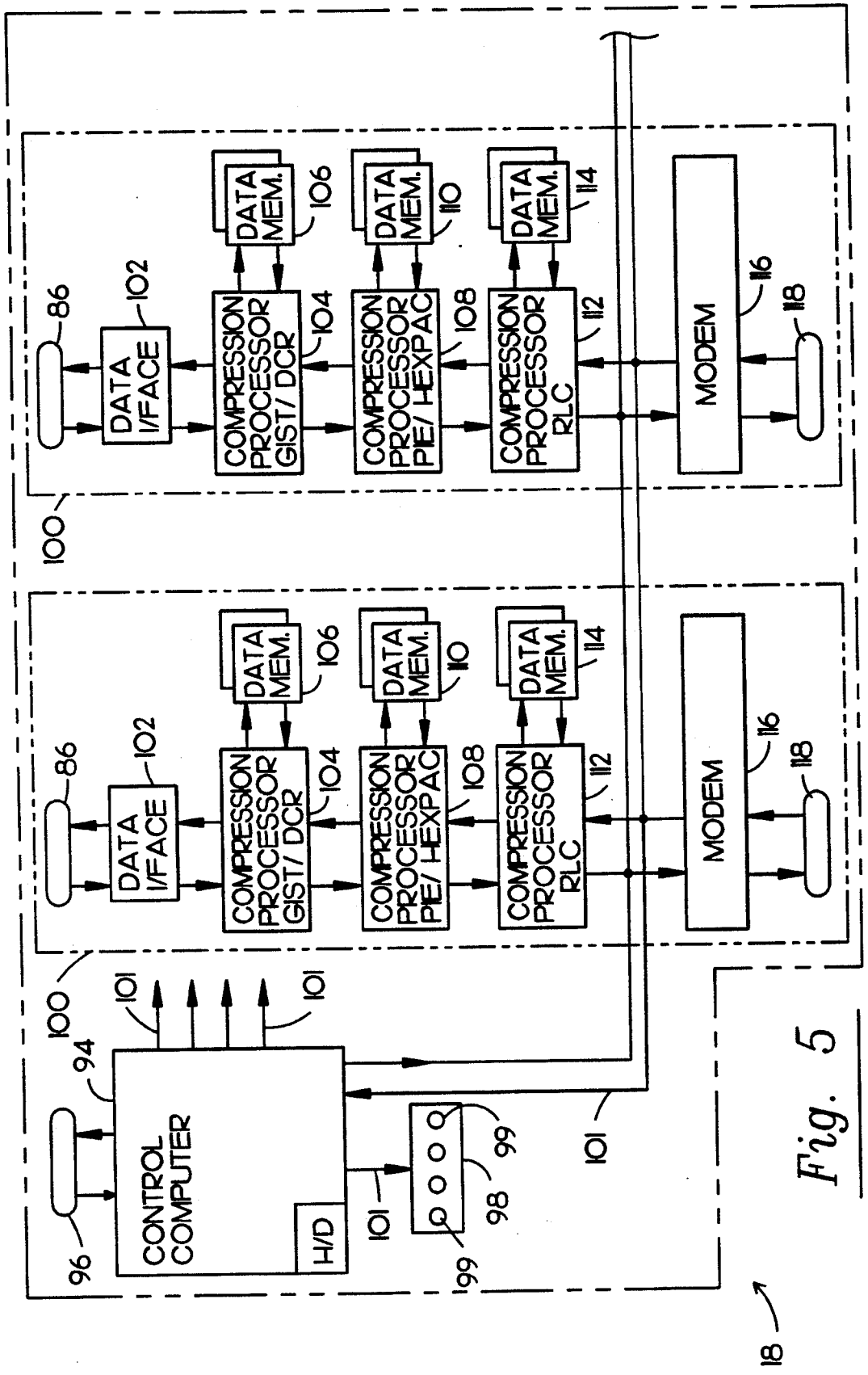

DATA MEM.
106
110
114
86
102
DATA I/FACE
COMPRESSION PROCESSOR GIST/ DCR
104
COMPRESSION PROCESSOR PE/ HEXPAC
108
COMPRESSION PROCESSOR RLC
112
MODEM
116
118
100
101
94
CONTROL COMPUTER
H/D
96
98
99
18

Fig. 5

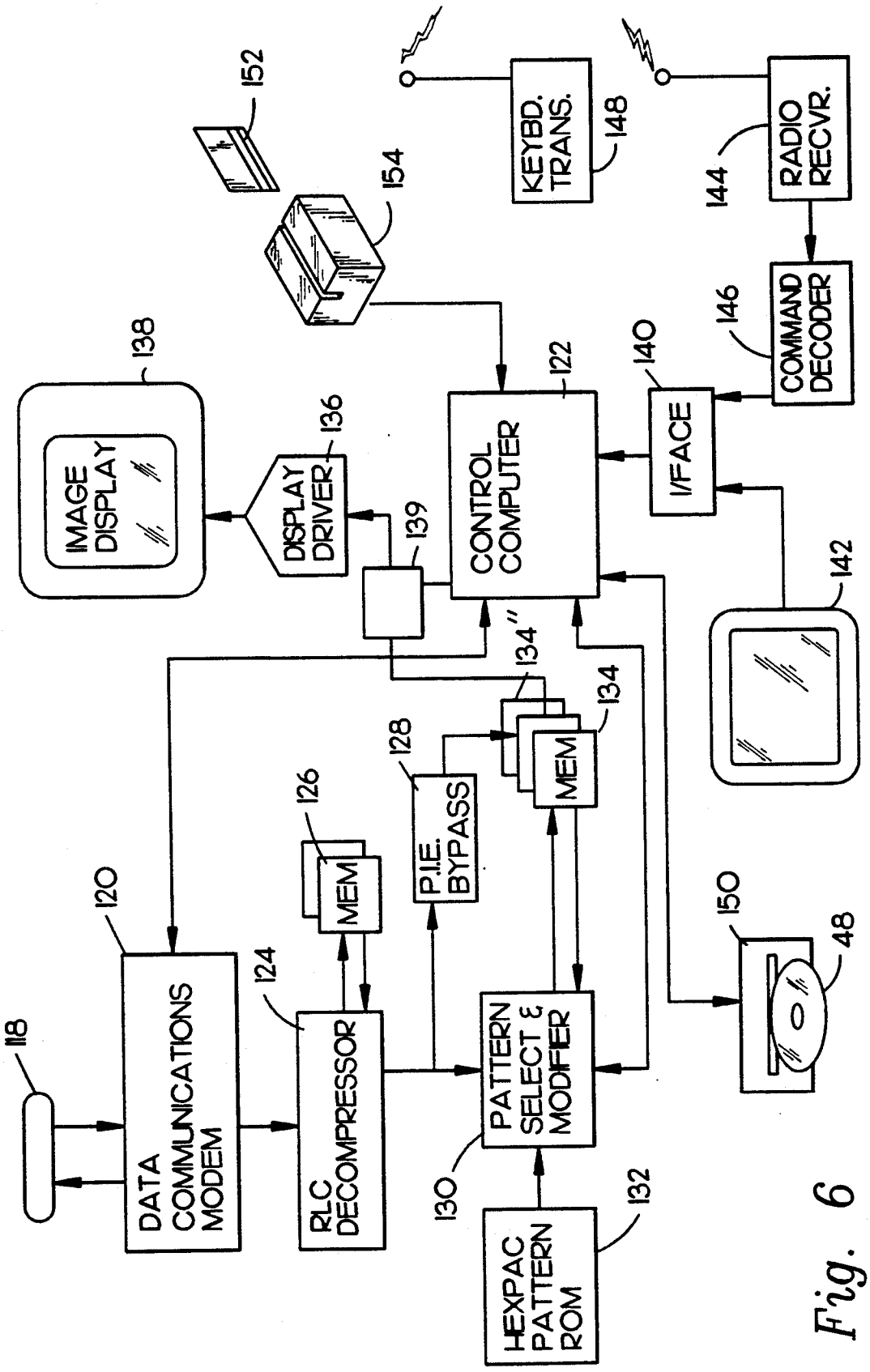

152
154
KEYBD. TRANS.
148
RADIO RECVR.
144
COMMAND DECODER
146
140
I/FACE
122
CONTROL COMPUTER
138
IMAGE DISPLAY
DISPLAY DRIVER
136
139
134"
134
MEM
128
P.I.E. BYPASS
126
MEM
120
DATA COMMUNICATIONS MODEM
124
RLC DECOMPRESSOR
PATTERN SELECT & MODIFIER
130
HEXPAC PATTERN ROM
132
118
142
150
48

Fig. 6

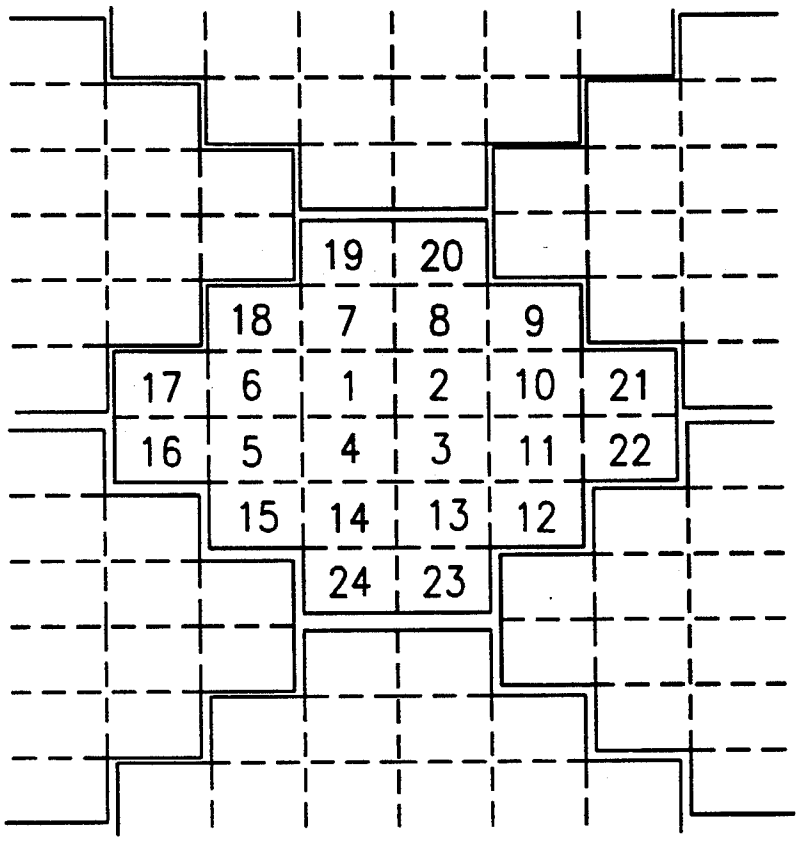


Fig. 7

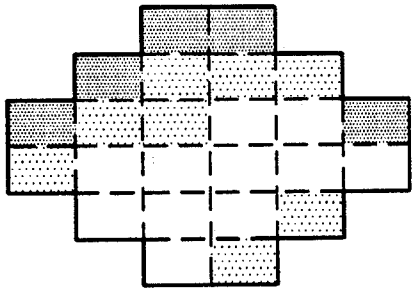


Fig. 8

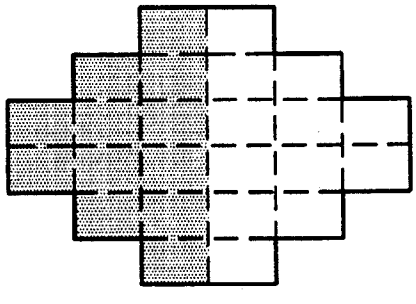


Fig. 9

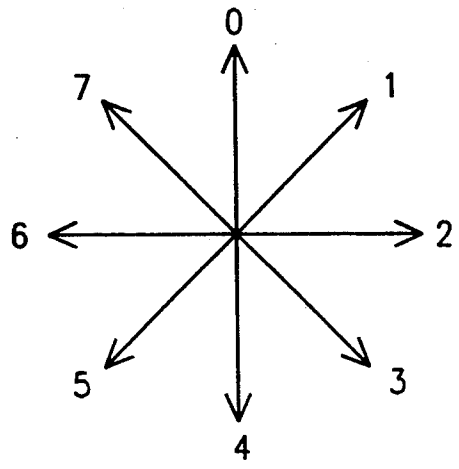


Fig. 10

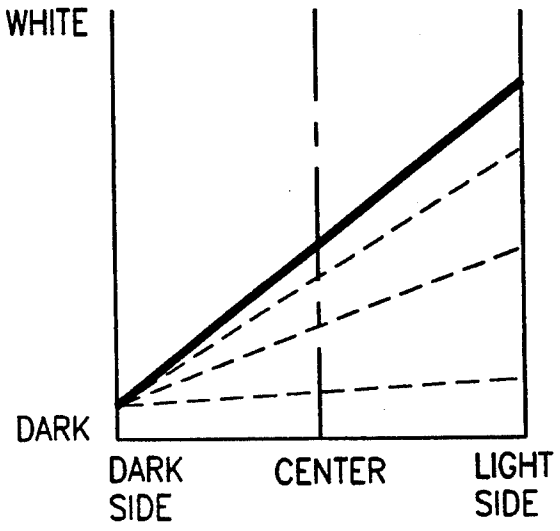


Fig. 11

Fig. 12

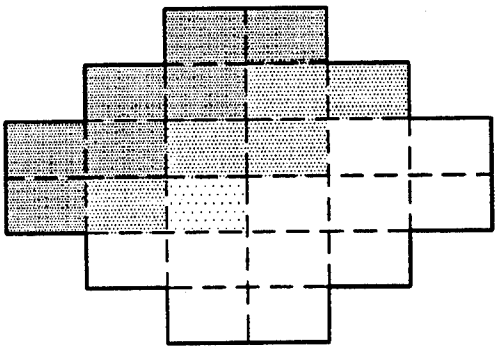


Fig. 13

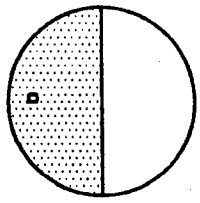


Fig. 14-A

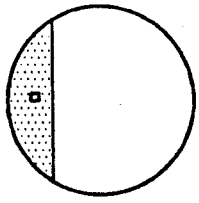


Fig. 14-B

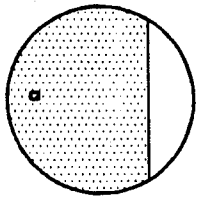


Fig. 14-C

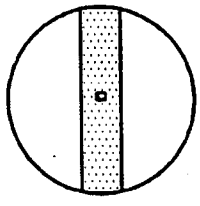


Fig. 14-D

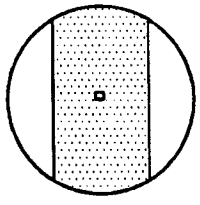


Fig. 14-E

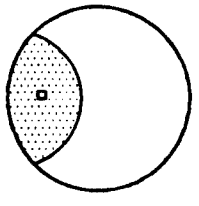


Fig. 14-F

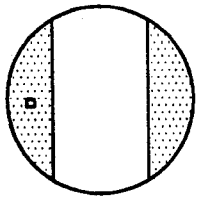


Fig. 14-G

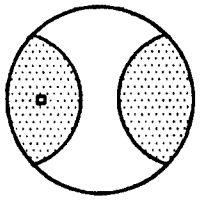


Fig. 14-H

AUTOMATED HIGH DEFINITION/RESOLUTION IMAGE STORAGE, RETRIEVAL AND TRANSMISSION SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

An automated high definition/resolution image storage, retrieval and transmission system for use with medical X-ray films and the like.

2. Description of the Prior Art

Many industries have their own unique inventory control means for the particular needs of the respective trade. The food industry, for example, has largely standardized on bar-code technology to track food items. The medical profession, however, in many areas has not kept technological pace notwithstanding the obvious need therefor. Storage and retrieval systems for medical image data such as X-ray films, CAT scans, angiograms, tomograms and MRI are commonly antiquated and often employing methods of the 1920's. For example, when image films are used in the physician must display these photo films on a light box.

Moreover, due to the diffuse responsibilities of multiple attending physicians and treatment sites involving patients with particularly complex conditions, image data for such patients is often misplaced, lost, or at best, difficult to find when needed. Hospitals maintain large "file rooms" to store the patient image data. The film image data is typically stored in a large brown envelope approximately 14 by 17 inches and open at one end. These become bulky to handle and store especially in a complex situation in which several of these folders are needed. Weight alone can build up to 15 pounds. It has proven time consuming to obtain image data from file rooms either due to administrative backlogs, lack of specialized filing personnel and misfiling.

Typically, the physician examines the patient in his office after the radiographical studies have been made in a hospital or diagnostic facility. These films and the information contained therein are often unavailable at the time of the examination. Thus, there is a need for remote access to these image data for rapid patient assessment and therapy recommendation.

U.S. Pat. No. 4,603,254 teaches a stimulable phosphor sheet carrying a radiation image stored therein scanned with stimulating rays. The light emitted from the stimulable phosphor sheet in proportion to the radiation energy stored therein is detected and converted into an electric signal converted to a digital signal. Digital data is created to reproduce the radiation image for use in diagnosis and storage.

U.S. Pat. No. 4,764,870 describes a system for transferring medical diagnostic information from the diagnostic site to remote stations. An internal analog video signal from imaging diagnostic equipment such as a CAT scanner or MRI equipment, is converted to an analog video signal of different, preferably standard, format that is stored and transmitted in the reformatted image information to the remote terminal. The received signal is stored, decoded and applied in appropriate analog video form to an associated CRT display for reproduction of the diagnostic images.

U.S. Pat. No. 5,005,126 shows a system for transferring medical diagnostic information from the diagnostic site to remote stations similar to that found in U.S. Pat. No. 4,764,870.

U.S. Pat. No. 5,019,975 teaches a method for constructing a data base in a medical image filing system comprising the steps or recording information indicating the time at which each medical image is recorded and a rank of importance for each medical image as image retrieval signal data for image signals corresponding to each medical image; recording the number of times the image signals corresponding to each medical image have been retrieved as image retrieval signal data and incrementing the number each time the image signals are retrieved; and when the data base is full of image retrieval signal data, deleting the image retrieval signal data corresponding to the image signals of the medical image in which at least (1) the time at which the medical image was recorded earlier than a predetermined time and (2) the rank of importance of the medical image is lower than a predetermined value.

U.S. Pat. No. 4,611,247 describes a radiation image reproducing apparatus to read a radiation image from a first recording medium as a visible image. Input devices of the apparatus enter data which are associated with a method of exposing an object to a radiation and object's exposed part. In response to the input data, a processing condition determining unit determines conditions optimum for a gradation processing and a spatial frequency processing. A processor system is provided for reading the radiation image stored in the first recording medium and processing the radiation image on the basis of conditions which the processing condition determining unit determines in response to the input data associated with the radiation image.

U.S. Pat. No. 4,750,137 discloses a method and a computer program for performing the method for optimizing signals being exchanged between a host unit and an addressable-buffer peripheral device. The program optimizes an outgoing signal from the host unit by (1) creating an updated-state map representing the state of the peripheral device buffer expected to exist after processing by the peripheral device of the outgoing signal, (2) performing an exclusive-or (XOR) operation using the updated-state map and a present-state map representing the existing state of the buffer, and (3) constructing and transmitting a substitute outgoing signal which represents only changes to the buffer, and in which all premodified field flags are turned off. Position-dependent characters, such as attribute bytes, are translated into nondata characters prior to incorporation into a map, and are retranslated into their original form for use in the substitute signal.

U.S. Pat. No. 4,858,129 teaches an X-ray CT apparatus in which a plurality of dynamic tomographic images obtained by repeatedly photographing a region of interest of a subject under examination are stored in an image memory for subsequent display on a display device. A processing device extracts data of pixels along a certain the tomographic images and stores the pixel data in the image memory, in the order of photographing time of the tomographic images, thus forming a time sequence image formed of picked-up pixels. The processing device reduces a tomographic image and the time sequence image and rearranges the reduced images in one frame area of the image memory for simultaneous display thereof on the display device.

U.S. Pat. No. 5,021,770 discloses an image display system having a plurality of CRT display screens. The system is of the type in which a number of images of specific portions of a patient having a specific identification code are selected from among a multitude of X-ray

image taken by a plurality of shooting methods, and when the regions or interest are specific, a plurality of appropriate images are further selected using the previously stored aptitude values for the regions and shooting methods and displayed on the plurality of CRT display screens. In order that the segments to be inspected can be pointed to on the screen on which the image of the patient is displayed, a memory is provided which is adapted to previously store codes corresponding to the specific image of the patient and to specify the respective regions of the image in such a manner that they correspond to the pixel positions of the image.

U.S. Pat. No. 4,879,665 teaches a medical picture filing system composed of a picture data memory device, a picture data input-output device for inputting-/outputting the picture data, a retrieving device for storing the picture data into the memory device and extracting it therefrom on the basis of retrieving data, a retrieving data input device for inputting the retrieving data into the retrieving device, a retrieving data storing device for storing the retrieving data, the retrieving data being classified by block of information obtained in one-time examination. When medical pictures are filed, retrieving data collected for each examination is utilized for reducing the amount of retrieving data, while when reproduced, retrieval is carried out for each one-time examination thereby shortening the time required for retrieval.

In light of recent advances in computer data basing, digitization and compression of image data, image enhancement algorithms and cost effective computer technology, the means for improved storage and retrieval of vital patient image data is now possible.

Such system should include the following major features:

1) means to more compactly store and more efficiently retrieve image data and automatically identify the data by patient name, image type, date and the like;

2) means for physicians to quickly and remotely access particular patient image data at the medical facility even if achieved at several different locations;

3) means to prevent loss of vital image data due to ordinary human handling and misplacement errors;

4) means to quickly and affordably access image data from the physician's office;

5) means to enhance the medical images by both contrast enhancement and zooming for improved diagnostics and/or surgical guidance; and

6) means to quickly and conveniently access image data and display on a large screen in the operating room with any desired enhancement or expansion.

As described more fully hereinafter, the present invention provides means to accomplish these goals. The system uses both general purpose system elements well known to those practiced in electronic arts and specific elements having significant novelty.

SUMMARY OF THE INVENTION

The present invention relates to an automated high definition/resolution image storage, retrieval and transmission system capable for storing, transmitting and displaying medical diagnostic quality images for use with medical X-ray films or the like.

The system comprises means to process the image data from patient imaging to physician usage. The major or significant processing stages are described hereinafter. Specifically, the major steps in the image data flow include:

PATIENT RADIOGRAPHY: The patient's body is imaged and a film is exposed as in an X ray room, MRI or CAT scan lab.

FILM PREPARATION: The film(s) is developed to create a visible image with OCR readable patient identification information superimposed thereon.

FILM INTERPRETATION: Commonly, a radiologist drafts an opinion letter for the film(s). This document preferably includes an optical character reader, or OCR, readable patient identification label or standard marking area.

IMAGE SCANNING AND DIGITIZING SUBSYSTEM: A scanner subsystem digitizes each patient image film and/or document on a high resolution scanner. This digitized data is transmitted by a local high speed data link to a separate or remote master storage unit. Patient identification information is read from a standard format on each file by OCR techniques and efficiently stored with the digitized image data. Enhanced scanner resolution and gray scale requirements are provided. Further, to reduce data rate requirements, data compaction or compression is accomplished within the scanner subsystem.

To back-up possible data link down time or scanner down time, the scanner subsystem may include a CD-ROM data storage drive so that image data may continue to be digitized. The CD-ROM disk may then be manually delivered to the file room unit for subsequent use.

In an optional embodiment, the digitized data of one or two images may be written to a compact semi-conductor memory card "RAM Cards". This form of data storage may be used to send selected images for special purposes such as when the image data is needed in another city for second opinion purposes.

At this point in the image data flow, there is a split in which the original film data is stored as a "master" in a file room and the image disk is made available for active "on-line" use in an image storage and retrieval subsystem.

FILM FILING: The patient image films may be placed in the industry standard 14 by 17 inch brown paper folders and placed on conventional filing shelves. However, it is preferred that older films be tagged and stored off-site to reduce the current excessive bulk of films in many hospital file rooms. The system would now make this practical since the original films would seldom need to be accessed.

In the preferred embodiment of the system, the patient may have his entire image data collected and written to one or more of the storage CD-ROM disks for archiving at the hospital.

IMAGE STORAGE AND RETRIEVAL SUBSYSTEM: This subsystem is a remotely controllable, automatically accessable image data subsystem to store and automatically retrieve, on-demand, the compressed digital information contained on the CD-ROM disks.

The image storage and retrieval subsystem has a high-speed data link connection to the scanning and digitizing subsystem and has a write drive recording mechanism which is dedicated to receiving the data from the scanning and digitizing subsystem. This CD-ROM write drive can operate without interrupting remote access operations.

Remote access may be made to the image storage and retrieval subsystem by a variety of telecommunication links. Access will be granted only if a valid user code has been presented. By means of several read-only CD

disk drives and electronic buffering, virtually simultaneous access can be granted to several or more users.

As explained more fully hereinafter, the medical image disk will contain relatively huge quantities of data making it impractical to send over conventional data communication links without very efficient data compression technology. While there are a variety of data compression techniques available, none are well tailored to this application. Thus, novel compression means are in the a remote telecommunication access subsystem.

TELECOMMUNICATION SUBSYSTEM: Occasionally circumstances may warrant manually making an extra copy of the patient's image files to be physically delivered to an authorized requester. However, for the system to fulfill broad service to the health care industry it must be able to efficiently telecommunicate image files to remote locations both cost effectively and within a reasonable time interval.

A novel medical facsimile technology is in the preferred embodiment which works interactively with a remote requester to send only what is needed at acceptable resolutions, and the presented image is progressively updated as the communications connection is maintained until the resolution limits of the user display are reached, after which time, other images are either sent or further enhanced.

The specific technical means for accomplishing this uses the following novel technologies: a) guided image selection and transmission (GIST); b) progressive image enhancement (PIE); c) display compatible resolution (DCR); d) hexagonal pattern classification compression (HexPac) and e) run length coding (RLC), RLC is well known to those skilled in the arts.

It appears practical to send immediately useful patient data in less than one minute over a phone line (9600 baud) whereas it take many hours by conventional coding and transmission means. When wide-band telecommunications as satellite, fiber optic, micro-wave links becomes more generally available at affordable prices, then the more complex data compression techniques described here will be less important, but until that time, these types of techniques are believed essential to overall system success.

This combination of technologies to efficiently compress the image data and transmit remotely comprises the telecommunication access subsystem. In practice, these technologies may be implemented for the most part with available computer modules although several special signal processor boards are needed.

REMOTE DISPLAY TERMINAL: The quality of the image available to the user is limited or determined by the receiving presentation terminal or monitor. Two specific presentation terminal types are envisioned in the preferred embodiment of the system, a modified personal computer terminal for use in a physician's office, hospital nurses' station and the like, and a large screen presentation terminal with remote controlled interaction primarily for operating room use.

Both terminals have means to show the available patient directory of images, and means to select an image, enhancement and zoom on selected areas. Image enhancement has heretofore been impractical for film based images and thus much subtle but important pathological information has been largely lost. This is especially true of X-ray data The ability to enhance subtle contrasted tissues areas is considered to be an important feature and benefit of the system.

An optional high-resolution printer (300 dpi or better) permits the physician to print out selected images. This will be especially valuable when the physician expands and enhances selected critical image areas since a cost effective printer would otherwise not have adequate gray scale or pixel resolution to give diagnostically useful output.

Each terminal consists of a standard high performance personal computer with one or more data source interfaces such as RAM card, CD-ROM disk or data modem, a decompression graphics interface circuit and graphics display. The large screen presentation terminal has a large screen display for easy viewing for a surgeon who may be ten or more feet distant. The large screen presentation terminal also has an optional remote control so that an attending technician or nurse can scroll images, enhance and zoom, at the surgeon's request.

A keynote of each terminal design is a very simple user interface based upon a limited selection menu and obviously pointed-to graphical icons.

The invention accordingly comprises the features of construction, combination of elements, and arrangement of parts which will be exemplified in the construction hereinafter set forth, and the scope of the invention will be indicated in the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

For a fuller understanding of the nature and object of the invention, reference should be had to the following detailed description taken in connection with the accompanying drawings in which:

FIG. **1** is a functional block diagram of the entire system of the present invention.

FIG. **2** is a functional block diagram of the image scanning and digitizing means.

FIG. **3** is a functional block diagram of the image data storage and retrieval means.

FIG. **4** is a perspective view of the image data storage and retrieval means.

FIG. **5** is a functional block diagram of the telecommunication means.

FIG. **6** is a functional block diagram of the remote display terminal means.

FIG. **7** depicts the hexagonal pattern of the hexagonal compression method.

FIG. **8** depicts an actual hexagonal pattern from an X-ray film. FIG. **9** depicts the selected predetermined hexagonal pattern most closely corresponding to the actual hexagonal pattern shown in FIG. **8**.

FIG. **10** graphically represents the predetermined rotational orientations for the predetermined hexagonal patterns.

FIG. **11** graphically depicts a selected gray level slope of the selected predetermined hexagonal pattern of FIG. **9**.

FIG. **12** depicts a single pixel from the predetermined hexagonal pattern.

FIG. **13** depicts a hexagonal pattern reconstructed by a remote display terminal means corresponding to the actual hexagonal pattern shown in FIG. **8**.

FIGS. **14**-A through **14**-H depict the predetermined set of orthogonal gray level patterns.

Similar reference characters refer to similar parts throughout the several views of the drawings.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

As shown in FIG. **1**, the present invention relates to an automated high definition/resolution image storage, retrieval and transmission system generally indicated as **10** for use with medical X-ray film **12** or other documents to provide simultaneous automated access to a common data base by a plurality of remote subscribers upon request from the remote subscribers.

The automated high definition/resolution image storage, retrieval and transmission system **10** comprises an image scanning and digitizing means **14** to transform the visual image from the medical X-ray film **12** or other documents into digital data, an image data storage and retrieval means **16** to store and selectively transfer digital data upon request, a telecommunication means **18** to selectively receive digital data from the image data storage and retrieval means **16** for transmission to one of a plurality of remote visual display terminals each indicated as **20** upon request from the respective remote visual display terminal **20** through a corresponding communications network **21** such as a telephone line, satellite link, cable network or local area network such as Ethernet or an ISDN service for conversion to a visual image for display at the remote requesting site.

To improve automation and tracking, a machine readable indicia or label **22** containing key patient information may be used in association with the medical X-ray film **12**. As shown, the machine readable indicia or label **22** is affixed to the medical X-ray film **12** prior to scanning by the image scanning and digitizing means **14** to provide file access and identification. Furthermore, digital data from alternate digitized image sources collectively indicated as **24** and file identification may be fed to the image data storage and retrieval means **16** for storage and retrieval.

FIG. **2** is a functional block diagram of the image scanning and digitizing means **14** capable of converting the visual image from the medical X-ray film **12** to digitized image data for transmission to the image data storage and retrieval means **16** over a bi-directional high speed data link **25**. Specifically, the image scanning and digitizing means **14** comprises a film loading and scanning section and a data compression and transmission section generally indicated as **26** and **27** respectively and a display and control section generally indicated as **28**. The film loading and scanning section **26** comprises a film input loader **30**, alignment and sizing chamber **32**, optical character reader **34** and film scanner/digitizer **36** capable of at least 500 dots per inch resolution **36**; while, the data compression and transmission section **27** comprises a data buffer memory **38**, low-loss data compression means **40**, local data modem **42** and transmission connector **44** to operatively couple the image scanning and digitizing means **14** to the image data storage and retrieval means **16**. The low-loss data compressor **40** is also operatively coupled to a compact disk data storage drive **46** capable of writing or storing compressed digitized patient image data on a compact disk **48**. The display and control section **28** comprises a keyboard/control console **50**, display terminal **52** and control computer **54** which is operatively coupled to the other components of the image scanning and digitizing means **14** through a plurality of conductors each indicated as **55**. A film collector tray **56** may be disposed adjacent the film scanner/digitizer **36** to receive the medical X-ray film **12** therefrom following processing.

To reduce the approximately 238 Megapixels required to digitize a 14 inch by 17 inch, medical X-ray film **12** with 700 dots or pixels per inch with a two byte level to a manageable size without significant information loss, a linear gray level prediction, modified run-length code generating logic circuitry is embodied within the low-loss data compression means **40** to dynamically compress the digitized data before storage or recording. The image data is compressed with acceptable diagnostic resolution loss. The low-loss data compression means **40** measures the "local" slope of the pixel gray level and continues to compare that estimated gray level for up to an entire scan line until a pixel region is reached which differs from the linear estimate by more than a predetermined amount. The data actually sent for that region consists of the slope of the line, actual level at the origin of the slope line and the number of pixels comprising that region. The circuitry will discard linear gray level slope differences of the original film which can be reliably determined to be noise or image "artifacts". A sudden pixel (if at 1000 dots per inch) dramatic change in gray level could be rejected as dust or film noise for example. The compressed data is a trade-off between complexity, speed and minimum data loss to reduce the total data quantity stored by a factor of approximately three. Thus, about 80 Megapixels of data may still have to be stored per 14 inch by 17 inch film image.

In the preferred embodiment, the bi-directional high speed communications link **25** transmits the low-loss compressed digitized data from the developing lab room to the hospital file room where the image data storage and retrieval means **16** will transfer and store the patient and image data in a new patient file on a compact disk **48**.

Two way communications between the image scanning and digitizing means **14** and the image data storage and retrieval means **16** minimizes data loss by insuring that a compact disk **48** be available to receive and store data. Moreover, the compact disk data storage drive **46** with re-writable ROM technology can record data even if communications with the image data storage and retrieval means **16** is disrupted. Thus the image scanning and digitizing means **14** can automatically start writing data to the compact disk data storage drive **46** as soon as a image data storage and retrieval means **16** fault is sense. The display and control section **28** informs the operator of the system status.

In operation, the film lab technician may stack one or more medical X-ray films **12** onto the input loader **30** as shown in FIG. **2**. A "read" button is depressed on the keyboard/control console **50** and each film **12** is thereafter fed in automatically, digitized and transmitted to the image data storage and retrieval means **16** located in the file room. As the reading of each film **12** is completed, the film **12** is deposited into the film collector tray **56**. System status, number-of-films read logging and so forth are shown on the display terminal **52**.

Initially, the image scanning and digitizing means **14** positions the film **12** in the alignment and sizing chamber **32** on a precision carrying platen for subsequent optical scanning. This platen contains optical sensors to sense the exact film size so only the useful image area is digitized. Once the film **12** is secured onto the movable platen, the film **12** is passed through the optical character reader **34** and then to the film scanner/digitizer **36**.

The patient data and image identification is first recorded onto the remote CD-ROM file directory in the image data storage and retrieval means **16** from the

OCR "pass" and then the compressed scanned image data is sequentially written to a compact disk **48** by a CD write drive for storage with the CD library storage of the image data storage and retrieval means **16** as described more fully hereinafter as the film **12** slowly passes through the film scanner/digitizer **36**.

Specifically, the film scanner/digitizer **36** converts the image to a digital representation of preferably at least a 700 dot per inch resolution. This digital data is temporarily stored in the data buffer memory **38** where the patient data from the optical character reader **34** and corresponding digitized image data from the file scanner/digitizer **36** are properly formatted for subsequent compression and transmission to the image data storage and retrieval means **16**. The stored data is then accessed by and compressed by the data compression means **40** as previously described and transmitted through the local data modem **42** and transmission connector **44** to the image data storage and retrieval means **16** or a compact disk data storage drive **46**. The display and control section **28** permits the X-ray lab staff to monitor system status, report quantity of documents and films processed and allow for scheduling local recording of image data on compact disks **48**.

FIGS. **3** and **4** show the image data storage and retrieval means **16** to receive and store the low-loss compressed digitized patient information and image data from the image scanning and digitizing means **14** and to selectively transmit the stored low-loss compressed digitized patient information and image data to one or more of the remote visual display terminal(s) **20** through corresponding telecommunication means **18** and corresponding communications network(s) **21** upon request from one or more of the remote display terminal(s) **20**.

The image data and retrieval means **16** is essentially a central data storage library for medical subscribers to remotely access and visually display patient data and information.

As described hereinafter, the image data storage and retrieval means **16** is robotically automated to minimize hospital staff requirements. At any given time, it is estimated that a typical hospital may have several hundred active patients with requirements for physician access to corresponding image files. An active patient may require one to three compact disks **48**. Thus, the image data storage and retrieval means **16** should have sufficient means to store and retrieve at least 500 compact disks **48**.

Further, to minimize personnel requirements, the image data storage and retrieval means **16** has a semi-automatic log-in mechanism for updating the compact disk inventory and an automatic mechanism for retrieving and reading the compact disks **48** remotely via communication link interfaces similar to juke box playback mechanisms. Except for the occasional loading of new empty compact disks **48** and removal of inactive compact disks **48**, the operation of the image data storage and retrieval means **16** is fully automatic permitting authorized access at any time.

As described more fully hereinafter, several playback drives with electronic buffering are incorporated so that essentially simultaneous access can be provided to several remote requesting users. An optional duplicating CD write drive and RAM-Card drive permits additional copies to be made locally upon demand for either back-up or other use. The image data storage and retrieval means **16** has an operator's console/desk arrangement for file maintenance and duplicating control by the hospital file room clerk. Control software is a simple menu selection design so that relatively unskilled personnel can maintain the central data storage library or image data bank.

As shown in the functional block diagram of FIG. **3**, the image data storage and retrieval means **16** comprises a local data modem **58** operatively coupled between the image scanning and digitizing means **14** through the transmission connectors **44** and bi-directional high speed communication link **25**, and a selector or multiplexer **60**. A format convertor **62** is operatively coupled between the alternate digitized image source(s) **24** such as CAT **64**, MRI **66** and/or video **68** and control computer **70** which is, in turn, coupled to a control console **72** including a visual display and input means such as a keyboard. The local data mode **58** is also coupled to the hard disk (H/D) of the control computer **70** through a conductor **71**. The other components of the image data storage and retrieval means **16** are coupled to the control computer **70** through a plurality of conductors each indicated as **73**. A CD write drive **74** is operatively coupled between the multiplexer or selector **60** and an auto disk storage/retrieve mechanism **76** which is, in turn, operatively coupled to a CD library storage **78**, a manual load/purge box **80** and a plurality of data retrieval and transmission channels each indicated as **81**. Each data retrieval and transmission channel **81** comprises a CD reader drive **82** operatively coupled through a corresponding data interface **84** to a corresponding transmission connector **86**. In addition, one of the CD reader drives **82** is operatively coupled through a selector switch **88** to an optional CD write/RAM card drive **90** configured to manually receive a compact disk **48** or RAM card.

As shown in FIG. **4**, the CD library storage **78** comprises at least one cabinet **200** to operatively house **800** compact disks **48** arranged or four shelves each indicated as **202** and the auto disk storage/retrieval mechanism **76** which comprises a CD coupler **204** to engage and grasp a selected compact disk **48** and move horizontally on a support member **206** that moves vertically on a pair of end support members each indicated as **208**. An access door **210** permits movement of compact disks **48** to and from the cabinet **200**. However, in normal operation, "old" patient data is removed by writing collected image data to a single compact disk **48** through the CD write/RAM card drive **90** thus freeing internally disposed compact disks **48** for new data. The CD write/-RAM card drive **90** may also be used to collect a patient's image data on a single compact disk **48** for use in the operating room's display terminal. This obviates the need for a high speed internal hospital local area network.

The computer associated with the CD robotic arm and drive mechanism performs ordinary library maintenance functions such as retrieval of outdated files, access statistics, entry of access validation codes, and so forth. This computer subsystem also handles data communication interface functions.

Internal to the environmentally controlled cabinet **200** are a plurality of playback mechanisms (field expandable to six) which are automatically controlled by the accessing physicians via the coupled communications system. Yet another CD-ROM write drive can record new data from the image scanning and digitizing means **14** or perform library functions such as consoli-

dation of a patient's data from several compact disks **48** to a single patient-dedicated compact disk **48**.

The internal computer maintains a file log of which compact disks **48** are empty and where each patient's image data is stored by disk number and track on a disk location. When the image scanning and digitizing means **14** requests to down-load data, the auto disk storage/retrieval mechanism **76**, of the image data storage and retrieval means **16** retrieves the "current" compact disk **48** which is being written with data (if not already loaded), then loads the compact disk **48** into the CD write drive **74**, and signals to the image scanning and digitizing means **14** to transmit. Image data is then recorded with a typical record time of 4 minutes for a full-size, high density image.

Once the robotic arm has delivered the compact disk **48** to the CD write drive **74**, the robotic arm is free to access and place other compact disks **48** onto CD reader drive **82** as commanded by its communications interface. The robotic arm can find and place a disk **48** into the appropriate CD reader **82** in approximately 10 seconds. Thus, there is minimal waiting time for disk access unless all CD readers drives **82** are in use.

As shown in FIG. **3**, data is received through the input transmission connector **44** to the CD write drive **74** through the selector switch **60**. Alternately, other image data from other sources such as CAT scanners **64** or MRI medical equipment **66** may be fed through the format convertor **62** for storage on a compact disk **48**. If the other image sources are written to CD write drive **74**, file identification data must be supplied to the format convertor **62** from the control computer **70**.

The image file data received from the image scanning and digitizing means **14** is directly written to free space on a compact disk **48** in the CD write drive **74**. No other data compression or special formatting is required as the image scanning and digitizing means **14** has performed these functions. As new image data is received from the image scanning and digitizing means **14** or another image source **24**, the image data is sequentially appended to the last file on the compact disk **48** currently being written to. Thus, no attempt is made to organize a single patient's image files onto a single compact disk **48**. However, each file received is logged into the control computer **70** through the conductor **71**. Therefore, the control computer **70** always knows what disk location in the CD library storage **78** contains any specified file. Once a compact disk **48** is filled with image data, the auto disk storage/retrieve mechanism **76** removes the compact disk **48** from the CD write drive **74** and stores the compact disk **48** in an empty location in the CD library storage **78**.

The plurality of data retrieval and transmission storage **78**. channels **81** service the data requests from subscribers. As previously indicated, a single data retrieval and transmission channel **81** includes the select switch **88** to direct image file data to the optional CD write/-RAM card drive **90**. By this means, all image data for an individual patient may be collected on one or more selected compact disks **48** for archiving or other use. However, normally, the control computer **70** will automatically remove old image data by removing the compact disk **48** from the CD library storage **78** and placing the compact disk **48** in the manual load/purge box **80**. The removal age and exceptions information are selected by the system operator from the control console **72**.

The control console **72** is also used to enter and maintain subscriber access identification codes in an "authorization file". This updated user authorization file data is sent through a transmission connector **92** to the telecommunications means **18** internal computer memory accessed by the control computer **70** as needed to accept or reject subscriber data link access requests. The user authorization file normally residing in the telecommunications means **18** may be remotely updated by authorized persons.

The number of data retrieval and transmission channels **81** depends on intended subscriber demand. The image data storage and retrieval means **16** is modular and may be upgraded as demand increases. Each data interface **84** operates cooperatively with the telecommunications means **18** to send only as much information as the telecommunications means **18** can compress and transmit to a remote visual display terminal **20** of a requesting subscriber in a given time interval. Thus, the interface is an asynchronous block-buffered type.

Since the entire system **10** is designed to provide easy and quick access to a patient's medical images, it is vital that these images be transmitted to a variety of locations in a timely and cost effective manner and further data compression is imperative. The telephone network is still the most commonly available network but has a severe data rate limitation of about 1200 bytes per second (9600 baud). While other high speed telecommunication channels such as time-shared cable, satellite link may eventually become commonly available, for the immediately forseeable future, the "phone" network must be used if system **10** is to be practical today.

As noted earlier, a typical medical image may be stored as 119 megabytes of data. At 1200 bytes per second, it could take 27 hours to completely transmit the already compressed medical image data. This is obviously unacceptable. To overcome this obstacle, the telecommunications means **18** as shown in FIG. **5** utilizes five distinct data handling technologies to achieve useful data image transmission in less than one minute:

(1) Guided Image Selection and Transmission or GIST depends upon interactive use by the physician to identify what portions of an image are needed for enhancement or better resolution. Thus the data actually transmitted to the subscriber's visual display terminal **20** is guided by the subscriber observing the image. In particular, once the user has an image displayed on his or her visual display terminal **20**, the user may outline a specific region of interest such as a lesion or tumerous growth for more detailed study. The operator may select this region using a "mouse" or light pen or similar well-known computer display terminal peripheral device Having selected this region, the visual display terminal **20** will display the more detailed pixel data be sent on this region. The telecommunications means **18** will continue to send further precision data until the natural resolution limits of the display are reached or all available data is sent and received. This process of expanding an image region is known as "zooming" in computer-aided design systems. The novel feature here is that the image is further refined in resolution when "zoomed". The means for doing this and knowing when to "stop" further pixel transmission is defined by the PIE and DCR technology described hereinafter,

(2) Progressive Image Enhancement or PIE utilizes the transmission time from the instant a first "crude" image is presented to the subscriber to the present time of observation to progressively enhance the quality of

the presented image. The longer the user observes a selected image, the "better" the image becomes in the sense of pixel resolution and quantity of gray levels. In the preferred embodiment, hexagonal pixel groups are first transmitted using the HexPac pattern compression technology described hereinafter. Once a full terminal screen display has been made composed of these hexagonal patterns, then the telecommunications means **18** transmits more precise pixel detail First all pixels located on the periphery of each hexagonal group are updated with their exact gray level values and thereafter, all inner pixels are similarly updated. If the display terminal's resolution is less than the 1000 dots per inch of the source image data, then pixel groups are sent, such as a square of four pixels, which match the display resolution and "zoom" expansion selected. This display matching technique is further defined hereinafter as DCR,

(3) Display Compatible Resolution or DCR transmits information about the user's terminal **20** back to the telecommunications means **18**. Only data with a resolution compatible with that terminal **20** will be sent. Any excess data-link connect time can be used to send other image data which is likely to be requested or has been pre-specified to be sent.

(4) An image pattern compression method comprising a Hexagonal Pattern Classification or HexPac exploits the two dimensional nature of images. The data received by telecommunications means **18** is first uncompressed and placed into a multi-scanline digital buffer. This image data is then divided up into hexagonal cells and matched against predefined patterns. Many fewer bits of data can be used to represent these predefined patterns thus substantially compressing the image data for phone-line transmission. The pixels of these hexagonal patterns may easily be "refined" by the PIE technology described earlier. If the DCR subsystem determines that the user terminal has a pixel area of, say, 1500 by 1000 dots, then the HexPac technology recreates a new super pixel which is the average gray level of all actual pixels within that super pixel area. This immediately reduces the quantity of pixels to be sent (to only 1500 by 1000 pixels). Without further data compression, this quantity of data would still require about 26 minutes of data transmission time at 9600 baud, the highest available phone network data rate.

(5) Run length coding or RLC permits data to be compressed by specifying how many pixels have the same gray level in a sequence or "run length" of scanning. The image data sent by a CD reader drive **82** to telecommunications means **18** is compressed with run-length coding but is nearly loss-less in the duplication of the original film data. To substantially reduce the quantity of data needed to send an acceptable medical image to a remote user terminal **20** over the data-rate limited phone-line modem, a "lossy" compression is used. Since the PIE and DCR techniques described earlier will eventually provide any degree of diagnostic image integrity desired, it is believed acceptable to initially transmit a "lossy" image provided it gives adequate resolution for the user to begin the analysis and guided image selection. Many fewer bits can describe this "run" of similar gray levels thus compressing the amount of data sent. This technique is well known and often used in facsimile transmission A one dimensional RLC is incorporated in the preferred embodiment but since HexPac elements are being coded, it can be considered more accurately a quasi two dimensional RLC compression.

FIG. **5** is a functional block diagram of the telecommunications means **18** including a control computer **94** operatively coupled to the image data storage and retrieval means **16** through a transmission connector **96**. The various components of the telecommunications means **18** including a status panel **98** with a plurality of system indicators each indicated as **99** and a plurality of data compression channels each generally indicated as **100** coupled to the control computer **94** by a plurality of conductors each indicated as **101**.

Each data compression channel **100** comprises a transmission connector **86**, a communications data interface **102**, a first compression processor or means **104** including logic means to generate the GIST and DCR data compressions and corresponding first data memory **106**, a second compression processor or means **108** including logic means to generate the PIE and HEXPAC data compressions and corresponding second data memory **110** and a third compression processor or means **12** including logic means to generate the RLC data compression and corresponding third data memory **114**, a corresponding modem **116** and a transmission connector **118**.

The control computer **94** coordinates or controls data flow to and from the plurality of data compression channels **100** through the transmission connectors **86** and **118** respectively. Validated subscriber image data requests are transmitted to the image data storage and retrieval means **16** which searches the image library file **78** for availability of the requested compact disk **48**. If available, the image data storage and retrieval means **16** loads the appropriate disk **48** from CD library storage **78** into a CD reader drive **82** and informs telecommunications means from **18** through the transmission connector **96** to the control computer **94** that a specific data interface **84** has data available to be transmitted through the corresponding transmission connectors **86**. Once a subscriber transaction has been turned over to a specific data retrieval and transmission channel **81**, the data compression channel **100** receives the data therefrom unless commanded to stop by a feedback control line. The data interface **102** is used to inform the CD reader drive **82** as to what portion of the image is requested by the first compression means **104**. Generally, the complete image is first requested. Thus the CD reader drive **82** is requested to read the image data from the start.

The data is temporarily stored in the first data memory **106**. Here the pixel data is first expanded from the RLC code into uncompressed pixel data. This is only done on a relatively few number of scan lines—about one tenth of an inch height of original image data. This uncompressed data is then remapped by the first compression means **104** into "larger" pixels whose average intensity is the average of all combined pixels compatible with the display resolution receiving remote visual display terminal **20**. This "super pixel" data is then fed to the second data memory **110**. The super pixel data in memory **110** is then processed by the second compression means **108**. Initially, the lowest resolution image will be transmitted to rapidly form a useful remote image on the requesting remote visual display terminal **20** through a communications network **21**. This will be done by combining super pixels in the second data memory **110** into hexagonal patterns which approximate the group of super pixels. These HexPac data packets are then sent to the third data memory **114**. There the Hex-

Pac data packets are further compressed by the third compression means **112**. These packets of run-length coded HexPac data packets are then transmitted through the corresponding modem **116** and transmission connector **118** over the selected communications network **21**. The modem **116** includes state of the art error control techniques such as block retransmission when a remote error has been detected. Thus, data transmission is essentially error free as needed for compressed data handling.

The control computer **94** includes circuitry means to monitor the activity of each data channel. The identification of each subscriber is logged along with the total connect time for billing purposes. Thus the control computer **94** generally coordinates the plurality of communication links and their connections to the particular data retrieval and transmission channel **81** within the image data storage and retrieval means **16** as well as granting access and performing connection accounting tasks. The status panel **98**, connected to the control computer **94** is used to aide in system debug and indicate operation of the data compression channels **100**. The status panel **98** would not normally be used by hospital personnel but by system service technicians.

The control computer **94** also has a permanent memory such as a hard disk to record subscriber usage data and internally sensed hardware problems. This data may be downloaded on any of the transmission connectors **118** when a correct authorization code has been received. Thus, the servicing company can acquire subscriber usage information remotely for billing purposes and system diagnostic purposes.

The preferred embodiment of the telecommunications means **18** uses modular communication channel hardware. Thus, the module may be customized to function with any type of communication channel such as satellite links, cable networks or a local area network such as Ethernet or ISDN services.

It is important to note that all communications is bidirectional so that if, say, a remote visual display terminal **20** should become temporarily "overloaded" with image data due to decompression processing delays or due to a detected data error, then, the remote visual display terminal **20** may request that data transmission be stopped or a block of data be repeated until it is received correctly.

FIG. **7** graphically shows a hexagonal group of the hexagonal compression method comprising a group of square image pixels partitioned into a hexagonal group. The pixels are numbered for convenience of reference from the inside to the outside in a clock-wise manner. Each hexagonal group or packet comprises 24 super pixels as earlier described but other numbers are possible. It is assumed that each pixel is gray level coded using 2 bytes of data. Thus, the hexagonal group requires (24×2) 48 bytes of data to fully represent the 24 super pixels comprising the image pattern at the user terminal **20**.

FIG. **8** shows a typical pattern as may occur in a region of an X-ray film **12**. The method predefines a group of likely patterns, one of which is represented as a "best" match as in FIG. **9** with the actual pattern in FIG. **8**. As shown in FIGS. **14**A through **14**H, there are 8 possible predetermined representative gray level patterns represented by 3 bits. These patterns are specifically selected to be essentially uncorrelated with each other even rotated relative to each other. As shown in FIG. **10**, these patterns may be rotated through 8 equal angles (another 3 bits of data) to best match the actual pattern. Rotation angle "1" is shown in FIG. **10** as the best match for the given example. Thus far, six bits have been used to approximate the actual pattern of FIG. **8**. As shown in FIGS. **14**A through **14**H, each fictitious pattern includes a dark and light regions and origin. Although FIG. **11** discloses a straight gray level slope corresponding to the pattern shown in FIG. **14**A, the gray level slope will vary with the fictitious pattern. For example, the gray level slope of the fictitious pattern shown in FIG. **14**D would closely approximate a V shape.

FIG. **11** shows how the gray level slope may be discretely selected to best match the slope of the actual pattern. Two bits are used to approximate this slope.

FIG. **12** shows that one particular pixel, such as the darkest pixel, has been selected to be fairly precisely gray level represented by means of 8 bits (256 gray levels).

The total bits required to approximate the actual pattern is 16 or two bytes. FIG. **13** shows how this fictitious or reconstructed pattern may be reproduced at the user terminal **20** when decoded.

In this example, only two bytes were required to represent "adequately" an original 48 bytes of image data. Thus, a 24 to 1 compression ratio has been achieved. Further, run-length encoding (RLC) may be used on these HexPac Groups to further reduce redundant spans of white and black. It is estimated that the combined compression ratio of HexPac and RLC on the super-pixel image is about 36 to 1 for this particular set of parameters. This combined compression technology reduces data transmission time (at 9600 baud) to approximately 43 seconds for an initial useful medical image.

For medical images, further enhancements through the PIE compression should favor the elimination of artificial lining between hexagonal patterns first. As the user continues to view the same image, then the PIE compression will progressively improve the gray level integrity by updating all number 24 pixels to 8 bits of gray level resolution and updating all number 23 pixels to 8 bits of gray level and so forth for all remaining pixels in descending order. This process takes about 10 minutes at 9600 baud to update all peripheral hexagonal pixels and about 20 minutes total for all pixels.

If the user continues to observe or request further image resolution, the telcommunication means **20** causes each pixel gray level to be updated by one additional bit in descending order again until the full 16 bits of gray level is received and stored at the terminal **20** for each super pixel. Each doubling of gray level resolution takes between 1 and 2.6 minutes at 9600 baud depending on the run length statistics of the gray levels.

FIG. **6** is a functional block diagram of a remote visual display terminal **20** to be operatively coupled to one of the data compression channels **100** of the telecommunication means **18** by a communications network **21** and a transmission connector **118**. The visual display terminal **20** comprises a data communications modem **120** operatively coupled to a control computer **122** and RLC decompression means **124**. The RLC decompression means **124** is, in turn, operatively coupled to a memory **126**, a PIE bypass **128** and a pattern select and modifier **130** which is operatively coupled to a Hexpac pattern ROM **132** and the control computer **122**. A memory **134** is operatively coupled between the PIE bypass **128** and pattern selector and modifier **130** and a display drive **136** which is operatively coupled to an

image display **138**. In addition, an image enhancing processor means **139** including circuitry to generate edge contrast enhancement, gray level contrast enhancement by means of gray level region expansion or differential gray level tracking and gray level enhancement or other state of the art image enhancement method well known to those skilled in the art. The control computer **122** is operatively coupled to an interface **140** to a first control or selector means **142** and a second control or selector means including a radio receiver **144** and signal command decoder **146** for use with portable keyboard transmitter **148**. In addition, an optional CD read/write drive **150** may be provided for use with a compact disk **48**.

The modem **120** has built-in compatible error correction technology to communicate with corresponding transmitting data compression channel **100**. After the user has selected the image data storage and retrieval means **16** and validated authority by swiping through an identification magnetic card **152** or otherwise through a magnetic card reader **154** entered an assigned security code, the operator may select a patient and one or more image files presented to him on the display screen **138**. Selection is accomplished by a touch-screen overlay on the first control or selector means **142** or by the keyboard transmitter **148** of the second control or selector means.

Once one or more images have been selected by the user, the modem **120** writes image data to the temporary memory **126** which is actively accessed by the RLC decompression means **124**. This decompressed data describes the HexPac patterns or packets as stripes of the image running, for example sequentially from left to right. These Hexpac pattern specifications, typically 2 data bytes or 16 bits are then routed to a pattern selection processor **130** which accesses the predefined patterns from Read-Only Memory device **132**. Each pattern is then rotated and gray-level modified by processor **130** according to the HexPac 16 bit pattern specification received from the RLC decompression means **124**. Each modified pattern is then written to a graphics display memory circuit **134**. As the graphics display memory circuit **134**, develops the pattern data, the display driver **136** and image display **138** show the image on the screen as it is received. In this manner, the entire "first pass" medical image is painted on the image display **138** screen.

If the user makes no further intervention, then once the image is fully displayed on this "first pass", then the progressive image enhancement technology requests pixel enhancement data. This enhancement data bypasses pattern selector and modifier **130** and is routed through the PIE bypass **128**. In the PIE bypass **128**, the enhanced pixel information is directed to the correct graphic memory locations in the graphics display means circuit **134'**. Thus, the display driver **136** and image display **138** are continually resolution enhanced.

If the image is fully enhanced to the limits set by the DCR in the control computer **122**, the image storage and retrieval means **16** is directed by the control computer **122** to begin sending new image data on the next selected image and begin storing this image data in a second graphics display memory circuit **134''**. This second data memory **134''** can hold one more images and may be selected immediately by the user when he is finished inspecting an earlier image. The user may further direct by touch screen command **142** that these be stored in the computer's hard disk or archived by the optional CD read/write drive **150**.

The user may at any time select a portion of the displayed image for further expansion by enabling or selecting the Guided Image Selection & Transmission (GIST) circuitry in the control computer **122** or image enhancement through the image enhancing processor means **139**. This may be accomplished either by touch screen control means **142** or the second remote control means **144/146/148**. This remote keyboard and transmitter unit **144/146/148** duplicates the on-display simulated push-buttons of the touch screen control means **142**. Coded command signals sent by **148** are received by radio receiver **144** and decoded by **146**. These commands are then accepted by control computer **122** as though they were normal keyboard commands.

The user may terminate a session with the image data storage and retrieval means **16** at any time by selection of stop and escape command. While a printer is not shown in this description, it can be an optional addition to terminal **20**.

In summary, the image data storage and retrieval means **16** selects the first image and writes that data to a temporary memory buffer in the telecommunication means **18**. Information about the subscriber's terminal is uploaded to the telecommunications means **18** so that the Display Compatible Resolution (DCR) logic circuitry knows when to stop sending added data for the requested first image. A special interactive compression computer then compresses this first image data using the HexPac circuitry and sends that data over the data link modem to the subscriber terminal **20**. Error detection and correction methods will generally be used in this communications link protocol.

Once a first "crude" image is sent to the subscriber visual display terminal **20**, then the Progress Image Enhancement (PIE) circuitry begins to send additional data to further refine the resolution of each hexagonal pixel region. If no further guidance is given by the subscriber, the PIE will continue to refine the picture's resolution until its natural limit is sent for the terminal **20**. Thereafter, the PIE will begin sending image data from the second specified film and loading it into yet another memory buffer. Thus, the data link connection is always transmitting useful data even though the subscriber may only be analyzing one image for some time.

However, should the user desire to zoom in on a particular region of an image, he or she may define that region desired on the terminal **20** by the Guided Image Selection & Transmission (GIST) to expand the visual display accordingly. The DCR will recognize the requirement for additional resolution and command the PIE to begin transmitting additional pixel information until such time as the DCR informs it that once again the natural display resolution limit has been reached.

The following image enhancement means present in the instant invention: edge contrast enhancement, gray level contrast enhancement by means of gray level region expansion or differential gray level tracking and gray level enhancement and may be accomplished by the image enhancing processor means **139** in the visual display terminals **20**.

The human eye cannot reliably discern gray level differences less than approximately 2%. Yet, significant tissue density information causes X-ray gray level differences in this range and below. The enhancement technologies above will cause these tissue density differ-

ences to be magnified thus revealing hitherto unseen image data.

To ensure ease of use, the following features are incorporated: touch-screen selection of commands, magnetic card identification of the subscriber or user, icon based menus or selection buttons on the CRT display and split image display screen overlays

It will thus be seen that the objects set forth above, among those made apparent from the preceding description are efficiently attained and since certain changes may be made in the above construction without departing from the scope of the invention, it is intended that all matter contained in the above description or shown in the accompanying drawing shall be interpreted as illustrative and not in a limiting sense.

It is also to be understood that the following claims are intended to cover all of the generic and specific features of the invention herein described, and all statements of the scope of the invention which, as a matter of language, might be said to fall therebetween.

Now that the invention has been described,

What is claimed is:

1. A medical image storage, retrieval and transmission system providing simultaneous automated access to an image data base by a plurality of remote subscribers upon request over a communications network, said system comprising

image digitizing means forming a first digitized representation of said image,

first data compression means generating from said first digitized representation a low-loss second digitized representation of said image,

image data storage and retrieval means comprising means to receive and store said second digitized representation and to selectively provide said second digitized representation to data channel compression means compressing said second digitized representation to form a third digitized representation, and

telecommunication means including means to selectively transmit said requested third digitized representation to a requesting remote visual display terminal, said telecommunication means initially telecommunicating a first portion of said third digital representation, said remote terminal including means to convert said first portion of said third representation to a visual image having an initial resolution less than a resolution limit of said requesting terminal, said telecommuncation means subsequently telecommunicating a second portion of said third digital representation, said second portion usable with said first portion to form an image having a resolution intermediate said initial resolution and said resolution limit.

2. A system of claim 1 wherein said first data compression means includes logic means to generate a run-length compressed digitized image data signal as said second representation.

3. A system of claim 1 wherein said first data compression means is operatively coupled to an external data storage drive to store said second representation.

4. A system of claim 1 wherein said image data storage and retrieval means comprises a data modem coupled to said image scanning and digitizing means, a write drive operatively coupled to said data modem to receive and to store said second digitized representation, and a plurality of data retrieval and transmission channels, each said channel comprising an image data reader means operatively coupled to said telecommunication means to selectively receive said second digitized representation of said image for transmission to a said requesting remote visual display terminal.

5. A system of claim 4, wherein a said image data reader means is operatively coupled to an external data write drive configured to receive a storage medium and to store compressed digital image data thereon.

6. A system of claim 4 wherein said telecommunication means comprises a control computer operatively coupled to said image data storage and retrieval means to selectively control data flow between said image data storage and retrieval means and said remote visual display terminal and a plurality of data compression channels coupled to said control computer, wherein each said data compression channel comprises a data memory including means to decompress said low-loss second representation of said image data received from said data retrieval and transmission channel and a compression means including logic means to compress and decompress second representation of said image data to form said third digitized representation of said image for transmission over said communication network to a said requesting remote visual display terminal.

7. A system of claim 1 wherein said first portion of said third representation of said image comprises a plurality of super pixels, and said second portion of said third representation comprises data representative of exact gray levels of a first subset of said super pixels and wherein a third portion of said third representation comprises similar data for a third subset of said super pixels.

8. A system of claim 1 wherein said remote terminal further includes means to select a region of a said image and said telecommunication means includes means to transmit a third portion of said third digitized representation, said third portion specific to said selected region, thereby providing an expanded visual display of said selected region, said expanded visual display containing more pixels than were included in said selected region.

9. A system of claim 1 wherein said remote visual display terminal further includes logic means to enhance an edge contrast of a displayed image.

10. A system of claim 1 wherein said remote visual display terminal further includes logic means to enhance gray level contrast by means of gray level region expansion.

11. A system of claim 1 wherein said remote visual display terminal further includes logic means for differential gray level tracking and gray level enhancement.

12. A system of claim 1 wherein individual patient information corresponding to said image is read by an optical character reader for compression and transmission with a said corresponding second digital representation to said image data storage and retrieval means.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,321,520
DATED : June 14, 1994
INVENTOR(S) : Jorge J. Inga, et al

It is certified that error appears in the above-indentified patent and that said Letters Patent is hereby corrected as shown below:

Column 20, lines 27 and 28, "and decompress" should read --said decompressed--.

Signed and Sealed this

Twenty-fifth Day of October, 1994

Attest:

BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

US005321520C1

(12) EX PARTE REEXAMINATION CERTIFICATE (8039th)

United States Patent

Inga et al.

(10) Number: US 5,321,520 C1

(45) Certificate Issued: Feb. 22, 2011

(54) **AUTOMATED HIGH DEFINITION/RESOLUTION IMAGE STORAGE, RETRIEVAL AND TRANSMISSION SYSTEM**

(75) Inventors: **Jorge J. Inga**, Tampa, FL (US); **Thomas V. Saliga**, Tampa, FL (US)

(73) Assignee: **Automated Medical Access Corporation**, Tampa, FL (US)

Reexamination Request:
No. 90/011,260, Sep. 30, 2010

Reexamination Certificate for:

Patent No.:	**5,321,520**
Issued:	**Jun. 14, 1994**
Appl. No.:	**07/915,298**
Filed:	**Jul. 20, 1992**

Certificate of Correction issued Oct. 25, 1994.

(51) **Int. Cl.**
H04N 1/21 (2006.01)
H04N 1/41 (2006.01)

(52) **U.S. Cl.** **358/403**; 358/1.9

(58) **Field of Classification Search** 358/403
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,520,671 A 6/1985 Hardin
4,903,317 A 2/1990 Nishihara et al.
4,941,190 A 7/1990 Joyce

OTHER PUBLICATIONS

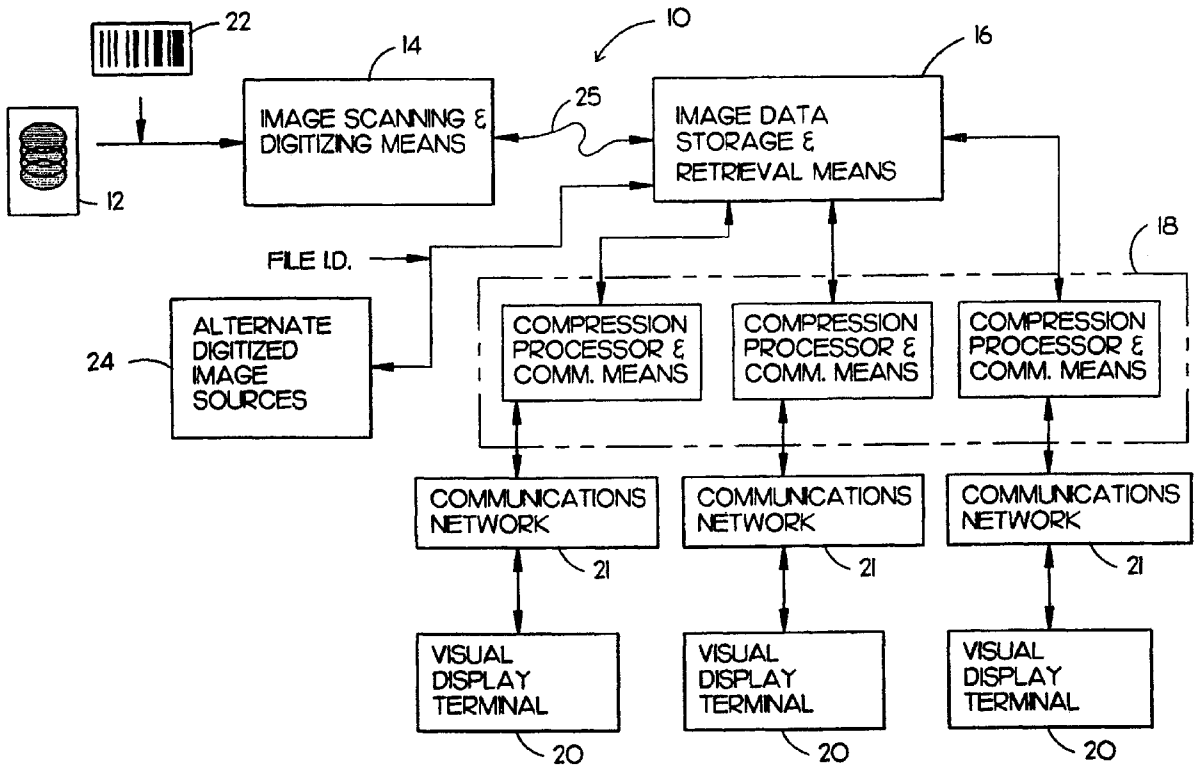
Sharaf E. Elnahas, Progressive Coding and Transmission of Digital Diagnostic Pictures, MI–5 IEEE Transactions on Med. Imaging 73 (Jun. 1986).*

* cited by examiner

Primary Examiner—Henry N Tran

(57) **ABSTRACT**

An automated high definition/resolution image storage, retrieval and transmission system for use with medical X-ray film or other documents to provide simultaneous automated access to a common data base by a plurality of remote subscribers upon request, the automated high definition/resolution image storage, retrieval and transmission system comprising an image scanning and digitizing subsystem to scan and digitize visual image information from an image film or the like; an image data storage and retrieval subsystem to receive and store the digitized information and to selectively provide the digitized information upon request from a remote site, a telecommunication subsystem to selectively transmit the requested digitized information from the image data storage and retrieval subsystem to the requesting remote visual display terminal for conversion to a visual image at the remote site to visually display the requested information from the image data storage and retrieval subsystem.



EX PARTE REEXAMINATION CERTIFICATE ISSUED UNDER 35 U.S.C. 307

NO AMENDMENTS HAVE BEEN MADE TO THE PATENT

AS A RESULT OF REEXAMINATION, IT HAS BEEN DETERMINED THAT:

The patentability of claims **1-12** is confirmed.

* * * * *

US006182114B1

(12) **United States Patent**
Yap et al.

(10) **Patent No.: US 6,182,114 B1**
(45) **Date of Patent: Jan. 30, 2001**

(54) **APPARATUS AND METHOD FOR REALTIME VISUALIZATION USING USER-DEFINED DYNAMIC, MULTI-FOVEATED IMAGES**

(75) Inventors: **Chee K. Yap**; **Ee-Chien Chang**, both of New York, NY (US); **Ting-Jen Yen**, Jersey City, NJ (US)

(73) Assignee: **New York University**, New York, NY (US)

(*) Notice: Under 35 U.S.C. 154(b), the term of this patent shall be extended for 0 days.

(21) Appl. No.: **09/005,174**

(22) Filed: **Jan. 9, 1998**

(51) **Int. Cl.**[7] **G06F 15/16**
(52) **U.S. Cl.** **709/203**; 709/246
(58) **Field of Search** 709/217, 219, 709/246, 247, 203; 707/10; 382/103, 233, 235, 232, 240, 302

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,622,632		11/1986	Tanimoto .	
5,341,466		8/1994	Perlin .	
5,481,622	*	1/1996	Gerhardt et al.	382/103
5,568,598	*	10/1996	Mack et al.	382/302 X
5,710,835	*	1/1998	Bradley	382/233
5,724,070	*	3/1998	Denninghoff et al.	382/235 X
5,861,920	*	1/1999	Mead et al.	382/232 X
5,880,856	*	3/1999	Ferriere	382/240 X
5,920,865	*	7/1999	Ariga	707/10

OTHER PUBLICATIONS

Tams Frajka et al., Progressive Image Coding with Spatially Variable Resolution, IEEE, Proceedings International Conference on Image Processing 1997, Oct. 1997, vol. 1, pp. 53–56.*

E. C. Chang et al., "Realtime Visualization of Large . . . " Mar. 31, 11997,pp. 1–9, Courant Institute of Mathematical Sciences, New York University, N.Y. U.S.A.

E. C. Chang et al., "A Wavelet Approach to Foveating Images", Jan. 10, 1997,pp. 1–11, Courant Institute of Mathematical Sciences, New York University, N.Y. U.S.A.

S.G. Mallat, "A Theory for Multiresolutional Signal Decomposition . . . ", IEEE Transactions on Pattern Analysis and Machine Intelligence,pp. 3–23, Jul. 1989, vol. 11, No. 7, IEEE Computer Society.

News Release, "Wavelet Image Features",Summus'Wavelet Image Compression,Summus 14 pages.

R.L. White et al., "Compression and Progressive Transmission of Astronomical Images", SPIE Technical Conference 2199, 1994.

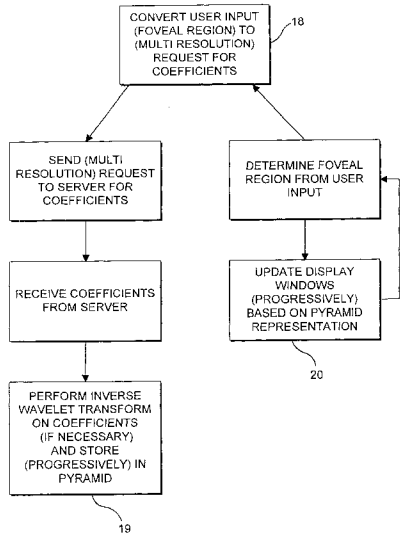
(List continued on next page.)

Primary Examiner—Zarni Maung
Assistant Examiner—Patrice Winder
(74) *Attorney, Agent, or Firm*—Baker Botts, L.L.P.

(57) **ABSTRACT**

A client apparatus which enables a realtime visualization of at least one image. The client apparatus includes a storage device which stores first data corresponding to a multifoveated representation of an original image, and a user input device which providing second data corresponding to at least one visualization command of at least one user. In addition, the client apparatus includes a processing arrangement which generates third data corresponding to a multi-foveated image using the first data, the second data and a foveation operator.

8 Claims, 6 Drawing Sheets



OTHER PUBLICATIONS

E.L. Schwartz, "The Development of Specific Visual . . . " Journal of Theoretical Biology, 69:655–685, 1977.
F.S. Hill Jr. et al.,"Interactive Image Query . . . " Computer Graphics, 17(3), 1983.
T.H. Reeves et al., "Adaptive Foveation of MPEG Video", Proceedings of the 4th ACM International Multimedia Conference, 1996.
R.S. Wallace et al., "Space–variant image processing". Int'l. J. of Computer Vision, 13:1(1994) 71–90.
E.L. Schwartz A quantitative model of the functional architecture: Biological cybernetics, 37(1980) 63–76.
P. Kortum et al., "Implementation of a Foveated Image . . . " Human Vision and Electronic Imagining, SPIE Proceedings vol. 2657, 350–360, 1996.
M.H. Gross et al., "Efficient triangular surface . . . ", IEEE Trans on Visualization and Computer Graphics, 2(2) 1996.

* cited by examiner

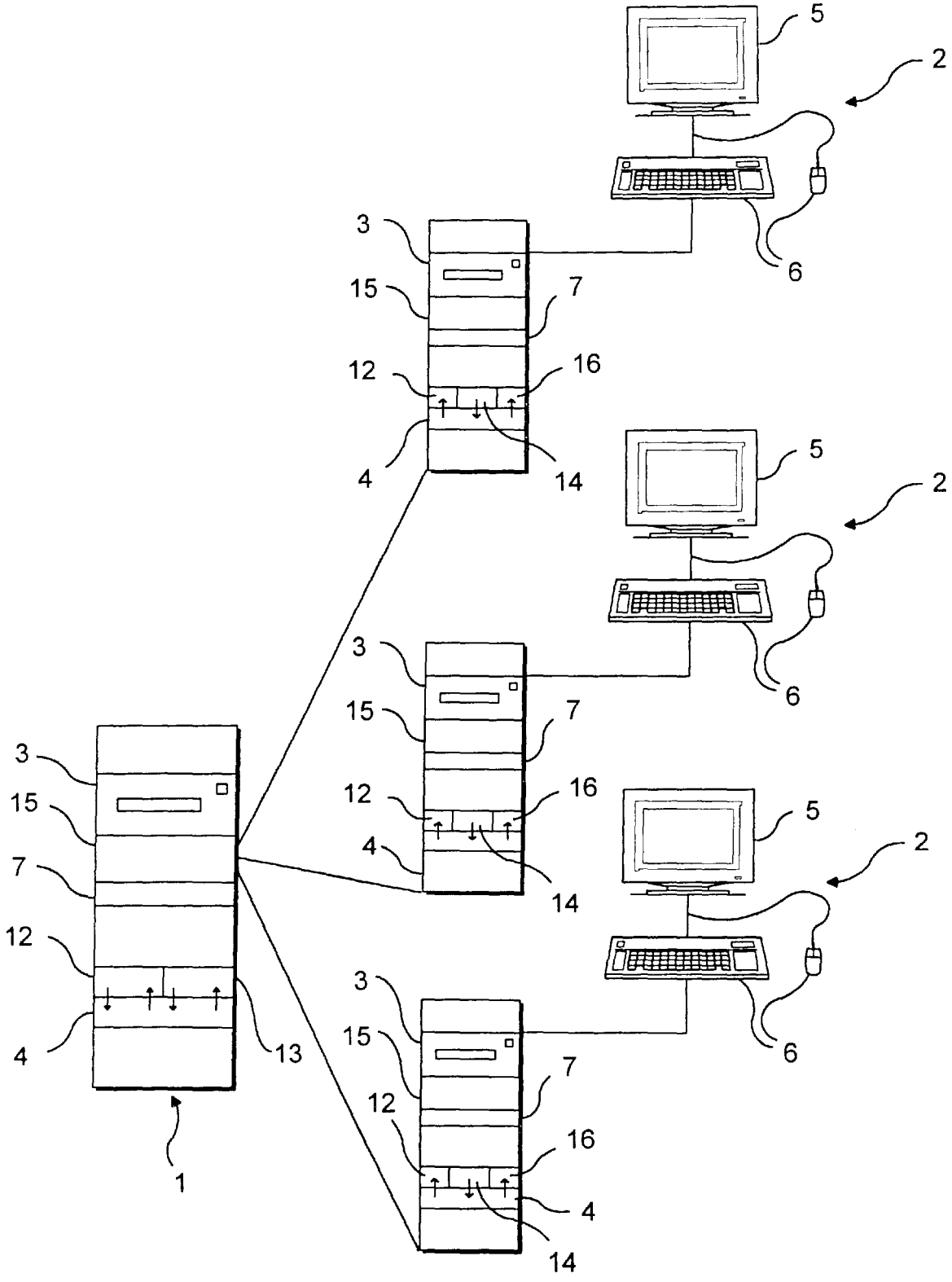


FIG. 1

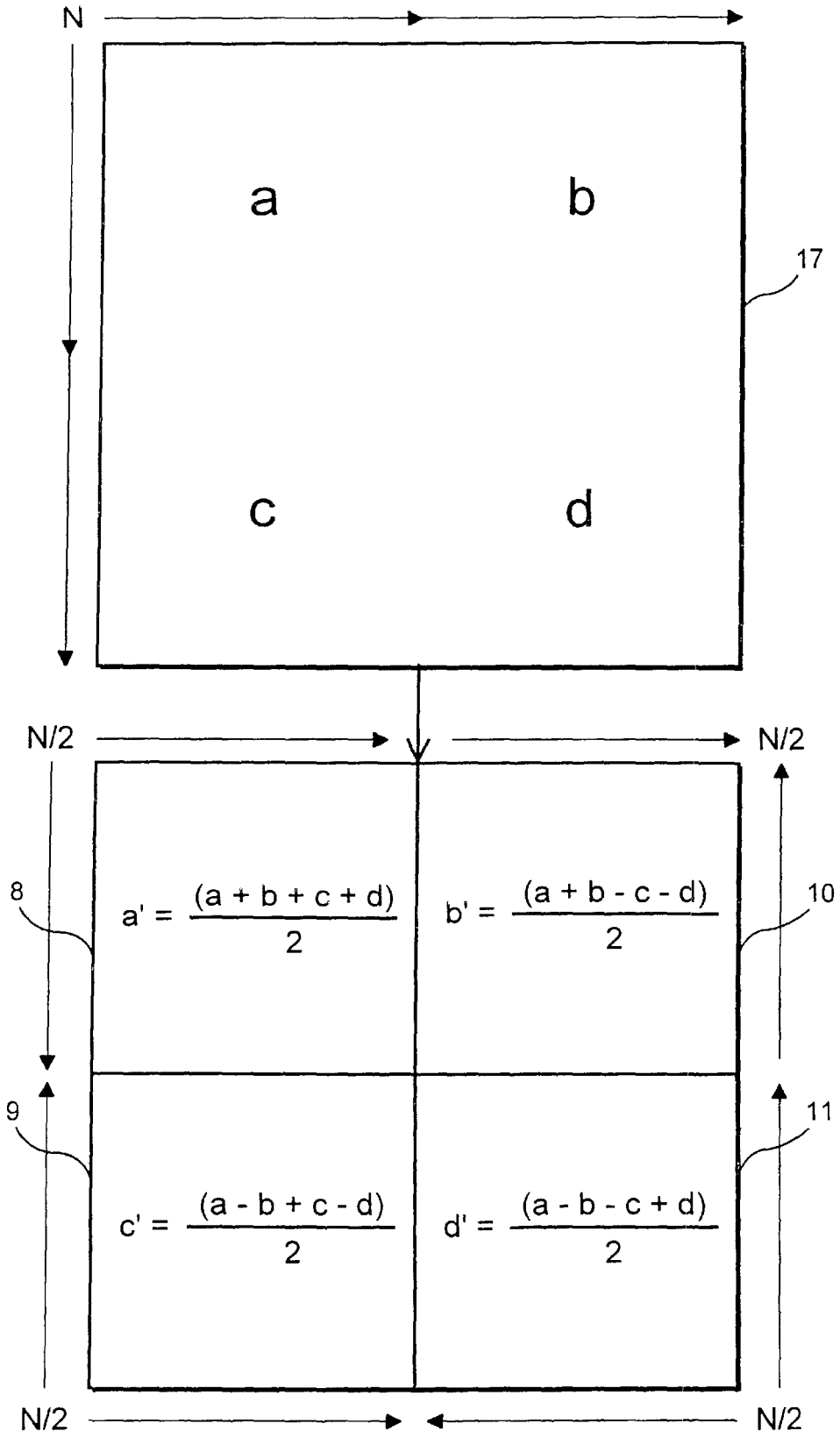


FIG. 2A

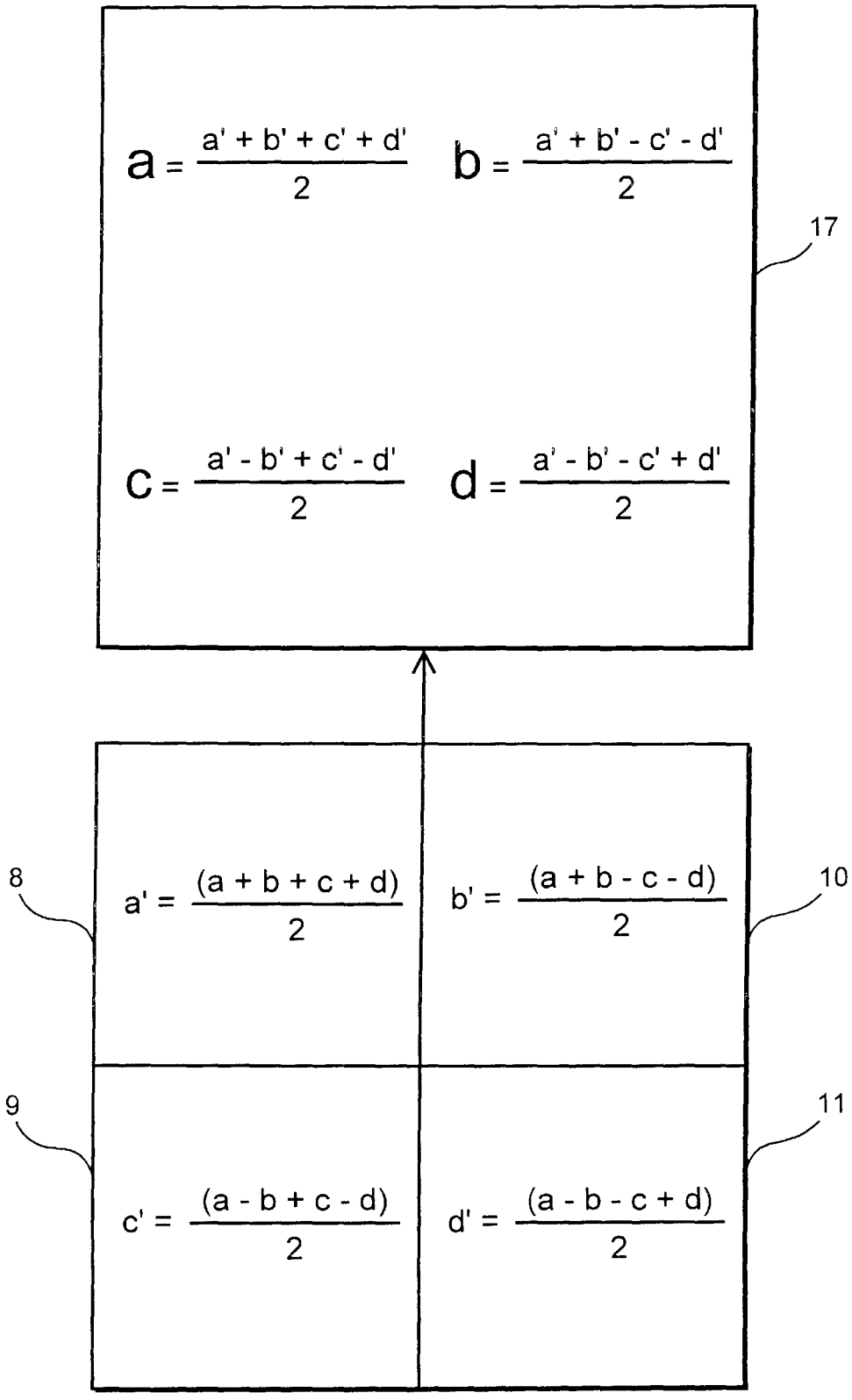

$a = \frac{a' + b' + c' + d'}{2}$
$b = \frac{a' + b' - c' - d'}{2}$
17
$c = \frac{a' - b' + c' - d'}{2}$
$d = \frac{a' - b' - c' + d'}{2}$
8
$a' = \frac{(a + b + c + d)}{2}$
$b' = \frac{(a + b - c - d)}{2}$
10
9
$c' = \frac{(a - b + c - d)}{2}$
$d' = \frac{(a - b - c + d)}{2}$
11

FIG. 2B

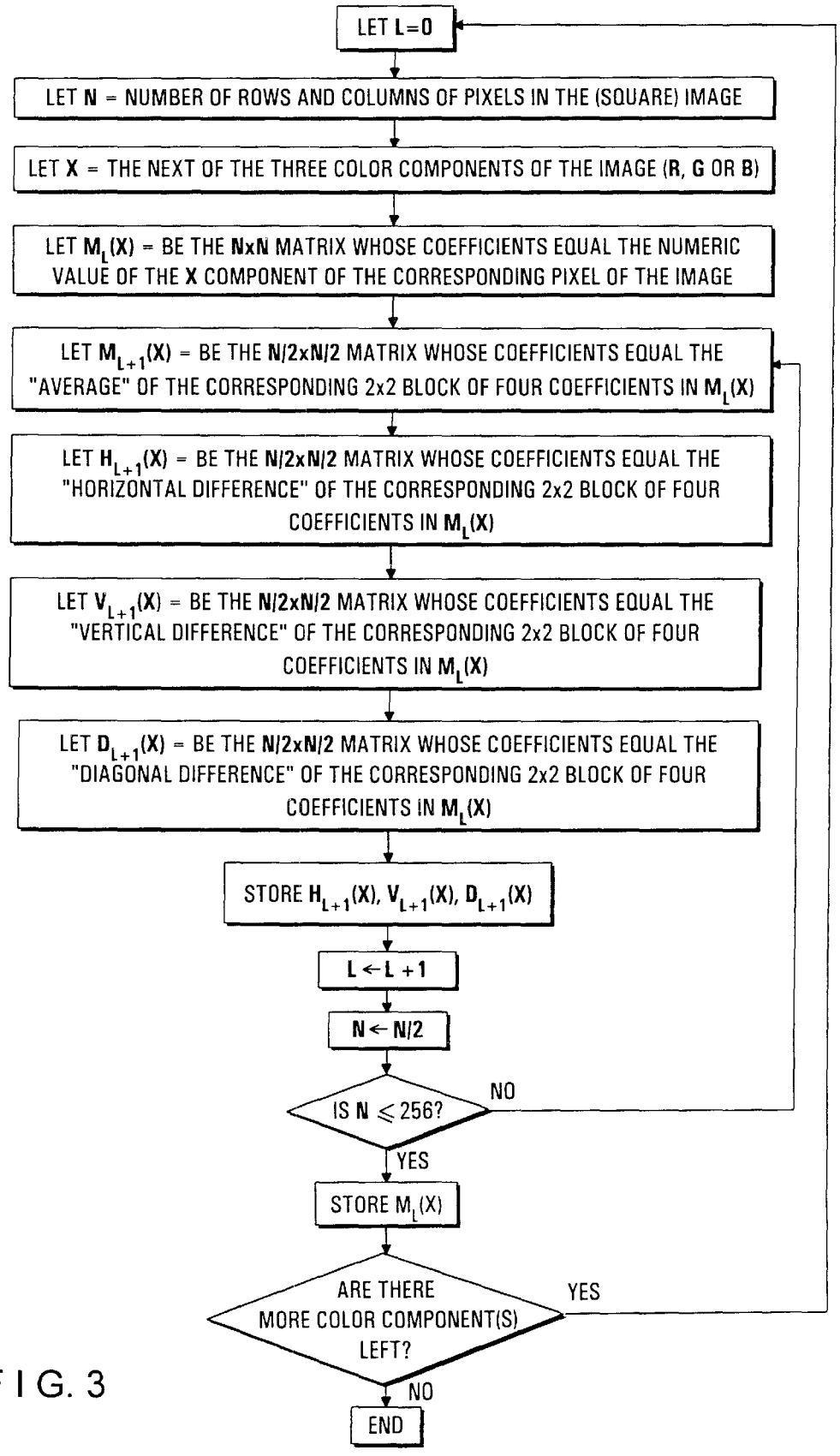

LET L=0
LET N = NUMBER OF ROWS AND COLUMNS OF PIXELS IN THE (SQUARE) IMAGE
LET X = THE NEXT OF THE THREE COLOR COMPONENTS OF THE IMAGE (R, G OR B)
LET $M_L(X)$ = BE THE NxN MATRIX WHOSE COEFFICIENTS EQUAL THE NUMERIC VALUE OF THE X COMPONENT OF THE CORRESPONDING PIXEL OF THE IMAGE
LET $M_{L+1}(X)$ = BE THE N/2xN/2 MATRIX WHOSE COEFFICIENTS EQUAL THE "AVERAGE" OF THE CORRESPONDING 2x2 BLOCK OF FOUR COEFFICIENTS IN $M_L(X)$
LET $H_{L+1}(X)$ = BE THE N/2xN/2 MATRIX WHOSE COEFFICIENTS EQUAL THE "HORIZONTAL DIFFERENCE" OF THE CORRESPONDING 2x2 BLOCK OF FOUR COEFFICIENTS IN $M_L(X)$
LET $V_{L+1}(X)$ = BE THE N/2xN/2 MATRIX WHOSE COEFFICIENTS EQUAL THE "VERTICAL DIFFERENCE" OF THE CORRESPONDING 2x2 BLOCK OF FOUR COEFFICIENTS IN $M_L(X)$
LET $D_{L+1}(X)$ = BE THE N/2xN/2 MATRIX WHOSE COEFFICIENTS EQUAL THE "DIAGONAL DIFFERENCE" OF THE CORRESPONDING 2x2 BLOCK OF FOUR COEFFICIENTS IN $M_L(X)$
STORE $H_{L+1}(X)$, $V_{L+1}(X)$, $D_{L+1}(X)$
L ← L + 1
N ← N/2
IS N ⩽ 256?
NO
YES
STORE $M_L(X)$
ARE THERE MORE COLOR COMPONENT(S) LEFT?
YES
NO
END

FIG. 3

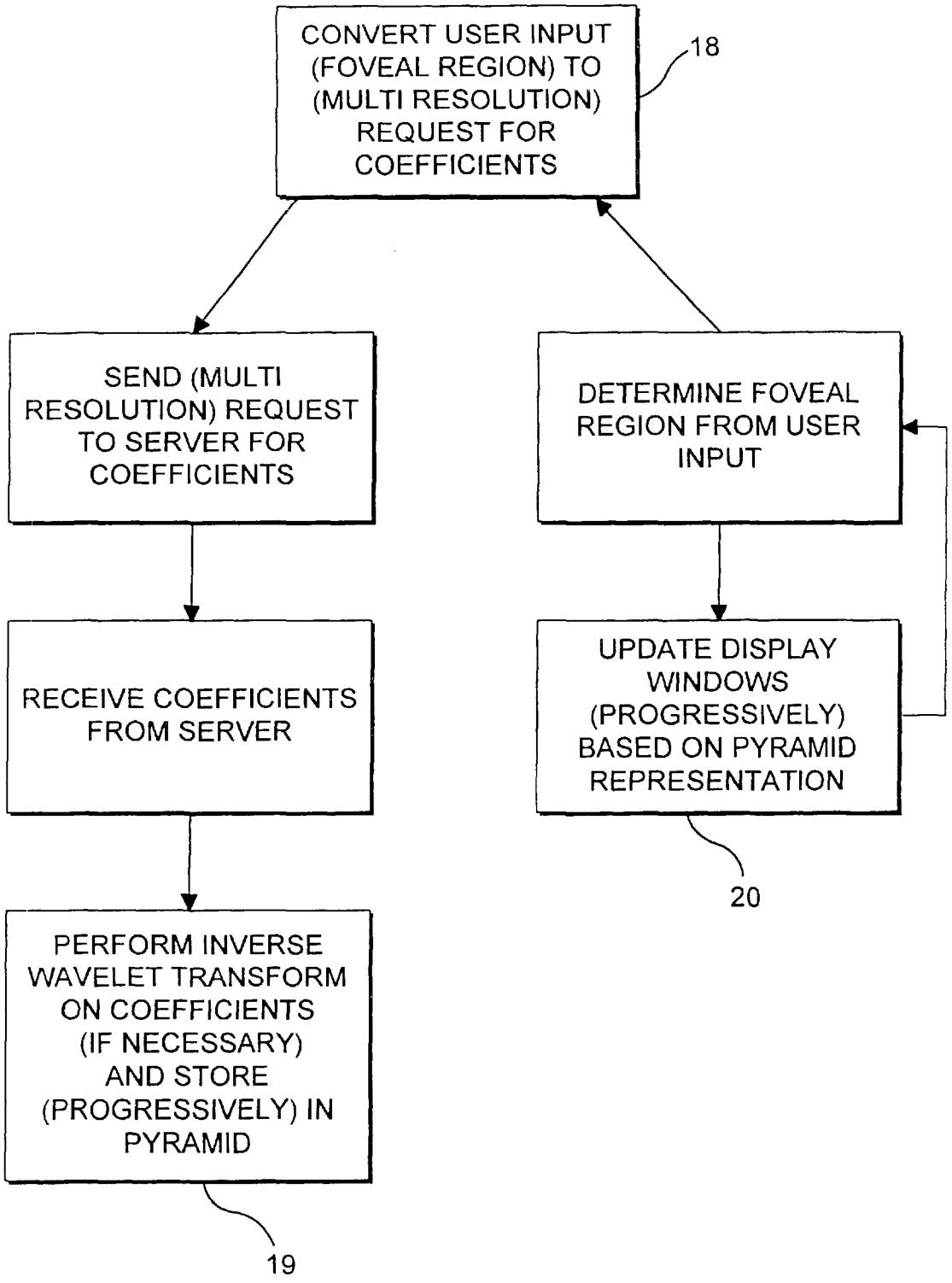

CONVERT USER INPUT (FOVEAL REGION) TO (MULTI RESOLUTION) REQUEST FOR COEFFICIENTS
18
SEND (MULTI RESOLUTION) REQUEST TO SERVER FOR COEFFICIENTS
DETERMINE FOVEAL REGION FROM USER INPUT
RECEIVE COEFFICIENTS FROM SERVER
UPDATE DISPLAY WINDOWS (PROGRESSIVELY) BASED ON PYRAMID REPRESENTATION
20
PERFORM INVERSE WAVELET TRANSFORM ON COEFFICIENTS (IF NECESSARY) AND STORE (PROGRESSIVELY) IN PYRAMID
19

FIG. 4

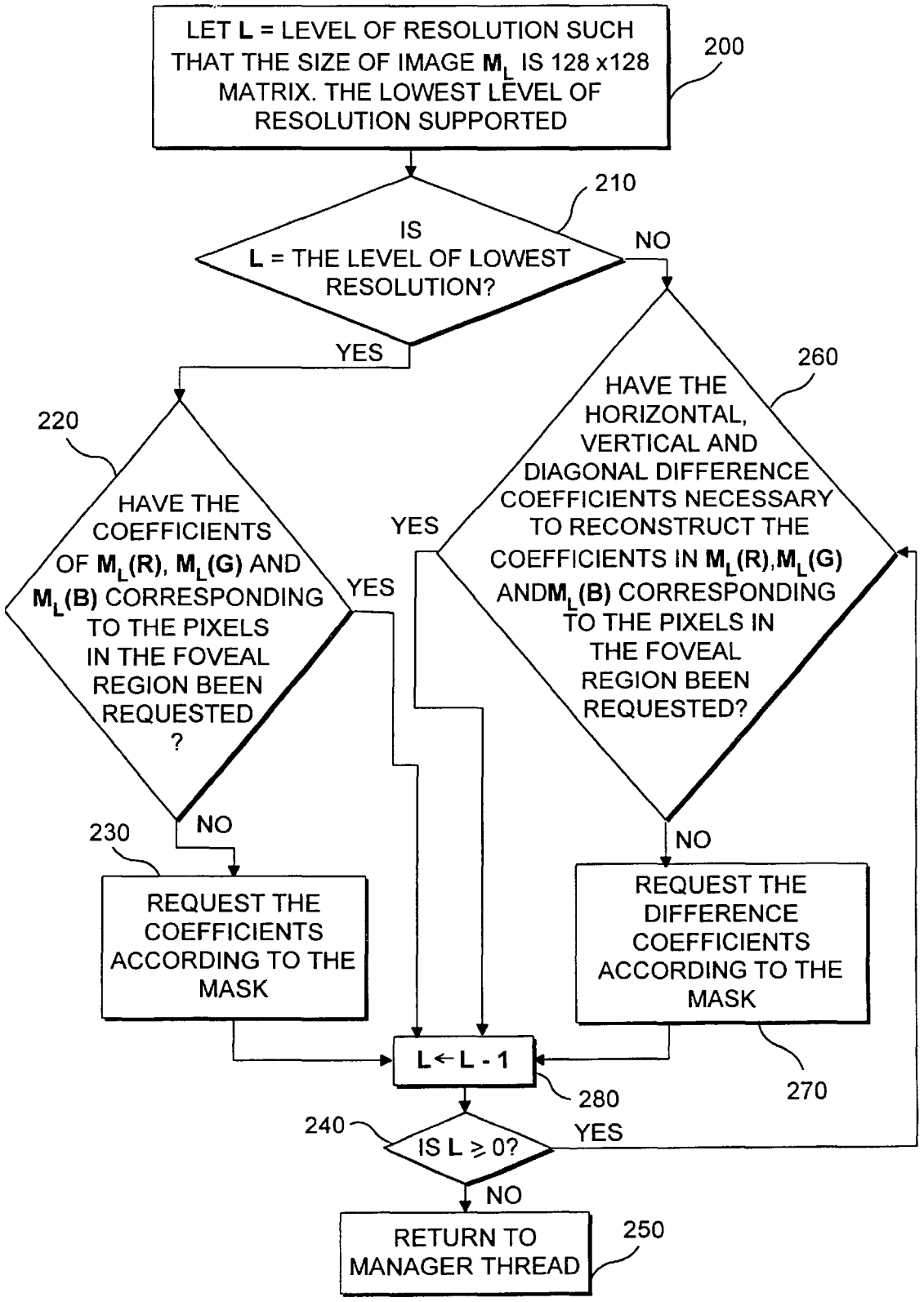

LET L = LEVEL OF RESOLUTION SUCH THAT THE SIZE OF IMAGE M_L IS 128 x128 MATRIX. THE LOWEST LEVEL OF RESOLUTION SUPPORTED
200
IS L = THE LEVEL OF LOWEST RESOLUTION?
210
NO
YES
HAVE THE COEFFICIENTS OF $M_L(R)$, $M_L(G)$ AND $M_L(B)$ CORRESPONDING TO THE PIXELS IN THE FOVEAL REGION BEEN REQUESTED ?
220
YES
NO
REQUEST THE COEFFICIENTS ACCORDING TO THE MASK
230
HAVE THE HORIZONTAL, VERTICAL AND DIAGONAL DIFFERENCE COEFFICIENTS NECESSARY TO RECONSTRUCT THE COEFFICIENTS IN $M_L(R)$,$M_L(G)$ AND$M_L(B)$ CORRESPONDING TO THE PIXELS IN THE FOVEAL REGION BEEN REQUESTED?
260
YES
NO
REQUEST THE DIFFERENCE COEFFICIENTS ACCORDING TO THE MASK
270
L←L - 1
280
IS L ≥ 0?
240
YES
NO
RETURN TO MANAGER THREAD
250

FIG. 5

APPARATUS AND METHOD FOR REALTIME VISUALIZATION USING USER-DEFINED DYNAMIC, MULTI-FOVEATED IMAGES

FIELD OF THE INVENTION

The present invention relates to a method and apparatus for serving images, even very large images, over a "thinwire" (e.g., over the Internet or any other network or application having bandwidth limitations).

BACKGROUND INFORMATION

The Internet, including the World Wide Web, has gained in popularity in recent years. The Internet enables clients/users to access information in ways never before possible over existing communications lines.

Often, a client/viewer desires to view and have access to relatively large images. For example, a client/viewer may wish to explore a map of a particular geographic location. The whole map, at highest (full) level of resolution will likely require a pixel representation beyond the size of the viewer screen in highest resolution mode.

One response to this restriction is for an Internet server to pre-compute many smaller images of the original image. The smaller images may be lower resolution (zoomed-out) views and/or portions of the original image. Most image archives use this approach. Clearly this is a sub-optimal approach since no preselected set of views can anticipate the needs of all users.

Some map servers (see, e.g., URLs http://www.mapquest.com and http://www.MapOnUs.com) use an improved approach in which the user may zoom and pan over a large image. However, transmission over the Internet involves significant bandwidth limitations (i.e transmission is relatively slow). Accordingly, such map servers suffer from at least three problems:

- Since a brand new image is served up for each zoom or pan request, visual discontinuities in the zooming and panning result. Another reason for this is the discrete nature of the zoom/pan interface controls.
- Significantly less than realtime response.
- The necessarily small fixed size of the viewing window (typically about 3"×4.5"). This does not allow much of a perspective.

To generalize, what is needed is an apparatus and method which allows realtime visualization of large scale images over a "thinwire" model of computation. To put it another way, it is desirable to optimize the model which comprises an image server and a client viewer connected by a low bandwidth line.

One approach to the problem is by means of progressive transmission. Progressive transmission involves sending a relatively low resolution version of an image and then successively transmitting better resolution versions. Because the first, low resolution version of the image requires far less data than the full resolution version, it can be viewed quickly upon transmission. In this way, the viewer is allowed to see lower resolution versions of the image while waiting for the desired resolution version. This gives the transmission the appearance of continuity. In addition, in some instances, the lower resolution version may be sufficient or may in any event exhaust the display capabilities of the viewer display device (e.g., monitor).

Thus, R. L. White and J. W. Percival, "Compression and Progressive Transmission of Astronomical Images," *SPIE Technical Conference* 2199, 1994, describes a progressive transmission technique based on bit planes that is effective for astronomical data.

However, utilizing progressive transmission barely begins to solve the "thinwire" problem. A viewer zooming or panning over a large image (e.g., map) desires realtime response. This of course is not achieved if the viewer must wait for display of the desired resolution of a new quadrant or view of the map each time a zoom and pan is initiated. Progressive transmission does not achieve this realtime response when it is the higher resolution versions of the image which are desired or needed, as these are transmitted later.

The problem could be effectively solved, if, in addition to variable resolution over time (i.e, progressive transmission), resolution is also varied over the physical extent of the image.

Specifically, using foveation techniques, high resolution data is transmitted at the user's gaze point but with lower resolution as one moves away from that point. The very simple rationale underlying these foveation techniques is that the human field of vision (centered at the gaze point) is limited. Most of the pixels rendered at uniform resolution are wasted for visualization purposes. In fact, it has been shown that the spatial resolution of the human eye decreases exponentially away from the center gaze point. E. L. Schwartz, "The Development of Specific Visual Projections in the Monkey and the Goldfish: Outline of a Geometric Theory of Receptotopic Structure," *Journal of Theoretical Biology,* 69:655–685, 1977

The key then is to mimic the movements and spatial resolution of the eye. If the user's gaze point can be tracked in realtime and a truly multi-foveated image transmitted (i.e., a variable resolution image mimicking the spatial resolution of the user's eye from the gaze point), all data necessary or useful to the user would be sent, and nothing more. In this way, the "thinwire" model is optimized, whatever the associated transmission capabilities and bandwidth limitations.

In practice, in part because eye tracking is imperfect, using multi-foveated images is superior to atempting display of an image portion of uniform resolution at the gaze point.

There have in fact been attempts to achieve multifoveated images in a "thinwire" environment.

F. S. Hill Jr., Sheldon Walker Jr. and Fuwen Gao, "Interactive Image Query System Using Progressive Transmission," *Computer Graphics,* 17(3), 1983, describes progressive transmission and a form of foveation for a browser of images in an archive. The realtime requirement does not appear to be a concern.

T. H. Reeves and J. A. Robinson, "Adaptive Foveation of MPEG Video," *Proceedings of the* 4th *ACM International Multimedia Conference,* 1996, gives a method to foveate MPEG-standard video in a thin-wire environment. MPEG-standard could provide a few levels of resolution but they consider only a 2-level foveation. The client/viewer can interactively specify the region of interest to the server/sender.

R. S. Wallace and P. W. Ong and B. B. Bederson and E. L. Schwartz, "Space-variant image processing". Intl. J. Of Computer Vision, 13:1 (1994) 71–90 discusses space-variant images in computer vision. "Space-Variant" may be regarded as synonymous with the term "multifoveated" used above. A biological motivation for such images is the complex logmap model of the transformation from the retina to the visual cortex (E. L. Schwartz, "A quantitative model of the functional architecture of human striate cortex with

application to visual illusion and cortical texture analysis", Biological Cybernetics, 37(1980) 63–76).

Philip Kortum and Wilson S. Geisler, "Implementation of a Foveated Image Coding System For Image Bandwidth Reduction," *Human Vision and Electronic Imaging, SPIE Proceedings Vol.* 2657, 350–360, 1996, implement a real time system for foveation-based visualization. They also noted the possibility of using foveated images to reduce bandwidth of transmission.

M. H. Gross, O. G. Staadt and R. Gatti, "Efficient triangular surface approximations using wavelets and quadtree data structures", IEEE Trans, On Visualization and Computer Graphics, 2(2), 1996, uses wavelets to produce multifoveated images.

Unfortunately, each of the above attempts are essentially based upon fixed super-pixel geometries, which amount to partitioning the visual field into regions of varying (predetermined) sizes called super-pixels, and assigning the average value of the color in the region to the super-pixel. The smaller pixels (higher resolution) are of course intended to be at the gaze point, with progressively larger super-pixels (lower resolution) about the gaze point.

However, effective real-time visulization over a "thin wire" requires precision and flexibility. This cannot be achieved with a geometry of predetermined pixel size. What is needed is a flexible foveation technique which allows one to modify the position and shape of the basic foveal regions, the maximum resolution at the foveal region and the rate at which the resolution falls away. This will allow the "thinwire" model to be optimized.

In addition, none of the above noted references addresses the issue of providing multifoveated images that can be dynamically (incrementally) updated as a function of user input. This property is crucial to the solution of the thinwire problem, since it is essential that information be "streamed" at a rate that optimally matches the bandwidth of the network with the human capacity to absorb the visual information.

SUMMARY OF THE INVENTION

The present invention overcomes the disadvantages of the prior art by utilizing means for tracking or approximating the user's gaze point in realtime and, based on the approximation, transmitting dynamic multifoveated image (s) (i.e., a variable resolution image over its physical extent mimicking the spatial resolution of the user's eye about the approximated gaze point) updated in realtime.

"Dynamic" means that the image resolution is also varying over time. The user interface component of the present invention may provide a variety of means for the user to direct this multifoveation process in real time.

Thus, the invention addresses the model which comprises an image server and a client viewer connected by a low bandwidth line. In effect, the invention reduces the bandwidth from server to client, in exchange for a very modest increase of bandwidth from the client to the server

Another object of the invention is that it allows realtime visualization of large scale images over a "thinwire" model of computation.

An additional advantage is the new degree of user control provided for realtime, active, visualization of images (mainly by way of foveation techniques). The invention allows the user to determine and change in realtime, via input means (for example, without limitation, a mouse pointer or eye tracking technology), the variable resolution over the space of the served up image(s).

An additional advantage is that the invention demonstrates a new standard of performance that can be achieved by large-scale image servers on the World Wide Web at current bandwidth or even in the near future.

Note also, the invention has advantages over the traditional notion of progressive transmission, which has no interactivity. Instead, the progressive transmission of an image has been traditionally predetermined when the image file is prepared. The invention's use of dynamic (constantly changing in realtime based on the user's input) multifoveated images allows the user to determine how the data are progressively transmitted.

Other advantages of the invention include that it allows the creation of the first dynamic and a more general class of multifoveated images. The present invention can use wavelet technology. The flexibility of the foveation approach based on wavelets allows one to easily modify the following parameters of a multifoveated image: the position and shape of the basic foveal region(s), the maximum resolution at the foveal region(s), and the rate at which the resolution falls away. Wavelets can be replaced by any multi resolution pyramid schemes. But it seems that wavelet-based approaches are preferred as they are more flexible and have the best compression properties.

Another advantage is the present invention's use of dynamic data structures and associated algorithms. This helps optimize the "effective real time behavior" of the system. The dynamic data structures allow the use of "partial information" effectively. Here information is partial in the sense that the resolution at each pixel is only partially known. But as additional information is streamed in, the partial information can be augmented. Of course, this principle is a corollary to progressive transmission.

Another advantage is that the dynamic data structures may be well exploited by the special architecture of the client program. For example, the client program may be multi-threaded with one thread (the "manager thread") designed to manage resources (especially bandwidth resources). This manager is able to assess network congestion, and other relevant parameters, and translate any literal user request into the appropriate level of demand for the network. For example, when the user's gaze point is focused on a region of an image, this may be translated into requesting a certain amount, say, X bytes of data. But the manager can reduce this to a request over the network of (say) X/2 bytes of data if the traffic is congested, or if the user is panning very quickly.

Another advantage of the present invention is that the server need send only that information which has not yet been served. This has the advantage of reducing communication traffic.

Further objects and advantages of the invention will become apparent from a consideration of the drawings and ensuing description.

BRIEF DESRIPTION OF DRAWINGS

FIG. **1** shows an embodiment of the present invention including a server, and client(s) as well as their respective components.

FIG. **2***a* illustrates one level of a particular wavelet transform, the Haar wavelet transform, which the server may execute in one embodiment of the present invention.

FIG. **2***b* illustrates one level of the Haar inverse wavelet transform.

FIG. **3** is a flowchart showing an algorithm the server may execute to perform a Haar wavelet transform in one embodiment of the present invention.

FIG. 4 shows Manager, Display and Network threads, which the client(s) may execute in one embodiment of the present invention.

FIG. 5 is a more detailed illustration of a portion of the Manager thread depicted in FIG. 4.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 1 depicts an overview of the components in an exemplary embodiment of the present invention. A server 1 is comprised of a storage device 3, a memory device 7 and a computer processing device 4. The storage device 3 can be implemented as, for example, an internal hard disk, Tape Cartridge, or CD-ROM. The faster access and greater storage capacity the storage device 3 provides, the more preferable the embodiment of the present invention. The memory device 7 can be implemented as, for example, a collection of RAM chips.

The processing device 4 on the server 1 has network protocol processing element 12 and wavelet transform element 13 running off it. The processing device 4 can be implemented with a single microprocessor chip (such as an Intel Pentium chip), printed circuit board, several boards or other device. Again, the faster the speed of the processing device 4, the more preferable the embodiment. The network protocol processing element 12 can be implemented as a separate "software" (i.e., a program, sub-process) whose instructions are executed by the processing device 4. Typical examples of such protocols include TCP/IP (the Internet Protocol) or UDP (User Datagram Protocol). The wavelet transform element 13 can also be implemented as separate "software" (i.e., a program, sub-process) whose instructions are executed by the processing device 4.

In a preferred embodiment of the present invention, the server 1 is a standard workstation or Pentium class system. Also, TCP/IP processing may be used to implement the network protocol processing element 12 because it reduces complexity of implementation. Although a TCP/IP implementation is simplest, it is possible to use the UDP protocol subject to some basic design changes. The relative advantage of using TCP/IP as against UDP is to be determined empirically. An additional advantage of using modern, standard network protocols is that the server 1 can be constructed without knowing anything about the construction of its client(s) 2.

According to the common design of modern computer systems, the most common embodiments of the present invention will also include an operating system running off the processing means device 4 of the server 1. Examples of operating systems include, without limitation, Windows 95, Unix and Windows NT. However, there is no reason a processing device 4 could not provide the functions of an "operating system" itself.

The server 1 is connected to a client(s) 2 in a network. Typical examples of such servers 1 include image archive servers and map servers on the World Wide Web.

The client(s) 2 is comprised of a storage device 3, memory device 7, display 5, user input device 6 and processing device 4. The storage device 3 can be implemented as, for example, an internal hard disks, Tape Cartridge, or CD-ROM. The faster access and greater storage capacity the storage device 3 provides, the more preferable the embodiment of the present invention. The memory device 7 can be implemented as, for example, a collection of RAM chips. The display 5 can be implemented as, for example, any monitor, whether analog or digital. The user input device 6 can be implemented as, for example, a keyboard, mouse, scanner or eye-tracking device.

The client 2 also includes a processing device 4 with network protocol processing element 12 and inverse wavelet transform element means 14 running off it. The processing device 4 can be implemented as, for example, a single microprocessor chip (such as an Intel Pentium chip), printed circuit board, several boards or other device. Again, the faster the run time of the processing device 4, the more preferable the embodiment. The network protocol processing element 12 again can be implemented as a separate "software" (i.e., a program, sub-process) whose instructions are executed by the processing device 4. Again, TCP/IP processing may be used to implement the network protocol processing element 12. The inverse wavelet transform element 14 also may be implemented as separate "software." Also running off the processing device 4 is a user input conversion mechanism 16, which also can be implemented as "software."

As with the server 1, according to the common design of modern computer systems, the most common embodiments of the present invention will also include an operating system running off the processing device 4 of the client(s) 2.

In addition, if the server 1 is connected to the client(s) 2 via a telephone system line or other systems/lines not carrying digital pulses, the server 1 and client(s) 2 both also include a communications converter device 15. A communications converter device 15 can be implemented as, for example, a modem. The communications converter device 15 converts digital pulses into the frequency/signals carried by the line and also converts the frequency/signals back into digital pulses, allowing digital communication.

In the operation of the present invention, the extent of computational resources (e.g., storage capacity, speed) is a more important consideration for the server 1, which is generally shared by more than one client 2, than for the client(s) 2.

In typical practice of the present invention, the storage device 3 of the server 1 holds an image file, even a very large image file. A number of client 2 users will want to view the image.

Prior to any communication in this regard between the server 1 and client(s) 2, the wavelet transform element 13 on the server 1 obtains a wavelet transform on the image and stores it in the storage device 3.

There has been extensive research in the area of wavelet theory. However, briefly, to illustrate, "wavelets" are defined by a group of basis functions which, together with coefficients dependant on an input function, can be used to approximate that function over varying scales, as well as represent the function exactly in the limit. Accordingly, wavelet coefficients can be categorized as "average" or "approximating coefficients" (which approximate the function) and "difference coefficients" (which can be used to reconstruct the original function exactly). The particular approximation used as well as the scale of approximation depend upon the wavelet bases chosen. Once a group of basis functions is chosen, the process of obtaining the relevant wavelet coefficients is called a wavelet transform.

In the preferred embodiment, the Haar wavelet basis functions are used. Accordingly, in the preferred embodiment, the wavelet transform element 13 on the server 1 performs a Haar wavelet transform on a file representation of the image stored in the storage device 3, and then stores the transform on the storage device 3. However, it is readily apparent to anyone skilled in the art that any of the wavelet

family of transforms may be chosen to implement the present invention.

Note that once the wavelet transform is stored, the original image file need not be kept, as it can be reconstructed exactly from the transform.

FIG. **2** illustrates one step of the Haar wavelet transform. Start with an n by n matrix of coefficients **17** whose entries correspond to the numeric value of a color component (say, Red, Green or Blue) of a square screen image of n by n pixels. Divide the original matrix **17** into 2 by 2 blocks of four coefficients, and for each 2×2 block, label the coefficient in the first column, first row "a,"; second column, first row "b"; second row, first column "c"; and second row, second column "d."

Then one step of the Haar wavelet transform creates four n/2 by n/2 matrices. The first is an n/2 by n/2 approximation matrix **8** whose entries equal the "average" of the corresponding 2 by 2 block of four coefficients in the original matrix **17**. As is illustrated in FIG. **2**, the coefficient entries in the approximation matrix **8** are not necessarily equal to the average of the corresponding four coefficients a, b, c and d (i.e., a'=(a+b+c+d)/4) in the original matrix **17**. Instead, here, the "average" is defined as (a+b+c+d)/2.

The second is an n/2 by n/2 horizontal difference matrix **10** whose entries equal b'=(a+b−c−d)/2, where a, b, c and d are, respectively, the corresponding 2×2 block of four coefficients in the original matrix **17**. The third is an n/2 by n/2 vertical difference matrix **9** whose entries equal c'=(a−b+c−d)/2, where a, b, c and d are, respectively, the corresponding 2×2 block of four coefficients in the original matrix **17**. The fourth is an n/2 by n/2 diagonal difference matrix **11** whose entries equal d'=(a−b−c+d)/2, where a, b, c and d are, respectively, the corresponding 2×2 block of four coefficients in the original matrix **17**.

A few notes are worthy of consideration. First, the entries a', b', c', d' are the wavelet coefficients. The approximation matrix **8** is an approximation of the original matrix **17** (using the "average" of each 2×2 group of 4 pixels) and is one fourth the size of the original matrix **17**.

Second, each of the 2×2 blocks of four entries in the original matrix **17** has one corresponding entry in each of the four n/2 by n/2 matrices. Accordingly, it can readily be seen from FIG. **2** that each of the 2×2 blocks of four entries in the original matrix **17** can be reconstructed exactly, and the transformation is invertible. Therefore, the original matrix **17** representation of an image can be discarded during processing once the transform is obtained.

Third, the transform can be repeated, each time starting with the last approximation matrix **8** obtained, and then discarding that approximation matrix **8** (which can be reconstructed) once the next wavelet step is obtained. Each step of the transform results in approximation and difference matrices ½ the size of the approximation matrix **8** of the prior step.

Retracing each step to synthesize the original matrix **17** is called the inverse wavelet transform, one step of which is depicted in FIG. **2***b*.

Finally, it can readily be seen that the approximation matrix **8** at varying levels of the wavelet transform can be used as a representation of the relevant color component of the image at varying levels of resolution.

Conceptually then, the wavelet transform is a series of approximation and difference matrices at various levels (or resolutions). The number of coefficients stored in a wavelet transform is equal to the number of pixels in the original matrix **17** image representation. (However, the number of bits in all the coefficients may differ from the number of bits in the pixels. Applying data compression to coefficients turns out to be generally more effective on coefficients.) If we assume the image is very large, the transform matrices must be further decomposed into blocks when stored on the storage means **3**.

FIG. **3** is a flowchart showing one possible implementation of the wavelet transform element **13** which performs a wavelet transform on each color component of the original image. As can be seen from the flowchart, the transform is halted when the size of the approximation matrix is 256×256, as this may be considered the lowest useful level of resolution.

Once the wavelet transform element **13** stores a transform of the image(s) in the storage means **3** of the server **1**, the server **1** is ready to communicate with client(s) **2**.

In typical practice of the invention the client **2** user initiates a session with an image server **1** and indicates an image the user wishes to view via user input means **6**. The client **2** initiates a request for the 256 by 256 approximation matrix **8** for each color component of the image and sends the request to the server **1** via network protocol processing element **12**. The server **1** receives and processes the request via network protocol processing element **12**. The server **1** sends the 256 by 256 approximation matrices **8** for each color component of the image, which the client **2** receives in similar fashion. The processing device **4** of the client **2** stores the matrices in the storage device **3** and causes a display of the 256 by 256 version of the image on the display **5**. It should be appreciated that the this low level of resolution requires little data and can be displayed quickly. In a map server application, the 256 by 256, coarse resolution version of the image may be useful in a navigation window of the display **5**, as it can provide the user with a position indicator with respect to the overall image.

A more detailed understanding of the operation of the client **2** will become apparent from the discussion of the further, continuous operation of the client **2** below.

Continuous operation of the client(s) **2** is depicted in FIG. **4**. In the preferred embodiment, the client(s) **2** processing device may be constructed using three "threads," the Manager thread **18**, the Network Thread **19** and the Display Thread **20**. Thread programming technology is a common feature of modern computers and is supported by a variety of platforms. Briefly, "threads" are processes that may share a common data space. In this way, the processing means can perform more than one task at a time. Thus, once a session is initiated, the Manager Thread **18**, Network Thread **19** and Display Thread **20** run simultaneously, independently and continually until the session is terminated. However, while "thread technology" is preferred, it is unnecessary to implement the client(s) **2** of the present invention.

The Display Thread **20** can be based on any modern windowing system running off the processing device **4**. One function of the Display Thread **20** is to continuously monitor user input device **6**. In the preferred embodiment, the user input device **6** consists of a mouse or an eye-tracking device, though there are other possible implementations. In a typical embodiment, as the user moves the mouse position, the current position of the mouse pointer on the display **5** determines the foveal region. In other words, it is presumed the user gaze point follows the mouse pointer, since it is the user that is directing the mouse pointer. Accordingly, the display thread **20** continuously monitors the position of the mouse pointer.

In one possible implementation, the Display Thread **20** places user input requests (i.e., foveal regions determined from user input device **6**) as they are obtained in a request queue. Queue's are data structures with first-in-first-out characteristics that are generally known in the art.

The Manager Thread **18** can be thought of as the brain of the client **2**. The Manager Thread **18** converts the user input request in the request queue into requests in the manager request queue, to be processed by the Network Thread **19**. The user input conversion mechanism **16** converts the user determined request into a request for coefficients.

A possible implementation of user input conversion mechanism **16** is depicted in the flow chart in FIG. **5**. Essentially, the user input conversion mechanism **16** requests all the coefficient entries corresponding to the foveal region in the horizontal difference **10** matrices, vertical difference **9** matrices, diagonal difference matrices **11** and approximation matrix **8** of the wavelet transform of the image at each level of resolution. (Recall that only the last level approximation matrix **8** needs to be stored by the server **1**.) That is, wavelet coefficients are requested such that it is possible to reconstruct the coefficients in the original matrix **17** corresponding to the foveal region.

As the coefficients are included in the request, they are masked out. The use of a mask is commonly understood in the art. The mask is maintained to determine which coefficients have been requested so they are not requested again. Each mask can be represented by an array of linked lists (one linked list for each row of the image at each level of resolution).

As shown in FIG. **5**, the input conversion mechanism **16** determines the current level of resolution ("L") of an image ("M_L") such that the image M_L is, e.g., 128×128 pixel matrix (for example, the lowest supported resolution), as shown in Step **200**. Then, the input conversion mechanism **16** determines if the current level L is the lowest resolution level (Step **210**). If so, it is determined if the three color coefficients (i.e., $M_L(R)$, $M_L(G)$, and $M_L(B)$) correspond to the foveal region that has been requested (Step **220**). If that is the case, then the input conversion mechanism **16** confirms that the current region L is indeed the lowest resolution region (Step **240**), and returns the control to the Manager Thread **18** (Step **250**). If, in Step **220**, it is determined that the three color coefficients have not been requested, these coefficients are requested using the mask described above, and the process continues to Step **240**, and the control is returned to the Manager Thread **18** (Step **250**).

If, in Step **210**, it is determined that the current level L is not the lowest resolution level, then the input conversion mechanism **16** determines whether the horizontal, vertical and diagonal difference coefficients (which are necessary to reconstruct the three color coefficients) have been requested (Step **260**). If so, then the input conversion mechanism **16** skips to Step **280** to decrease the current level L by 1. Otherwise a set of difference coefficients may be requested. This set depends on the mask and the foveal parameters (e.g., a shape of the foveal region, a maximum resolution, a rate of decay of the resolution, etc.). The user may select "formal" values for these foveal parameters, but the Manager Thread **18** may, at this point, select the "effective" values for these parameters to ensure a trade-off between (1) achieving a reasonable response time over the estimated current network bandwidth, and (2) achieving a maximum throughput in the transmission of data. The process then continues to Step **280**. Thereafter, the input conversion mechanism **16** determines whether the current level L is greater or equal to zero (Step **240**). If that is the case, the process loops back to step **260**. Otherwise, the control is returned to the Manager Thread **18** (Step **250**).

The Network Thread **19** includes the network protocol processing element **12**. The Network Thread obtains the (next) multi-resolution request for coefficients corresponding to the foveal region from request queue and processes and sends the request to the server **1** via network protocol processing element **12**.

Notice that the data requested is "local" because it represents visual information in the neighborhood of the indicated part of the image. The data is incremental because it represents only the additional information necessary to increase the resolution of the local visual information. (Information already available locally is masked out).

The server **1** receives and processes the request via network protocol processing element **12**, and sends the coefficients requested. When the coefficients are sent, they are masked out. The mask is maintained to determine which coefficients have been sent and for deciding which blocks of data can be released from main memory. Thus, an identical version of the mask is maintained on both the client **2** side and server **1** side.

The Network Thread **19** of the client **2** receives and processes the coefficients. The Network Thread **19** also includes inverse wavelet transform element **14**. The inverse wavelet transform element **14** performs an inverse wavelet transform on the received coefficients and stores the resulting portion of an approximation matrix **8** each time one is obtained (i.e., at each level of resolution) in the storage device **3** of the client **2**. The sub-image is stored at each (progressively higher, larger and less course) level of its resolution.

Note that as the client **2** knows nothing about the image until it is gradually filled in as coefficients are requested. Thus, sparse matrices (sparse, dynamic data structures) and associated algorithms can be used to store parts of the image received from the server **1**. Sparse matrices are known in the art and behave like normal matrices except that the memory space of the matrix are not allocated all at once. Instead the memory is allocated in blocks of sub-matrices. This is reasonable as the whole image may require a considerable amount of space.

Simultaneously, the Display thread **20** (which can be implemented using any modern operating system or windowing system) updates the display **5** based on the pyramid representation stored in the storage device **3**.

Of course, the Display thread **20** continues its monitoring of the user input device **6** and the whole of client **2** processing continues until the session is terminated.

A few points are worthy of mention. Notice that since lower, coarser resolution images will be stored on the client **2** first, they are displayed first Also, the use of foveated images ensures that the incremental data to update the view is small, and the requested data can arrive within the round trip time of a few messages using, for example, the TCP/IP protocol.

Also notice, that a wavelet coefficient at a relatively coarser level of resolution corresponding to the foveal region affects a proportionately larger part of the viewer's screen than a coefficient at a relatively finer level of resolution corresponding to the foveal region (in fact, the resolution on the display **5** exponentially away from the mouse pointer). Also notice the invention takes advantage of progressive transmission, which gives the image perceptual continuity. But unlike the traditional notion of progressive

transmission, it is the client **2** user that is determining transmission ordering, which is not pre-computed because the server **1** doesn't know what the client(s) **2** next request will be. Thus, as noted in the objects and advantages section, the "thinwire" model is optimized.

Note that in the event the thread technology is utilized to implement the present invention, semaphores data structures are useful if the threads share the same data structures (e.g., the request queue). Semaphores are well known in the art and ensure that only one simultaneous process (or "thread") can access and modify a shared data structure at one time. Semaphores are supported by modern operating systems.

CONCLUSION

It is apparent that various useful modifications can be made to the above description while remaining within the scope of the invention.

For example, without limitation, the user can be provided with two modes for display: to always fill the pixels to the highest resolution that is currently available locally or to fill them up to some user specified level. The client **2** display **5** may include a re-sizable viewing window with minimal penalty on the realtime performance of the system. This is not true of previous approaches. There also may be an auxiliary navigation window (which can be re-sized but is best kept fairly small because it displays the entire image at a low resolution). The main purpose of such a navigation window would be to let the viewer know the size and position of the viewing window in relation to the whole image.

It is readily seen that further modifications within the scope of the invention provide further advantages to the user. For example, without limitation, the invention may have the following capabilities: continuous realtime panning, continuous realtime zooming, foveating, varying the foveal resolution and modification of the shape and size of the foveal region. A variable resolution feature may also allow the server **1** to dynamically adjust the amount of transmitted data to match the effective bandwidth of the network.

While the above description contains many specificities, these should not be construed as limitations on the scope of the invention, but rather as an exemplification of one preferred embodiment thereof. Many other variations are possible. Accordingly, the scope of the invention should be determined not by the embodiment(s) illustrated, but by the appended claims and their legal equivalents.

What is claimed is:

1. A client apparatus for enabling a realtime visualization of at least one image, the client apparatus comprising:

a storage device storing first data corresponding to a multifoveated representation of an original image,

a user input device providing second data corresponding to at least one visualization command of at least one user; and

a processing arrangement generating third data corresponding to a multifoveated image using the first data, the second data and a foveation operator.

2. The client apparatus of claim **1**, further comprising a network protocol processing element which provides the third data using a TCP/IP protocol.

3. The client apparatus of claim **1**, wherein the processing element transmits the third data to the at least one client via the Internet.

4. The client apparatus of claim **1**, wherein the user input device includes a mouse device.

5. The client apparatus of claim **1**, wherein the user input device includes at least one of an eye-tracking device and a keyboard.

6. The client apparatus of claim **1**, wherein the foveation operator is specified using parameters that include at least one of:

a set of foveation points,

a shape of a foveated region,

a maximum resolution of the foveated region, and

a rate at which a maximum resolution of the foveal region decays.

7. The client apparatus of claim **1**,

wherein the processing arrangement receives the original image from a server, and

wherein the memory arrangement stores a data structure representing the multifoveated image, the data structure that is optimized for the client apparatus being independent of an image representation provided by a server.

8. The client apparatus of claim **1**, wherein the third data corresponding to the multifoveated image is generated for at least one of

a first arbitrary-shaped foveal region,

a second arbitrarily-fine foveal region, and

an arbitrary union of the first and second foveal regions.

* * * * *

US005179638A

United States Patent [19]

Dawson et al.

[11] Patent Number: 5,179,638

[45] Date of Patent: Jan. 12, 1993

[54] METHOD AND APPARATUS FOR GENERATING A TEXTURE MAPPED PERSPECTIVE VIEW

[75] Inventors: John F. Dawson; Thomas D. Snodgrass; James A. Cousens, all of Albuquerque, N. Mex.

[73] Assignee: Honeywell Inc., Minneapolis, Minn.

[21] Appl. No.: 514,598

[22] Filed: Apr. 26, 1990

[51] Int. Cl.[5] G06F 15/62

[52] U.S. Cl. 395/125; 395/127; 395/130

[58] Field of Search 395/125, 126, 127, 130; 364/443, 723; 340/729

[56] References Cited

U.S. PATENT DOCUMENTS

4,677,576	6/1987	Berlin, Jr. et al.	395/127 X
4,876,651	10/1989	Dawson et al.	395/126 X
4,884,220	11/1989	Dawson et al.	395/125
4,899,293	2/1990	Dawson et al.	395/125 X
4,940,972	7/1990	Mouchot et al.	395/125 X
4,985,854	1/1991	Wittenburg	395/126 X
5,020,014	5/1991	Miller et al.	364/723

Primary Examiner—Gary V. Harkcom
Assistant Examiner—Mark K. Zimmerman
Attorney, Agent, or Firm—Ronald E. Champion; George A. Leone, Sr.

[57] ABSTRACT

A method and apparatus for providing a texture mapped perspective view for digital map systems. The system includes apparatus for storing elevation data, apparatus for storing texture data, apparatus for scanning a projected view volume from the elevation data storing apparatus, apparatus for processing, apparatus for generating a plurality of planar polygons and apparatus for rendering images. The processing apparatus further includes apparatus for receiving the scanned projected view volume from the scanning apparatus, transforming the scanned projected view volume from object space to screen space, and computing surface normals at each vertex of each polygon so as to modulate texture space pixel intensity. The generating apparatus generates the plurality of planar polygons from the transformed vertices and supplies them to the rendering apparatus which then shades each of the planar polygons. In one alternate embodiment of the invention, the polygons are shaded by apparatus of the rendering apparatus assigning one color across the surface of each polygon. In yet another alternate embodiment of the invention, the rendering apparatus interpolates the intensities between the vertices of each polygon in a linear fashion as in Gouraud shading.

8 Claims, 7 Drawing Sheets

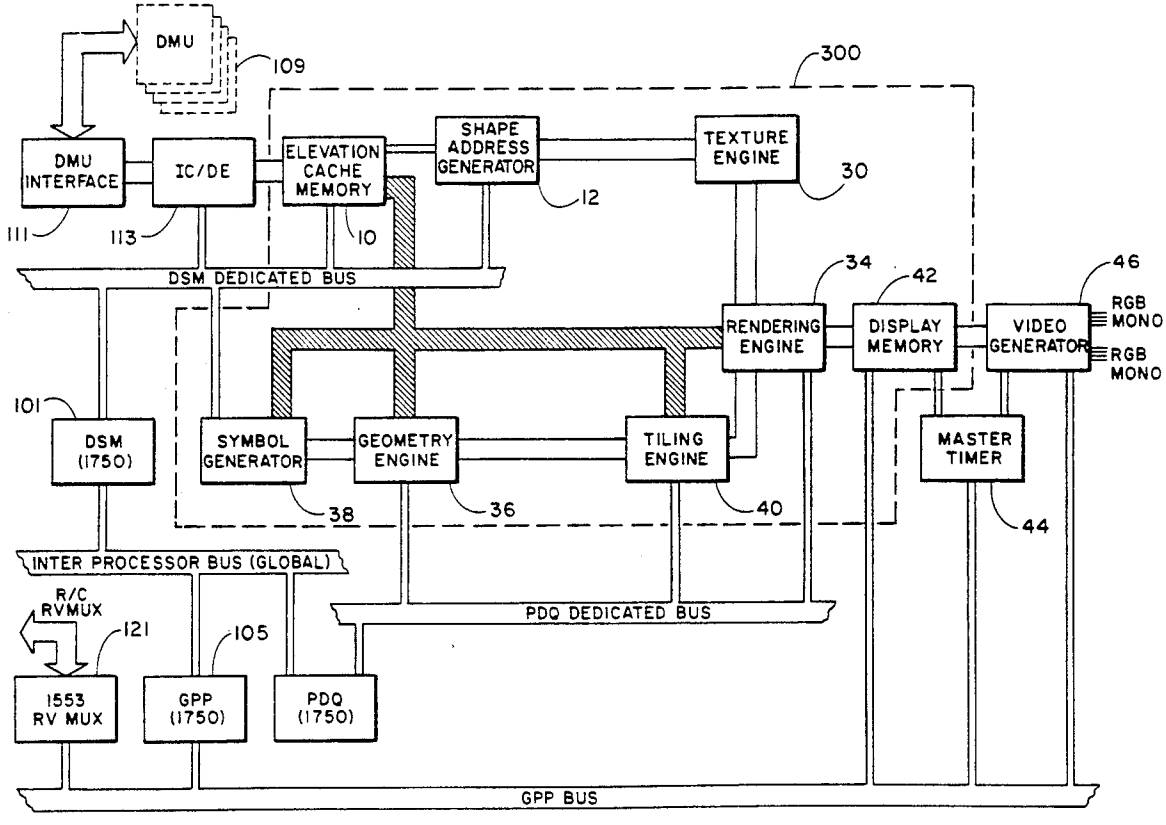


Fig.-1

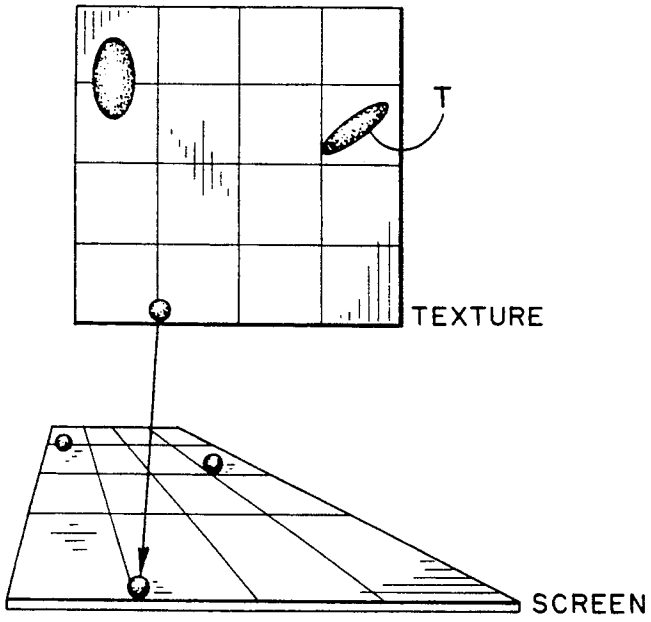

T
TEXTURE
SCREEN

Fig.-2

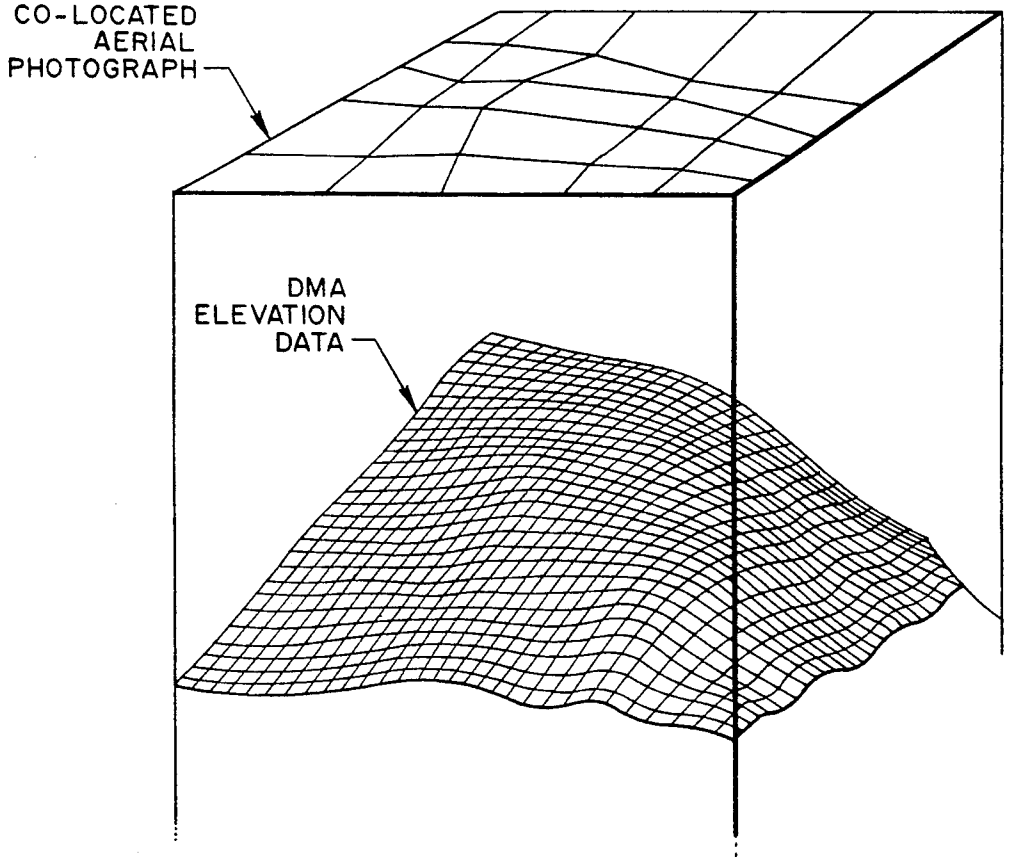

CO-LOCATED
AERIAL
PHOTOGRAPH
DMA
ELEVATION
DATA

Fig.-3

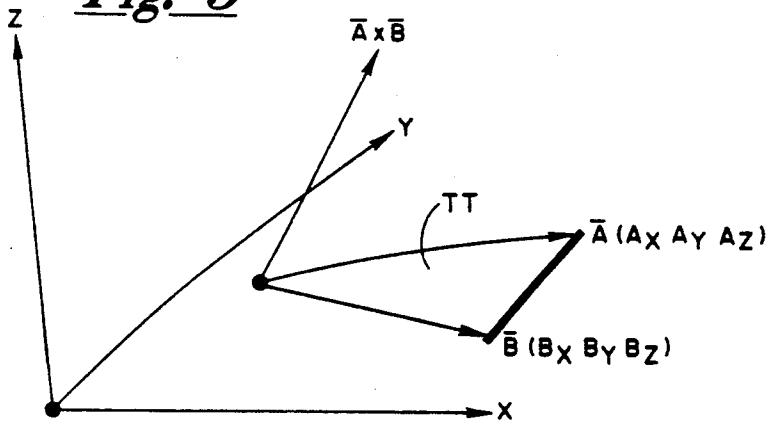
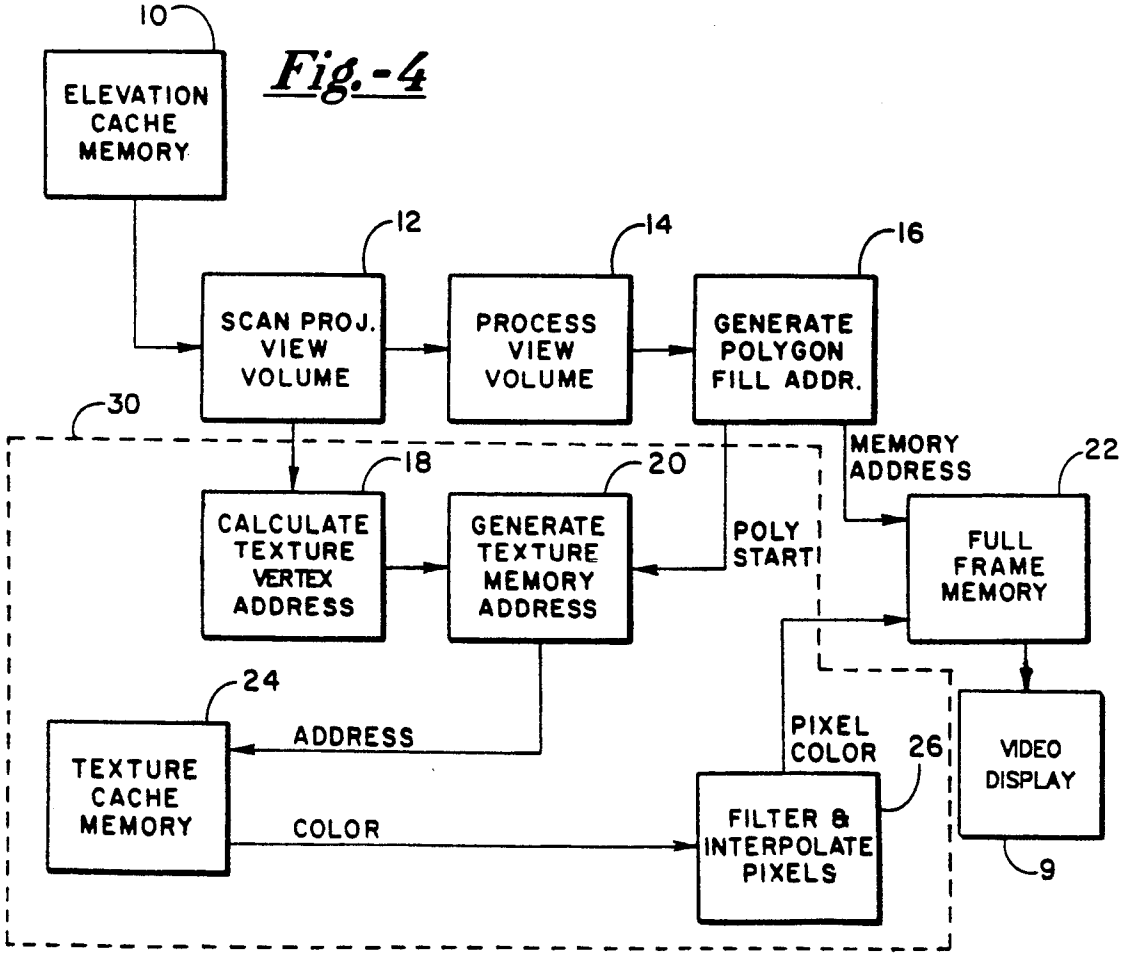

Z
$\bar{A} \times \bar{B}$
Y
TT
$\bar{A}$ (A_X A_Y A_Z)
$\bar{B}$ (B_X B_Y B_Z)
X

Fig.-4


10
ELEVATION CACHE MEMORY
12
SCAN PROJ. VIEW VOLUME
14
PROCESS VIEW VOLUME
16
GENERATE POLYGON FILL ADDR.
30
18
CALCULATE TEXTURE VERTEX ADDRESS
20
GENERATE TEXTURE MEMORY ADDRESS
POLY START
MEMORY ADDRESS
22
FULL FRAME MEMORY
24
TEXTURE CACHE MEMORY
ADDRESS
COLOR
PIXEL COLOR
26
FILTER & INTERPOLATE PIXELS
VIDEO DISPLAY
9

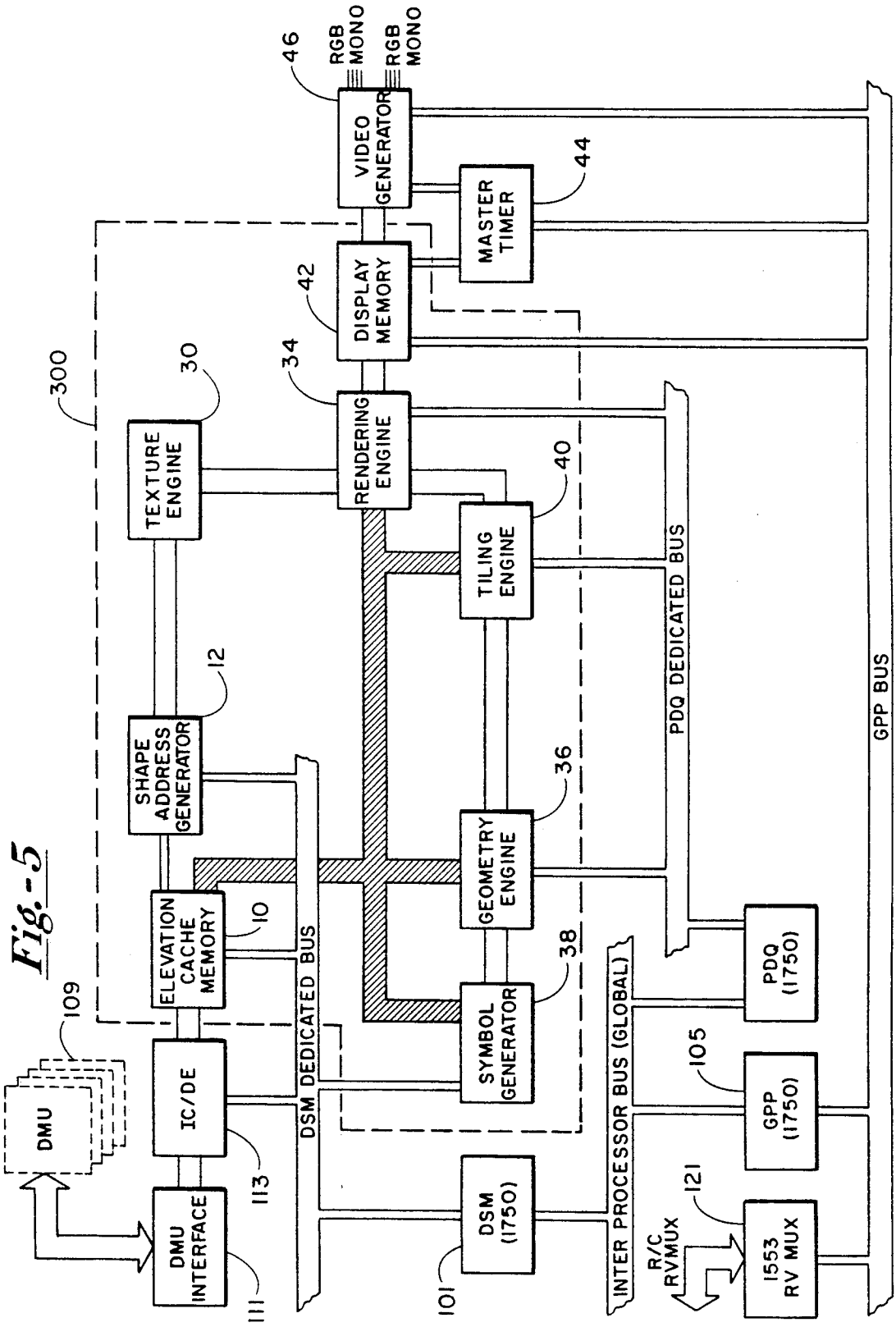

46
RGB MONO
RGB MONO
VIDEO GENERATOR
MASTER TIMER
44
DISPLAY MEMORY
42
34
RENDERING ENGINE
300
30
TEXTURE ENGINE
40
TILING ENGINE
12
SHAPE ADDRESS GENERATOR
36
GEOMETRY ENGINE
ELEVATION CACHE MEMORY
10
38
SYMBOL GENERATOR
109
DMU
IC/DE
113
DMU INTERFACE
111
DSM DEDICATED BUS
DSM (1750)
101
PDQ DEDICATED BUS
INTER PROCESSOR BUS (GLOBAL)
PDQ (1750)
105
GPP (1750)
121
R/C RVMUX
1553 RV MUX
GPP BUS

Fig.-5

Fig.-6

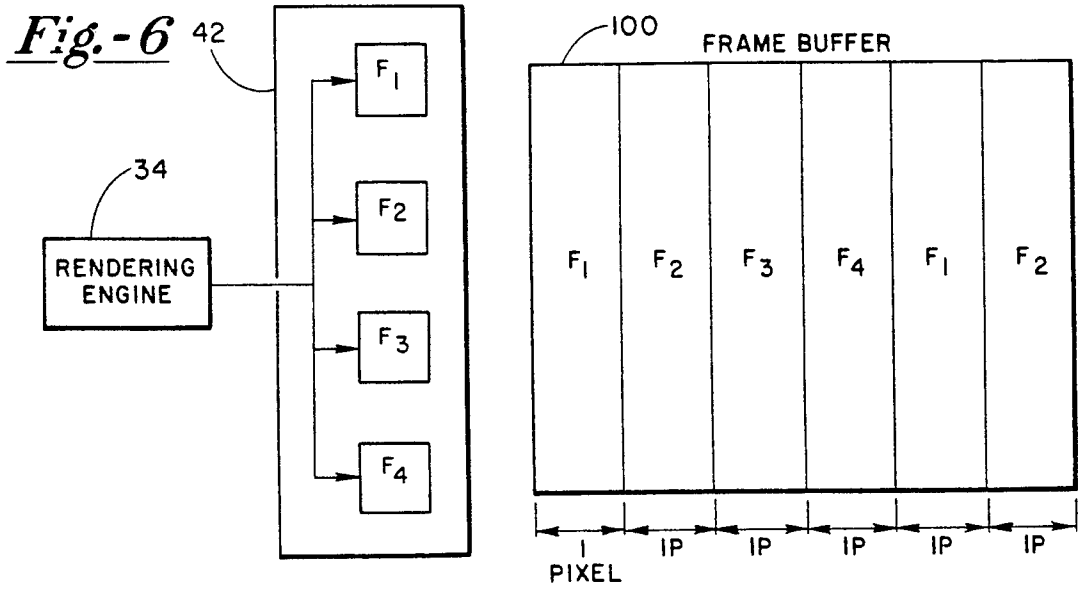


Fig.-7A

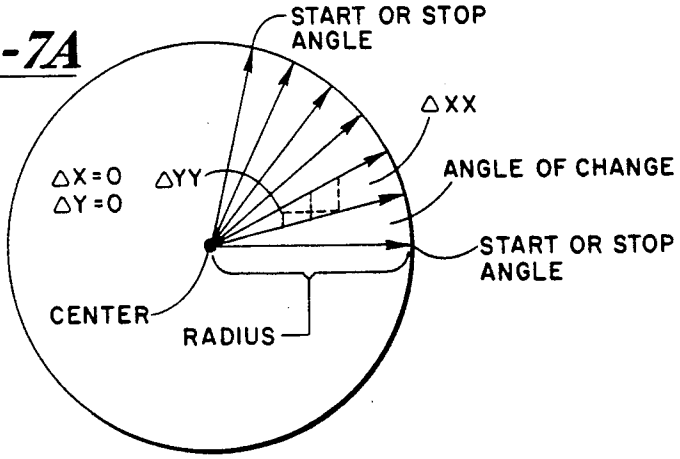


Fig.-7B

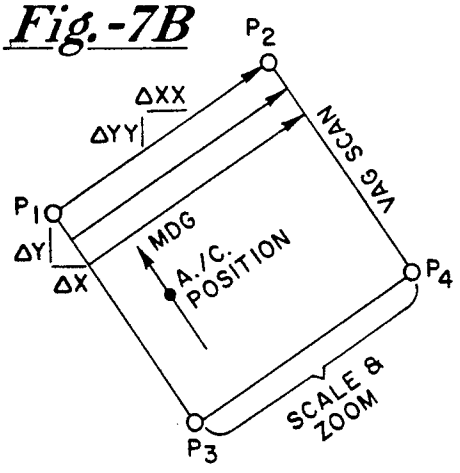


Fig.-7C

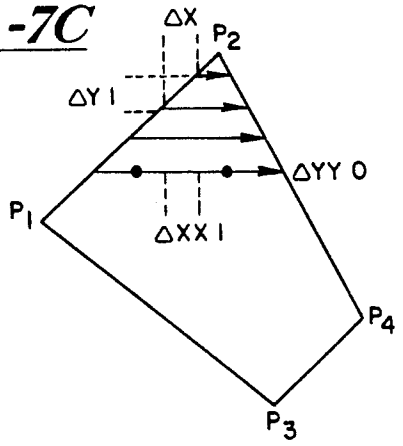
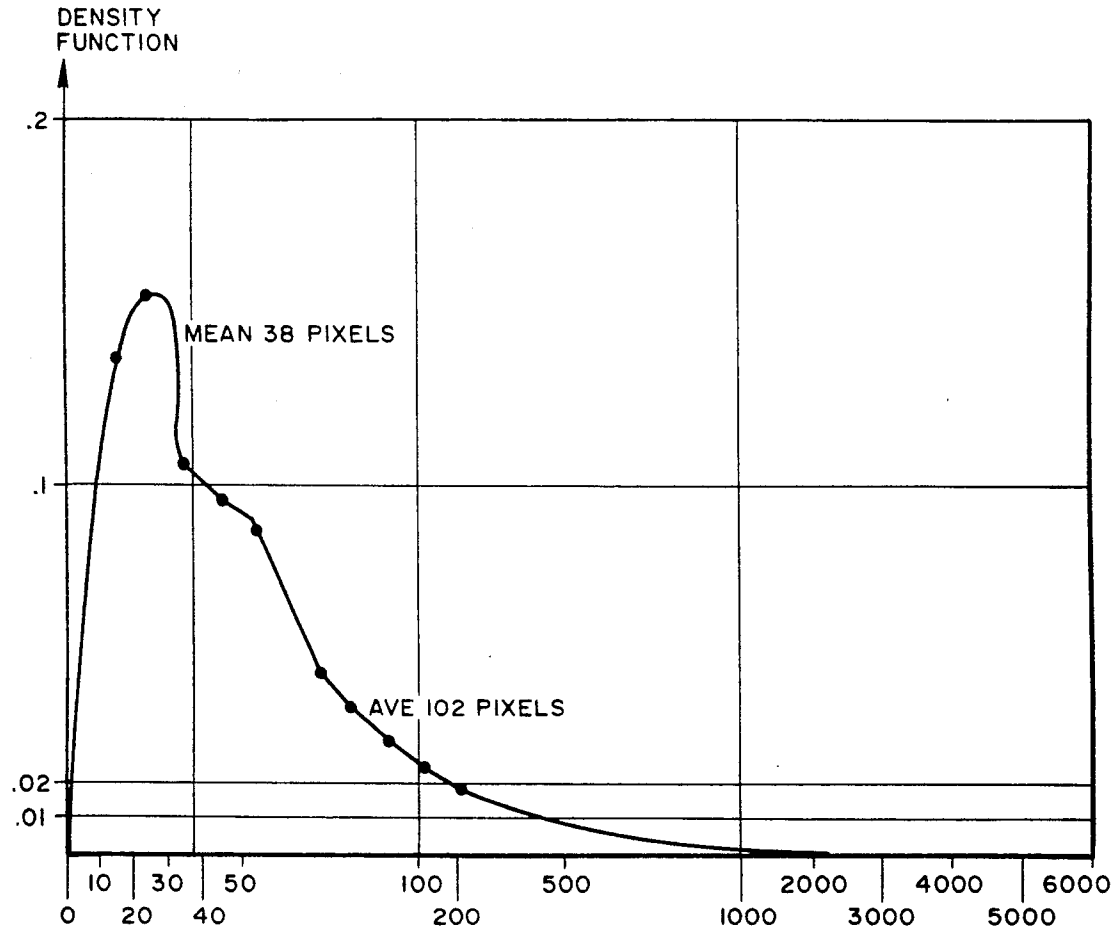


Fig.-8


DENSITY
FUNCTION
.2
.1
.02
.01
MEAN 38 PIXELS
AVE 102 PIXELS
0
10
20
30
40
50
100
200
500
1000
2000
3000
4000
5000
6000

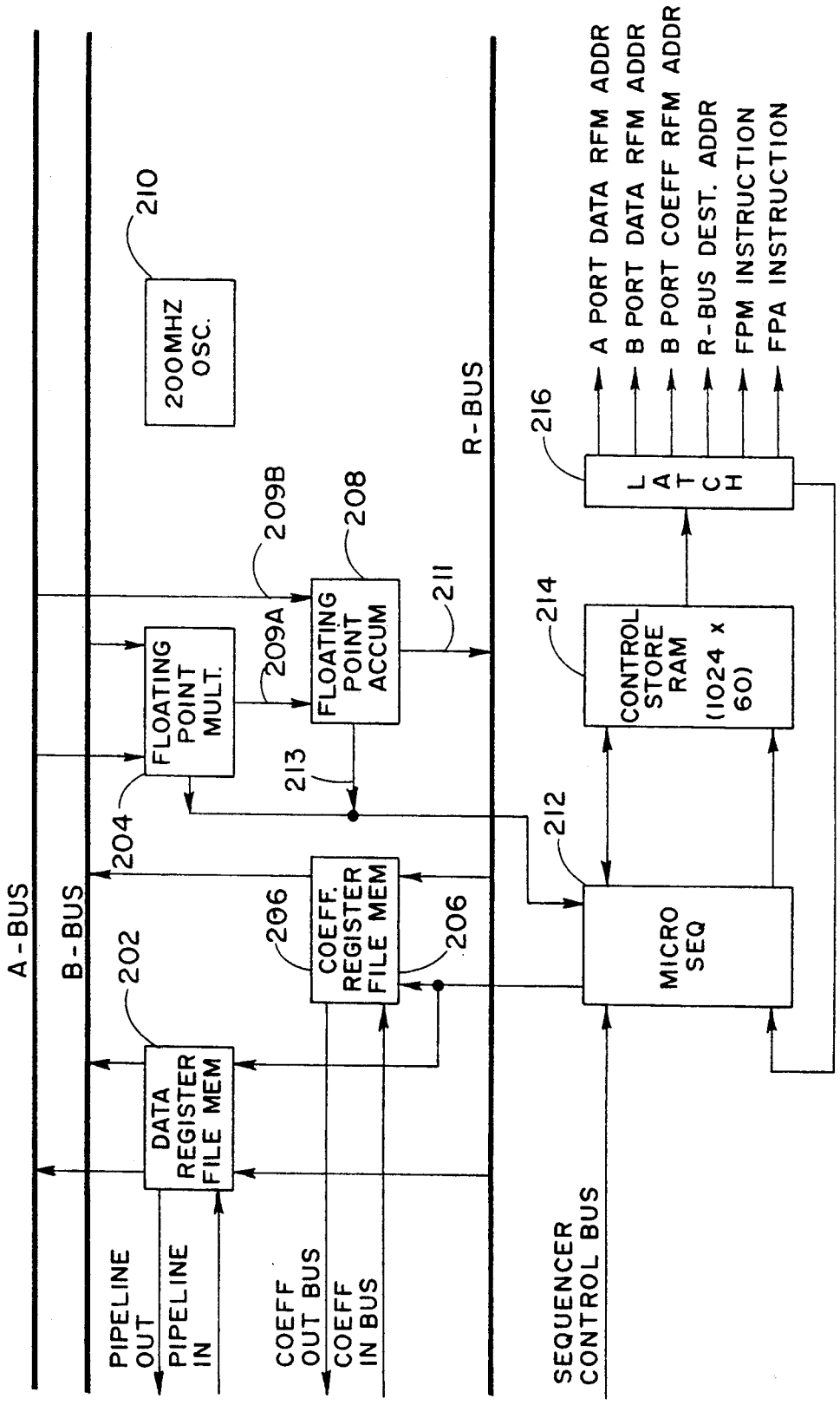

A-BUS
B-BUS
R-BUS
200MHZ OSC.
210
FLOATING POINT MULT.
209A
FLOATING POINT ACCUM
209B
208
211
213
204
DATA REGISTER FILE MEM
202
COEFF. REGISTER FILE MEM
206
206
MICRO SEQ
212
CONTROL STORE RAM (1024 x 60)
214
LATCH
216
A PORT DATA RFM ADDR
B PORT DATA RFM ADDR
B PORT COEFF RFM ADDR
R-BUS DEST. ADDR
FPM INSTRUCTION
FPA INSTRUCTION
PIPELINE OUT
PIPELINE IN
COEFF OUT BUS
COEFF IN BUS
SEQUENCER CONTROL BUS

Fig.-9

Fig.-10A

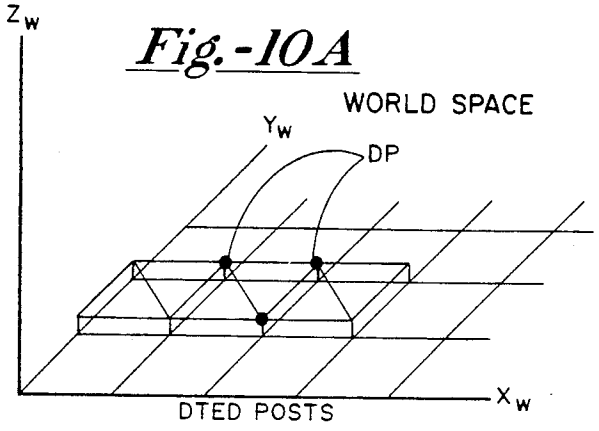


Fig.-10B

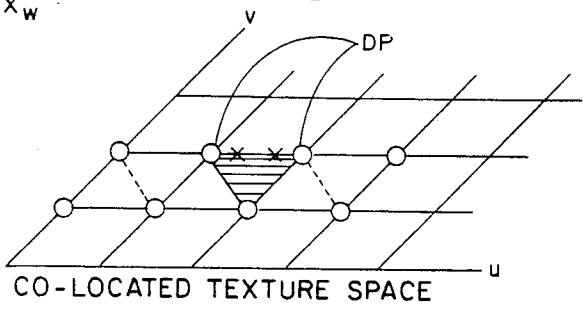


Fig.-10C

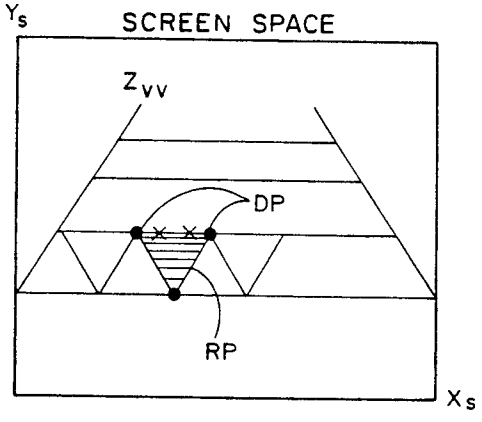
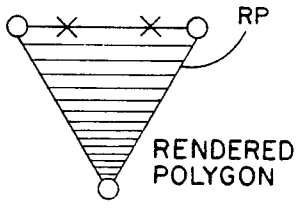


Fig.-10D



METHOD AND APPARATUS FOR GENERATING A TEXTURE MAPPED PERSPECTIVE VIEW

The present invention is directed generally to graphic display systems and, more particularly, to a method and apparatus for generating texture mapped perspective views for a digital map system.

RELATED APPLICATIONS

The following applications are included herein by reference:

(1) U.S. Pat. No. 4,876,651 filed May 11, 1988, issued Oct. 24, 1989 entitled "Digital Map System" which was assigned to the assignee of the present invention;

(2) Assignee copending application Ser. No. 09/514,685 filed Apr. 26, 1990, entitled "High Speed Processor for Digital Signal Processing";

(3) U.S. Pat. No. 4,884,220 entitled "Generator with Variable Scan Patterns" filed Jun. 7. 1988, issued Nov. 28, 1989, which is assigned to the assignee of the present invention;

(4) U.S. Pat. No. 4,899,293 entitled "A method of Storage and Retrieval of Digital Map Data Based Upon a Tessellated Geoid System", filed Dec. 14, 1988, issued Feb. 6, 1990;

(5) U.S. Pat. No. 5,020,014 entitled "Generic Interpolation Pipeline Processor", filed Feb. 7, 1989, issued May 28, 1991, which is assigned to the assignee of the present invention;

(6) Assignee's copending patent application Ser. No. 07/732,725 filed Jul. 18, 1991 entitled "Parallel Polygon/Pixel Rendering Engine Architecture for Computer Graphics" which is a continuation of patent application 07/419,722 filed Oct. 11, 1989 now abandoned;

(7) Assignee's copending patent application Ser. No. 07/514,724 filed Apr. 26, 1990 entitled "Polygon Tiling Engine";

(8) Assignee's copending patent application Ser. No. 07/514,723 filed Apr. 26, 1990 entitled "Polygon Sort Engine"; and

(9) Assignee's copending patent application Ser. No. 07/514,742 filed Apr. 26, 1990 entitled "Three Dimensional Computer Graphic Symbol Generator".

BACKGROUND OF THE INVENTION

Texture mapping is a computer graphics technique which comprises a process of overlaying aerial reconnaissance photographs onto computer generated three dimensional terrain images. It enhances the visual reality of raster scan images substantially while incurring a relatively small increase in computational expense. A frequent criticism of known computer-generated synthesized imagery has been directed to the extreme smoothness of the image. Prior art methods of generating images provide no texture, bumps, outcroppings, or natural abnormalities in the display of digital terrain elevation data (DTED).

In general, texture mapping maps a multidimensional image to a multidimensional space. A texture may be thought of in the usual sense such as sandpaper, a plowed field, a roadbed, a lake, woodgrain and so forth or as the pattern of pixels (picture elements) on a sheet of paper or photographic film. The pixels may be arranged in a regular pattern such as a checkerboard or may exhibit high frequencies as in a detailed photograph of high resolution LandSat imagery. Texture may also be three dimensional in nature as in marble or woodgrain surfaces. For the purposes of the invention, texture mapping is defined to be the mapping of a texture onto a surface in three dimensional object space. As is illustrated schematically in FIG. 1, a texture space object T is mapped to a display screen by means of a perspective transformation.

The implementation of the method of the invention comprises two processes. The first process is geometric warping and the second process is filtering. FIG. 2 illustrates graphically the geometric warping process of the invention for applying texture onto a surface. This process applies the texture onto an object to be mapped analogously to a rubber sheet being stretched over a surface. In a digital map system application, the texture typically comprises an aerial reconnaissance photograph and the object mapped is the surface of the digital terrain data base as shown in FIG. 2. After the geometric warping has been completed, the second process of filtering is performed. In the second process, the image is resampled on the screen grid.

The invention provides a texture mapped perspective view architecture which addresses the need for increased aircraft crew effectiveness, consequently reducing workload, in low altitude flight regimes characterized by the simultaneous requirement to avoid certain terrain and threats. The particular emphasis of the invention is to increase crew situational awareness. Crew situational awareness has been increased to some degree through the addition of a perspective view map display to a plan view capability which already exists in digital map systems. See, for example, assignee's copending application Ser. No. 07/192,798, for a DIGITAL MAP SYSTEM, filed May 11, 1988, issued Oct. 24, 1989 as U.S. Pat. No. 4,876,651 which is incorporated herein by reference in its entirety. The present invention improves the digital map system capability by providing a means for overlaying aerial reconnaissance photographs over the computer generated three dimensional terrain image resulting in a one-to-one correspondence from the digital map image to the real world. In this way the invention provides visually realistic cues which augment the informational display of such a computer generated terrain image. Using these cues an aircraft crew can rapidly make a correlation between the display and the real world.

The architectural challenge presented by texture mapping is that of distributing the processing load to achieve high data throughput using parallel pipelines and then recombining the parallel pixel flow into a single memory module known as a frame buffer. The resulting contention for access to the frame buffer reduces the effective throughput of the pipelines in addition to requiring increased hardware and board space to implement the additional pipelines. The method and apparatus of the invention addresses this challenge by effectively combining the low contention attributes of a single high speed pipeline with the increased processing throughput of parallel pipelines.

SUMMARY OF THE INVENTION

A method and apparatus for providing a texture mapped perspective view for digital map systems is provided. The invention comprises means for storing elevation data, means for storing texture data, means for scanning a projected view volume from the elevation data storing means, means for processing the projected view volume, means for generating a plurality of planar polygons and means for rendering images. The process-

ing means further includes means for receiving the scanned projected view volume from the scanning means, transforming the scanned projected view volume from object space to screen space, and computing surface normals at each vertex of each polygon so as to modulate texture space pixel intensity. The generating means generates the plurality of planar polygons from the transformed vertices and supplies them to the rendering means which then shades each of the planar polygons.

A primary object of the invention is to provide a technology capable of accomplishing a fully integrated digital map display system in an aircraft cockpit.

In one alternate embodiment of the invention, the polygons are shaded by means of the rendering means assigning one color across the surface of each polygon.

In yet another alternate embodiment of the invention, the rendering means interpolates the intensities between the vertices of each polygon in a linear fashion as in Gouraud shading.

It is yet another object of the invention to provide a digital map system including capabilities for perspective view, transparency, texture mapping, hidden line removal, and secondary visual effects such as depth cues and artifact (i.e., anti-aliasing) control.

It is yet another object of the invention to provide the capability for displaying forward looking infrared (FLIR) data and radar return images overlaid onto a plan and perspective view digital map image by fusing images through combining or subtracting other sensor video signals with the digital map terrain display.

It is yet another object of the invention to provide a digital map system with an arbitrary warping capability of one data base onto another data base which is accommodated by the perspective view texture mapping capability of the invention.

Other objects, features and advantages of the invention will become apparent to those skilled in the art through the drawings, description of the preferred embodiment and claims herein. In the drawings, like numerals refer to like elements.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows the mapping of a textured object to a display screen by a perspective transformation.

FIG. **2** illustrates graphically the geometric warping process of the invention for applying texture onto a surface.

FIG. **3** illustrates the surface normal calculation as employed by the invention.

FIG. **4** presents a functional block diagram of one embodiment of the invention.

FIG. **5** illustrates a top level block diagram of one embodiment of the texture mapped perspective view architecture of the invention.

FIG. **6** schematically illustrates the frame buffer configuration as employed by one embodiment of the invention.

FIGS. *7a*, *7b* and *7c* illustrate three examples of display format shapes.

FIG. **8** graphs the density function for maximum pixel counts.

FIG. **9** is a block diagram of one embodiment of the geometry array processor as employed by the invention.

FIGS. **10A**, **10B**, **10C** and **10D** illustrated the tagged architectural texture mapping as provided by the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Generally, perspective transformation from texture space having coordinates U, V to screen space having coordinates X, Y requires an intermediate transformation from texture space to object space having coordinates X_0, Y_0, Z_0. Perspective transformation is accomplished through the general perspective transform equation as follows:

$$[X\ Y\ Z\ H] = [X\ Y\ Z\ 1]\ X \begin{bmatrix} A & B & C & P \\ D & E & F & Q \\ G & H & I & R \\ L & M & N & S \end{bmatrix}$$

where a point (X,Y,Z) in 3-space is represented by a four dimensional position vector [X Y Z H] in homogeneous coordinates.

The 3×3 sub-matrix

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}$$

accomplishes scaling, shearing, and rotation.

The 1×3 row matrix [L M N] produces translation.

The 3×1 column matrix

$$\begin{bmatrix} P \\ Q \\ R \end{bmatrix}$$

produces perspective transformation.

The 1×1 scalar [S] produces overall scaling.

The Cartesian cross-product needed for surface normal requires a square root. As shown in FIG. **3**, the surface normal shown is a vector $A\times B$ perpendicular to the plane formed by edges of a polygon as represented by vectors A and B, where $A\times B$ is the Cartesian cross-product of the two vectors. Normalizing the vector allows calculation for sun angle shading in a perfectly diffusing Lambertian surface. This is accomplished by taking the vector dot product of the surface normal vector with the sun position vector. The resulting angle is inversely proportional to the intensity of the pixel of the surface regardless of the viewing angle. This intensity is used to modulate the texture hue and intensity value.

$$\frac{A\times B}{||A||\,||B||} \text{ where } \begin{matrix} A = Ax^2 + Ay^2 + Az^2 \\ B = Bx^2 + By^2 + Bz^2 \end{matrix}$$

A terrain triangle TT is formed by connecting the endpoints of vectors A and B, from point B_X, B_Y, B_Z to point A_X, A_Y, A_Z.

Having described some of the fundamental basis for the invention, a description of the method of the invention will now be set out in more detail below.

Referring now to FIG. **4**, a functional block diagram of one embodiment of the invention is shown. The invention functionally comprises a means for storing ele-

vation data **10**, a means for storing texture data **24**, a means for scanning a projected view volume from the elevation data storing means **12**, means for processing view volume **14** including means for receiving the scanned projected view volume from the scanning means **12**, means for generating polygon fill addresses **16**, means for calculating texture vertices addresses **18**, means for generating texture memory addresses **20**, means for filtering and interpolating pixels **26**, a full-frame memory **22**, and video display **9**. The processing means **14** further includes means for transforming the scanned projected view volume from object space to screen space and means for computing surface normals at each vertex of each polygon so as to calculate pixel intensity.

The means for storing elevation data **10** may preferably be a cache memory having at least a 50 nsec access time to achieve 20 Hz bi-linear interpolation of a 512×512 pixel resolution screen. The cache memory further may advantageously include a 256×256 bit buffer segment with 2K bytes of shadow RAM used for the display list. The cache memory may arbitrarily be reconfigured from 8 bits deep (data frame) to 64 bits (i.e., comprising the sum of texture map data (24 bits)+DTED (16 bits)+aeronautical chart data (24 bits)). A buffer segment may start at any cache address and may be written horizontally or vertically. Means for storing texture data **24** may advantageously be a texture cache memory which is identical to the elevation cache memory except that it stores pixel information for warping onto the elevation data cache. Referring now to FIG. **5**, a top level block diagram of the texture mapped perspective view architecture is shown. The architecture implements the functions as shown in FIG. **4** and the discussion which follows shall refer to functional blocks in FIG. **4** and corresponding elements in FIG. **5**. In some cases, such as element **14**, there is a one-to-one correspondence between the functional blocks in FIG. **4** and the architectural elements of FIG. **5**. In other cases, as explained hereinbelow, the functions depicted in FIG. **4** are carried out by a plurality of elements shown in FIG. **5**. The elements shown in FIG. **5** comprising the texture mapped perspective view system **300** of the invention include elevation cache memory **10**, shape address generator (SHAG) **12**, texture engine **30**, rendering engine **34**, geometry engine **36**, symbol generator **38**, tiling engine **40**, and display memory **42**. These elements are typically part of a larger digital map system including a digital map unit (DMU) **109**, DMU interface **111**, IC/DE **113**, a display stream manager (DSM) **101**, a general purpose processor (GPP) **105**, RV MUX **121**, PDQ **123**, master time **44**, video generator **46** and a plurality of data bases. The latter elements are described in assignee's Digital Map System U.S. Pat. No. 4,876,651.

GEOMETRY ENGINE

The geometry engine **36** is comprised of one or more geometry array processors (GAPs) which process the 4×4 Euler matrix transformation from object space (sometimes referred to as "world" space) to screen space. The GAPs generate X and Y values in screen coordinates and Zvv values in range depth. The GAPs also compute surface normals at each vertex of a polygon representing an image in object space via Cartesian cross-products for Gouraud shading, or they may assign one surface normal to the entire polygon for flat shading and wire mesh. Intensity calculations are performed using a vector dot product between the surface normal or normals and the illumination source to implement a Lambertian diffusely reflecting surface. Hue and intensity values are then assigned to the polygon. The method and apparatus of the invention also provides a dot rendering scheme wherein the GAPs only transform one vertex of each polygon and the tiling engine **40**, explained in more detail below, is inhibited. In this dot rendering format, hue and intensity are assigned based on the planar polygon containing the vertex and the rendering engine is inhibited. Dot polygons may appear in the same image as multiple vertex polygons or may comprise the entire image itself. The "dots" are passed through the polygon rendering engine **34**. A range to the vertices or polygon (Zvv) is used if a fog or "DaVinci" effect are invoked as explained below. The GAPs also transform three dimensional overlay symbols from world space to screen space.

Referring now to FIG. **9**, a block diagram of one example embodiment of a geometry array processor (GAP) is shown. The GAP comprises a data register file memory **202**, a floating point multiplier **204**, a coefficient register file memory **206**, a floating point accumulator **208**, a 200 MHz oscillator **210**, a microsequencer **212**, a control store RAM **214**, and latch **216**.

The register file memory may advantageously have a capacity of 512 by 32 bits. The floating point accumulator **208** includes two input ports **209A** and **209B** with independent enables, one output port **211**, and a condition code interface **212** responsive to error codes. The floating point accumulator operates on four instructions, namely, multiply, no-op, pass A, and pass B. The microsequencer **212** operates on seven instructions including loop on count, loop on condition, jump, continue, call, return and load counter. The microsequencer includes a debug interface having a read/write (R/W) internal register, R/W control store memory, halt on address, and single step, and further includes a processor interface including a signal interrupt, status register and control register. The GAP is fully explained in the assignee's co-pending application No. 07/514,685 filed Apr. 26, 1990 entitled High Speed Processor for Digital Signal Processing which is incorporated herein by reference in its entirety.

In one alternative embodiment of the invention, it is possible to give the viewer of the display the visual effect of an environment enshrouded in fog. The fog option is implemented by interpolating the color of the triangle vertices toward the fog color. As the triangles get smaller with distance, the fog particles become denser. By using the known relationship between distance and fog density, the fog thickness can be "dialed" or adjusted as needed. The vertex assignment interpolates the vertex color toward the fog color as a function of range toward the horizon. The fog technique may be implemented in the hardware version of the GAP such as may be embodied in a GaAs semiconductor chip. If a linear color space (typically referred to as "RGB" to reflect the primary colors, red, green and blue) is assumed, the amount of fog is added as a function of range to the polygon vertices' color computation by well known techniques. Thus, as the hue is assigned by elevation banding or monochrome default value, the fog color is tacked on. The rendering engine **34**, explained in more detail below, then straight forwardly interpolates the interior points.

In another alternative embodiment of the invention, a DaVinci effect is implemented. The DaVinci effect

causes the terrain to fade into the distance and blend with the horizon. It is implemented as a function of range of the polygon vertices by the GAP. The horizon color is added to the vertices similarly to the fog effect.

SHAPE ADDRESS GENERATOR (SHAG)

The SHAG **12** receives the orthographically projected view volume outline onto cache from the DSM. It calculates the individual line lengths of the scans and the delta x and delta y components. It also scans the elevation posts out of the elevation cache memory and passes them to the GAPs for transformation. In one embodiment of the invention, the SHAG preferably includes two arithmetic logic units (ALUs) to support the **50** nsec cache **10**. In the SHAG, data is generated for the GAPs and control signals are passed to the tiling engine **40**. DFAD data is downloaded into overlay RAM (not shown) and three dimensional symbols are passed to the GAPs from symbol generator **38**. Elevation color banding hue assignment is performed in this function. The SHAG generates shapes for plan view, perspective view, intervisibility, and radar simulation. These are illustrated in FIG. **7**. The SHAG is more fully explained in assignee's copending application, Ser. No. 203,660, Generator With Variable Scan Patterns, filed Jun. 7, 1988 issued as U.S. Pat. No. 4,884,220 on Nov. 28, 1989 which is incorporated herein by reference in its entirety.

A simple Lambertian lighting diffusion model has proved adequate for generating depth cueing in one embodiment of the invention. The sun angle position is completely programmable in azimuth and zenith. It may also be self-positioning based on time of day, time of year, latitude and longitude. A programmable intensity with gray scale instead of color implements the moon angle position algorithm. The display stream manager (DSM) programs the sun angle registers. The illumination intensities of the moon angle position may be varied with the lunar waxing and waning cycles.

TILING ENGINE AND TEXTURE ENGINE

Still referring to FIGS. **4** and **5**, the means for calculating texture vertex address **18** may include the tiling engine **40**. Elevation posts are vertices of planar triangles modeling the surface of the terrain. These posts are "tagged" with the corresponding U,V coordinate address calculated in texture space. This tagging eliminates the need for interpolation by substituting an address lookup. Referring to FIGS. **10A**, **10B**, **10C** and **10D**, with continuing reference to FIGS. **4** and **5**, the tagged architectural texture mapping as employed by the invention is illustrated. FIG. **10A** shows an example of DTED data posts, DP, in world space. FIG. **10B** shows the co-located texture space for the data posts. FIG. **10C** shows the data posts and rendered polygon in screen space. FIG. **10D** illustrates conceptually the interpolation of tagged addresses into a rendered polygon RP. The texture engine **30** performs the tagged data structure management and filtering processes. When the triangles are passed to the rendering engine by the tiling engine for filling with texture, the tagged texture address from the elevation post is used to generate the texture memory address. The texture value is filtered by filtering and interpolation means **26** before being written to full-frame memory **22** prior to display.

The tiling engine generates the planar polygons from the transformed vertices in screen coordinates and passes them to the rendering engine. For terrain polygons, a connectivity offset from one line scan to the next is used to configure the polygons. For overlay symbols, a connectivity list is resident in a buffer memory (not shown) and is utilized for polygon generation. The tiling engine also informs the GAP if it is busy. In one embodiment 512 vertices are resident in a 1K buffer.

All polygons having surface normals more than 90 degrees from LOS are eliminated from rendering. This is known in the art as backface removal. Such polygons do not have to be transformed since they will not be visible on the display screen. Additional connectivity information must be generated if the polygons are non-planar as the transformation process generates implied edges. This requires that the connectivity information be dynamically generated. Thus, only planar polygons with less than 513 vertices are implemented. Non-planar polygons and dynamic connectivity algorithms are not implemented by the tiling engine. The tiling engine is further detailed in assignee's copending applications of even filing date herewith entitled Polygon Tiling Engine, as referenced hereinabove and Polygon Sort Engine, as referenced hereinabove, both of which are incorporated herein by reference.

RENDERING ENGINE

Referring again to FIG. **5**, the rendering engine **34** of the invention provides a means of drawing polygons in a plurality of modes. The rendering engine features may include interpolation algorithms for processing coordinates and color, hidden surface removal, contour lines, aircraft relative color bands, flat shading, Gourand shading, phong shading, mesh format or screen door effects, ridgeline display, transverse slice, backface removal and RECE (aerial reconnaissance) photo modes. With most known methods of image synthesis, the image is generated by breaking the surfaces of the object into polygons, calculating the color and intensity at each vertex of the polygon, and drawing the results into a frame buffer while interpolating the colors across the polygon. The color information at the vertices is calculated from light source data, surface normal, elevation and/or cultural features.

The interpolation of coordinate and color (or intensity) across each polygon must be performed quickly and accurately. This is accomplished by interpolating the coordinate and color at each quantized point or pixel on the edges of the polygon and subsequently interpolating from edge to edge to generate the fill lines. For hidden surface removal, such as is provided by a Z-buffer in a well-known manner, the depth or Z-value for each pixel is also calculated. Furthermore, since color components can vary independently across a surface or set of surfaces, red, green and blue intensities are interpolated independently. Thus, a minimum of six different parameters (X,Y,Z,R,G,B) are independently calculated when rendering polygons with Gouraud shading and interpolated Z-values.

Additional features of the rendering engine include a means of providing contour lines and aircraft relative color bands. For these features the elevation also is interpolated at each pixel. Transparency features dictate that an alpha channel be maintained and similarly interpolated. These requirements imply two additional axes of interpolation bringing the total to eight. The rendering engine is capable of processing polygons of one vertex in its dot mode, two vertices in its line mode, and three to 512 coplanar vertices in its polygon mode.

In the flat shading mode the rendering engine assigns the polygon a single color across its entire surface. An arbitrary vertex is selected to assign both hue and intensity for the entire polygon. This is accomplished by assigning identical RGB values to all vertices. Interpolation is performed normally but results in a constant value. This approach will not speed up the rendering process but will perform the algorithm with no hardware impact.

The Gouraud shading algorithm included in the rendering engine interpolates the intensities between the vertices of each polygon rendered in a linear fashion. This is the default mode. The Phong shading algorithm interpolates the surface normals between the vertices of the polygon between applying the intensity calculations. The rendering engine would thus have to perform an illumination calculation at each pixel after interpolation. This approach would significantly impact the hardware design. This algorithm may be simulated, however, using a weighing function (typically a function of cosine (Θ)) around a narrow band of the intensities. This results in a non-linear interpolation scheme and provides for a simulated specular reflectance. In an alternative embodiment, the GAP may be used to assign the vertices of the polygon this non-linear weighing via the look-up table and the rendering engine would interpolate as in Gouraud shading.

Transparency is implemented in the classical sense using an alpha channel or may be simulated with a screen door effect. The screen door effect simply renders the transparent polygon as normal but then only outputs every other or every third pixel. The mesh format appears as a wire frame overlay with the option of rendering either hidden lines removed or not. In the case of a threat dome symbol, all polygon edges must be displayed as well as the background terrain. In such a case, the fill algorithm of the rendering engine is inhibited and only the polygon edges are rendered. The intensity interpolation is performed on the edges which may have to be two pixels wide to eliminate strobing. In one embodiment, an option for terrain mesh includes the capability for tagging edges for rendering so that the mesh appears as a regular orthogonal grid.

Typical of the heads up display (HUD) format used in aircraft is the ridgeline display and the transverse slice. In the ridgeline format, a line drawing is produced from polygon edges whose slopes change sign relative to the viewpoint. All polygons are transformed, tiled, and then the surface normals are computed and compared to the viewpoint. The tiling engine strips away the vertices of non-ridge contributing edges and passes only the ridge polygons to the rendering engine. In transverse slice mode, fixed range bins relative to the aircraft are defined. A plane orthogonal to the view LOS is then passed through for rendering. The ridges then appear to roll over the terrain as the aircraft flies along. These algorithms are similar to backface removal. They rely upon the polygon surface normal being passed to the tiling engine.

One current implementation of the invention guarantees non-intersecting polygon sides by restricting the polygons rendered to be planar. They may have up to 512 vertices. Polygons may also consist of one or two vertices. The polygon "end" bit is set at the last vertex and processed by the rendering engine. The polygon is tagged with a two bit rendering code to select mesh, transparent, or Gouraud shading. The rendering engine also accomplishes a fine clip to the screen for the polygon and implements a smoothing function for lines.

An optional aerial reconnaissance (RECE) photo mode causes the GAP to texture map an aerial reconnaissance photograph onto the DTED data base. In this mode the hue interpolation of the rendering engine is inhibited as each pixel of the warping is assigned a color from the RECE photo. The intensity component of the color is dithered in a well known manner as a function of the surface normal as well as the Z-depth. These pixels are then processed by the rendering engine for Z-buffer rectification so that other overlays such as threats may be accommodated. The RECE photos used in this mode have been previously warped onto a tessellated geoid data base and thus correspond pixel-for-pixel to the DTED data. See assignee's aforereferenced copending application for A Method of Storage and Retrieval of Digital Map Data Based Upon A Tessellated Geoid System, which is hereby incorporated by reference in its entirety. The photos may be denser than the terrain data. This implies a deeper cache memory to hold the RECE photos. Aeronautical chart warping mode is identical to RECE photos except that aeronautical charts are used in the second cache. DTED warping mode utilizes DTED data to elevation color band aeronautical charts.

The polygon rendering engine may preferably be implemented in a generic interpolation pipeline processor (GIPP) of the type as disclosed in assignee's aforereferenced patent entitled Generic Interpolation Pipeline Processor, which is incorporated herein by reference in its entirety. In one embodiment of the invention, the GIPPs fill in the transformed polygons using a bi-linear interpolation scheme with six axes (X,Y,Z,R,G,B). The primitive will interpolate a 16 bit pair and 8 bit pair of values simultaneously, thus requiring 3 chips for a polygon edge. One embodiment of the system of the invention has been sized to process one million pixels each frame time. This is sufficient to produce a 1K×1K high resolution chart, or a 512×512 DTED frame with an average of four overwrites per pixel during hidden surface removal with GIPPs outputting data at a 60 nsec rate, each FIFO, F1-F4, as shown in FIG. **6**, will receive data on the average of every 240 nsec. An even distribution can be assumed by decoding on the lower 2X address bits. Thus, the memory is divided into one pixel wide columns FIG. **6** is discussed in more detail below.

Referring again to FIGS. **4** and **5**, the "dots" are passed through the GIPPs without further processing. Thus, the end of each polygon's bit is set. A ZB buffer is needed to change the color of a dot at a given pixel for hidden dot removal. Perspective depth cuing is obtained as the dots get closer together as the range from the viewpoint increases.

Bi-linear interpolation mode operates in plan view on either DLMS or aeronautical charts. It achieves 20 Hz interpolation on a 512×512 display. The GIPPs perform the interpolation function.

DATA BASES

A Level I DTED data base is included in one embodiment of the invention and is advantageously sampled on three arc second intervals. Buffer segments are preferably stored at the highest scales (104.24 nm) and the densest data (13.03 nm). With such a scheme, all other scales can be created. A Level II DTED data base is also included and is sampled at one arc second inter-

vals. Buffer segments are preferably stored only at the densest data (5.21 nm).

A DFAD cultural feature data base is stored in a display list of 2K words for each buffer segment. The data structure consists of an icon font call, a location in cache, and transformation coefficients from model space to world space consisting of scaling, rotation, and position (translation). A second data structure comprises a list of polygon vertices in world coordinates and a color or texture. The DFAD data may also be rasterized and overlaid on a terrain similar to aerial reconnaissance photos.

Aeronautical charts at the various scales are warped into the tessellated geoid. This data is 24 bits deep. Pixel data such as LandSat, FLIR, data frames and other scanned in source data may range from one bit up to 24 bits in powers of two (1,2,4,8,16,24).

FRAME BUFFER CONFIGURATION

Referring again to FIG. **6**, the frame buffer configuration of one embodiment of the invention is shown schematically. The frame buffer configuration is implemented by one embodiment of the invention comprises a polygon rendering chip **34** which supplies data to full-frame memory **42**. The full-frame memory **42** advantageously includes first-in, first-out buffers (FIFO) F_1, F_2, F_3 and F_4. As indicated above with respect to the discussion of the rendering engine, the memory is divided up into one pixel wide columns as shown in FIG. **6**. By doing so, however, chip select must changed on every pixel when the master timer **44** shown in FIG. **5** reads the memory. However, by orienting the SHAG scan lines at 90 degrees to the master timer scan lines, the chip select will change on every line. The SHAG starts scanning at the bottom left corner of the display and proceeds to the upper left corner of the display.

With the image broken up in this way, the probability that the GIPP will write to the same FIFO two times in a row, three times, four, and so on can be calculated to determine how deep the FIFO must be. Decoding on the lower order address bits means that the only time the rendering engine will write to the same FIFO twice in a row is when a new scan line is started. At four deep as shown in the frame buffer graph **100**, the chances of the FIFO filling up are approximately one in 6.4K. With an image of 1 million pixels, this will occur an acceptably small number of times for most applications. The perspective view transformations for 10,000 polygons with the power and board area constraints that are imposed by an avionics environment is significant. The data throughput for a given scene complexity can be achieved by adding more pipeline in parallel to the architecture. It is desirable to have as few pipelines as possible, preferably one, so that the image reconstruction at the end of the pipeline does not suffer from an arbitration bottleneck for a Z-buffered display memory.

In one embodiment of the invention, the processing throughput required has been achieved through the use of GaAs VSLI technology for parallel pipelines and a parallel frame buffer design has eliminated contention bottlenecks. A modular architecture allows for additional functions to be added to further the integration of the digital map into the avionics suite. The system architecture of the invention has high flexibility while maintaining speed and data throughput. The polygonal data base structure approach accommodate arbitrary scene complexity and a diversity of data base types.

The data structure of the invention is tagged so that any polygon may be rendered via any of the implemented schemes in a single frame. Thus, a particular image may have Gouraud shaded terrain, transparent threat domes, flat shaded cultural features, lines, and dots. In addition, since each polygon is tagged, a single icon can be comprised of differently shaded polygons. The invention embodies a 24 bit color system, although a production map would be scaled to 12 bits. A 12 bit system provides 4K colors and would require a 32K by 8 RGB RAM look-up table (LUT).

MISCELLANEOUS FEATURES

The display formats in one example of the invention are switchable at less than 600 milliseconds between paper chart, DLMS plan and perspective view. A large cache (1 megabit D-RAMs) is required for texture mapping. Other format displays warp chart data over DTED, or use DTED to pseudo-color the map. For example, change the color palate LUT for transparency. The GAP is used for creating a true orthographic projection of the chart data.

An edit mode for three dimensions is supported by the apparatus of the invention. A three dimensional object such as a "pathway in the sky" may be tagged for editing. This is accomplished by first, moving in two dimensions at a given AGL, secondly, updating the AGL in the three dimensional view, and finally, updating the data base.

The overlay memory from the DMC may be video mixed with the perspective view display memory.

Freeze frame capability is supported by the invention. In this mode, the aircraft position is updated using the cursor. If the aircraft flies off the screen, the display will snap back in at the appropriate place. This capability is implemented in plan view only. There is data frame software included to enable roaming through cache memory. This feature requires a two axis roam joystick or similar control. Resolution of the Z-buffer is 16 bits. This allows 64K meters down range.

The computer generated imagery has an update rate of 20 Hz. The major cycle is programmable and variable with no frame extend invoked. The system will run as fast as it can but will not switch ping-pong display memories until each functional unit issues a "pipeline empty" message to the display memory. The major cycle may also be locked to a fixed frame in multiples of 16.6 milliseconds. In the variable frame mode, the processor clock is used for a smooth frame interpolation for roam or zoom. The frame extend of the DMC is eliminated in perspective view mode. Plan view is implemented in the same pipeline as the perspective view. The GPP **105** loads the countdown register on the master timer to control the update rate.

The slowest update rate is 8.57 Hz. The image must be generated in this time or the memories will switch. This implies a pipeline speed of 40 million pixels per second. In a 512×512 image, it is estimated that there would be 4 million pixels rendered worst case with heavy hidden surface removal. In most cases, only million pixels need be rendered. FIG. **8** illustrates the analysis of pixel over-writes. The minimum requirement for surface normal resolution so that the best image is achieved is 16 bits. Tied to this is the way in which the normal is calculated. Averaging from surrounding tiles gives a smoother image on scale change or zoom. Using one tile is less complex, but results in poorer image

quality. Surface normal is calculated on the fly in accordance with known techniques.

DISPLAY MEMORY

This memory is a combination of scene and overlay with a Z-buffer. It is distributed or partitioned for optimal loading during write, and configured as a frame buffer during read-out. The master time speed required is approximately 50 MHz. The display memory resolution can be configured as $512 \times 512 \times 12$ or as $1024 \times 1024 \times 12$. The Z-buffer is 16 bits deep and $1K \times 1K$ resolution. At the start of each major cycle, the Z-values are set to plus infinity (FF Hex). Infinity (Zmax) is programmable. The back clipping plane is set by the DSM over the control bus.

At the start of each major cycle, the display memory is set to a background color. In certain modes such as mesh or dot, this color will change. A background color register is loaded by the DSM over the configuration bus and used to fill in the memory.

VIDEO GENERATOR/MASTER TIMER

The video generator **46** performs the digital to analog conversion of the image data in the display memory to send to the display head. It combines the data stream from the overlay memory of the DMC with the display memory from the perspective view. The configuration bus loads the color map.

A 30 Hz interlaced refresh rate may be implemented in a system employing the present invention. Color pallets are loadable by the GPP. The invention assumes a linear color space in RGB. All colors at zero intensity go to black.

THREE DIMENSIONAL SYMBOL GENERATOR

The three-dimensional symbol generator **38** performs the following tasks:

1. It places the model to world transformation coefficients in the GAP.

2. It operates in cooperation with the geometry engine to multiply the world to screen transformation matrix by the model to world transformation matrix to form a model to screen transformation matrix. This matrix is stored over the model to world transformation matrix.

3. It operates in cooperation with the model to screen transformation matrix to each point of the symbol from the vertex list to transform the generic icon to the particular symbol.

4. It processes the connectivity list in the tiling engine and forms the screen polygons and passes them to the rendering engine.

One example of a three-dimensional symbol generator is described in detail in the assignee's aforereferenced patent application entitled "Three Dimensional Computer Graphic Symbol Generator".

The symbol generator data base consists of vertex list library and 64K bytes of overlay RAM and a connectivity list. Up to 18K bytes of DFAD (i.e., 2K bytes display list from cache shadow RAM $\times 9$ buffer segments) are loaded into the overlay RAM for cultural feature processing. The rest of the memory holds the threat/intelligence file and the mission planning file for the entire gaming area. The overlay RAM is loaded over the control bus from the DSM processor with the threat and mission planning files. The SHAG loads the DFAD files. The symbol libraries are updated via the configuration bus.

The vertex list contains the relative vertex positions of the generic library icons. In addition, it contains a 16 bit surface normal, a one bit end of polygon flag, and a one bit end of symbol flag. The table is $32K \times 16$ bits. A maximum of 512 vertices may be associated with any given icon. The connectivity list contains the connectivity information of the vertices of the symbol. A 64K by 12 bit table holds this information.

A pathway in the sky format may be implemented in this system. It consists of either a wire frame tunnel or an elevated roadbed for flight path purposes. The wire frame tunnel is a series of connected transparent rectangles generated by the tiling engine of which only the edges are visible (wire mesh). Alternatively, the polygons may be precomputed in world coordinates and stored in a mission planning file. The roadbed is similarly comprised of polygons generated by the tiler along a designated pathway. In either case, the geometry engine must transform these polygons from object space (world coordinate system) to screen space. The transformed vertices are then passed to the rendering engine. The parameters (height, width, frequency) of the tunnel and roadbed polygons are programmable.

Another symbol used in the system is a waypoint flag. Waypoint flags are markers consisting of a transparent or opaque triangle on a vertical staff rendered in perspective. The waypoint flag icon is generated by the symbol generator as a macro from a mission planning file. Alternatively, they may be precomputed as polygons and stored. The geometry engine receives the vertices from the symbol generator and performs the perspective transformation on them. The geometry engine passes the rendering engine the polygons of the flag staff and the scaled font call of the alphanumeric symbol. Plan view format consists of a circle with a number inside and is not passed through the geometry engine.

DFAD data processing consists of a generalized polygon renderer which maps 32K points possible down to 256 polygons or less for a given buffer segment. These polygons are then passed to the rendering engine. This approach may redundantly render terrain and DFAD for the same pixels but easily accommodates declutter of individual features. Another approach is to rasterize the DFAD and use a texture warp function to color the terrain. This would not permit declutter of individual features but only classes (by color). Terrain color show-through in sparse overlay areas would be handled by a transparent color code (screen door effect). No verticality is achieved.

There are 298 categories of aerial, linear, and point features. Linear features must be expanded to a double line to prevent interlace strobing. A point feature contains a length, width, and height which can be used by the symbol generator for expansion. A typical lake contains 900 vertices and produces 10 to 20 active edges for rendering at any given scan line. The number of vertices is limited to 512. The display list is 64K bytes for a 1:250K buffer segment. Any given feature could have 32K vertices.

Up to 2K bytes of display list per buffer segment DTED is accommodated for DFAD. The DSM can tag the classes or individual features for clutter/declutter by toggling bits in the overlay RAM of the SHAG.

The symbol generator processes macros and graphic primitives which are passed to the rendering engine. These primitives include lines, arcs, alphanumerics, and two dimensional symbology. The rendering engine

draws these primitives and outputs pixels which are anti-aliased. The GAP transforms these polygons and passes them to the rendering engine. A complete 4×4 Euler transformation is performed. Typical macros include compass rose and range scale symbols. Given a macro command, the symbol generator produces the primitive graphics calls to the rendering engine. This mode operates in plan view only and implements two dimensional symbols. Those skilled in the art will appreciate that the invention is not limited to specific fonts.

Three dimensional symbology presents the problem of clipping to the view volume. A gross clip is handled by the DSM in the cache memory at scan out time. The base of a threat dome, for example, may lie outside the orthographic projection of the view volume onto cache, yet a part of its dome may end up visible on the screen. The classical implementation performs the functions of tiling, transforming, clipping to the view volume (which generates new polygons), and then rendering. A gross clip boundary is implemented in cache around the view volume projection to guarantee inclusion of the entire symbol. The anomaly under animation to be avoided is that of having symbology sporadically appear and disappear in and out of the frame at the frame boundaries. A fine clip to the screen is performed downstream by the rendering engine. There is a 4K boundary around the screen which is rendered. Outside of this boundary, the symbol will not be rendered. This causes extra rendering which is clipped away.

Threat domes are represented graphically in one embodiment by an inverted conic volume. A threat/intelligence file contains the location and scaling factors for the generic model to be transformed to the specific threats. The tiling engine contains the connectivity information between the vertices and generates the planar polygons. The threat polygons are passed to the rendering engine with various viewing parameters such as mesh, opaque, dot, transparent, and so forth.

Graticles represent latitude and longitude lines, UTM klicks, and so forth which are warped onto the map in perspective. The symbol generator produces these lines.

Freeze frame is implemented in plan view only. The cursor is flown around the screen, and is generated by the symbol generator.

Programmable blink capability is accommodated in the invention. The DSM updates the overlay RAM toggle for display. The processor clock is used during variable frame update rate to control the blink rate.

A generic threat symbol is modeled and stored in the three dimensional symbol generation library. Parameters such as position, threat range, and angular threat view are passed to the symbol generator as a macro call (similar to a compass rose). The symbol generator creates a polygon list for each threat instance by using the parameters to modify the generic model and place it in the world coordinate system of the terrain data base. The polygons are transformed and rendered into screen space by the perspective view pipeline. These polygons form only the outside envelope of the threat cone.

This invention has been described herein in considerable detail in order to comply with the Patent Statues and to provide those skilled in the art with the information needed to apply the novel principles and to construct and use such specialized components as are required. However, it is to be understood that the invention can be carried out by specifically different equipment and devices, and that various modifications, both as to the equipment details and operating procedures, can be accomplished without departing from the scope of the invention itself.

What is claimed is:

1. A system for providing a texture mapped perspective view for a digital map system wherein objects are transformed from texture space having U, V coordinates to screen space having X, Y coordinates comprising:

(a) a cache memory means for storing terrain data including elevation posts, wherein the cache memory means includes an output and an address bus;

(b) a shape address generator means for scanning cache memory having an ADDRESS SIGNAL coupled to the cache memory means address bus wherein the shape address generator means scans the elevation posts out of the cache memory means;

(c) a geometry engine coupled to the cache memory means output to receive the elevation posts scanned from the cache memory by the shape address generator means, the geometry engine including means for

i. transformation of the scanned elevation posts from object space to screen space so as to generate transformed vertices in screen coordinates for each elevation post, and

ii. generating three dimensional coordinates;

(d) a tilling engine coupled to the geometry engine for generating planar polygons from the generated three dimensional coordinates;

(e) a symbol generator to the geometry engine for transmitting a vertex list to the geometry engine wherein the geometry engine operates on the vertex list to transform the vertex list into screen space X, Y coordinates and passes the screen space X, Y coordinates to the tilling engine for generating planar polygons which form icons for display and processing information from the tilling engine into symbols,

(f) a texture engine means coupled to receive the ADDRESS SIGNAL from the shape address generator means including a texture memory and including a means for generating a texture vertex address to texture space correlated to an elevation post address and further including a means for generating a texture memory address for scanning the texture memory wherein the texture memory provides texture data on a texture memory data bus in response to being scanned by the texture memory address;

(g) a rendering engine having an input coupled to the tilling engine and the texture memory data bus for generating image data from the planar polygons; and

(h) a display memory for receiving image data from the rendering engine output wherein the display memory includes at least four first-in, first-out memory buffers.

2. The apparatus of claim **1** wherein each polygon has a surface and the rendering means assigns one color across the surface of each polygon.

3. The apparatus of claim **1** wherein the vertices of each polygon have an intensity and the rendering means interpolates the intensities between the vertices of each polygon in a linear fashion.

4. The apparatus of claim **1** wherein the rendering means further includes means for generating transparent polygons and passing the transparent polygon to the display memory.

5. A method for providing a texture mapped perspective view for a digital map system having a cache memory, a geometry engine coupled to the cache memory, a shape address generator coupled to the cache memory, a tiling engine coupled to the geometry engine, a symbol generator coupled to the geometry engine and the tiling engine, a texture engine coupled to the cache memory, a rendering engine coupled to the tiling engine and the texture engine, and a display memory coupled to the rendering engine, wherein objects are transformed from texture space having U, V coordinates to screen space having X, Y coordinates, the method comprising the steps of:

(a) storing terrain data, including elevation posts, in the cache memory;

(b) scanning the cache memory to retrieve the elevation posts;

(c) transforming the terrain data from elevation posts in object space to transformed vertices in screen space, and

(d) generating planar polygons from the generated three dimensional coordinates;

(e) transmitting a vertex list to the geometry engine, operating the geometry engine to transform the vertex list into screen space X, Y coordinates and passing the screen space X, Y coordinates to the tiling engine for generating planar polygons which form icons for display;

(f) tagging elevation posts with corresponding addresses in texture space;

(g) generating image data in the rendering engine from the planar polygons and the tagged elevation posts; and

(h) storing the generated image data in the display memory wherein the display memory comprises at least four first-in, first-out memory buffers and the step of storing the generated images includes storing the generated image data in the at least four First-in, First-out memory buffers.

6. The method of claim 5 wherein each polygon has a surface and wherein the step of generating image data further includes the steps of assigning one color across the surface of each polygon.

7. The method of claim 5 wherein the vertices of each polygon have an intensity and the step of generating image data further includes the step of interpolating the intensities between the vertices of each polygon in a linear fashion.

8. The method of claim 5 wherein the step of generating image data further includes the step of generating transparent polygons and passing the transparent polygons to the display memory.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,179,638
DATED : January 12, 1993
INVENTOR(S) : John F. Dawson, Thomas D. Snodgrass, and James A. Cousens

It is certified that error appears in the above-indentified patent and that said Letters Patent is hereby corrected as shown below:

Column 17, line 21, after "and" insert --generating three dimensional coordinates for the transformed vertices in screen space--.

Signed and Sealed this

Twenty-second Day of March, 1994

Attest:

BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks

Computer Graphics Volume 17, Number 3 July 1983

Pyramidal Parametrics

Lance Williams

Computer Graphics Laboratory
New York Institute of Technology
Old Westbury, New York

Abstract

The mapping of images onto surfaces may substantially increase the realism and information content of computer-generated imagery. The projection of a flat source image onto a curved surface may involve sampling difficulties, however, which are compounded as the view of the surface changes. As the projected scale of the surface increases, interpolation between the original samples of the source image is necessary; as the scale is reduced, approximation of multiple samples in the source is required. Thus a constantly changing sampling window of view-dependent shape must traverse the source image.

To reduce the computation implied by these requirements, a set of prefiltered source images may be created. This approach can be applied to particular advantage in animation, where a large number of frames using the same source image must be generated. This paper advances a "pyramidal parametric" prefiltering and sampling geometry which minimizes aliasing effects and assures continuity within and between target images.

Although the mapping of texture onto surfaces is an excellent example of the process and provided the original motivation for its development, pyramidal parametric data structures admit of wider application. The aliasing of not only surface texture, but also highlights and even the surface representations themselves, may be minimized by pyramidal parametric means.

General Terms: Algorithms.

Keywords and Phrases: Antialiasing, Illumination Models, Modeling, Pyramidal Data Structures, Reflectance Mapping, Texture Mapping, Visible Surface Algorithms.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation--display algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling--curve, surface, solid and object representations, geometric algorithms, languages and systems; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism--color, shading, shadowing, and texture.

1. Pyramidal Data Structures

Pyramidal data structures may be based on various subdivisions: binary trees, quad trees, oct trees, or n-dimensional hierarchies [17]. The common feature of these structures is a succession of levels which vary the resolution at which the data is represented.

The decomposition of an image by two-dimensional binary subdivision was a pioneering strategy in computer graphics for visible surface determination [15]. The approach was essentially a synthesis-by-analysis: the image plane was subdivided into quadrants recursively until analysis of a subsection showed that surface ordering was sufficiently simple to permit rendering. Such subdivision and analysis has been subsequently adopted to generate spatial data structures [5], which have been used to represent images [9] both for pattern recognition [13] and for transmission [10], [14]. In the field of computer graphics, such data structures have been adopted for texture mapping [4], [16], and generalized to represent objects in space [11].

The application of pyramidal data to image storage and transmission may permit significant compression of the data to be stored or transmitted. This is so because highly detailed features may be localized within an otherwise low-frequency image, permitting the sampling rate to be reduced for large sections of the image. Besides permitting bandwidth compression, the representation orders data in such a way that the general character of images may be recalled or transmitted before the specific details.

Pattern recognition and classification often require the comparison of a candidate image against a set of canonical patterns. This is an operation the expense of which increases as the square of the resolution at which it is performed. The use of pyramidal data structures in pattern recognition and classification permits the comparison of the gross features of two-dimensional functions preliminary to the minute particulars; a good general reference on this application is [12].

Computer Graphics Volume 17, Number 3 July 1983

In computer graphics, pyramidal texture maps may be used to perform arbitrary mappings of a function with minimal aliasing artifacts and reduced computation. Once again, images may be represented at different spatial bandwidths. The concern is that inappropriate resolution misrepresents the data; that is, sampling high-resolution data at larger sample intervals invites aliasing.

2. Parametric Interpolation

By a pyramidal parametric data structure, we will mean simply a pyramidal structure with both intra- and inter-level interpolation. Consider the case of an image represented as a two-dimensional array of samples. Interpolation is necessary to produce a continuous function of two parameters, U and V. If, in addition, a third parameter (call it D) moves us up and down a hierarchy of corresponding two-dimensional functions, with interpolation between (or among) the levels of the pyramid providing continuity, the structure is pyramidal parametric.

The practical distinction between such a structure and an ordinary interpolant over an n-dimensional array of samples is that the number of samples representing each level of the pyramid may be different.

3. Mip Mapping

"Mip" mapping is a particular format for two-dimensional parametric functions, which, along with its associated addressing scheme, has been used successfully to bandlimit texture mapping at New York Institute of Technology since 1979. The acronym "mip" is from the Latin phrase "multum in parvo," meaning "many things in a small place." Mip mapping supplements bilinear interpolation of pixel values in the texture map (which may be used to smoothly translate and magnify the texture) with interpolation between prefiltered versions of the map (which may be used to compress many pixels into a small place). In this latter capacity, mip offers much greater speed than texturing algorithms which perform explicit convolution over an area in the texture map for each pixel rendered [1], [6].

Mip owes its speed in compressing texture to two factors. First, a fair amount of filtering of the original texture takes place when the mip map is first created. Second, subsequent filtering is approximated by blending different levels of the mip map. This means that all filters are approximated by linearly interpolating a set of square box filters, the sides of which are powers-of-two pixels in length. Thus, mapping entails a fixed overhead, which is independent of the area filtered to compute a sample.

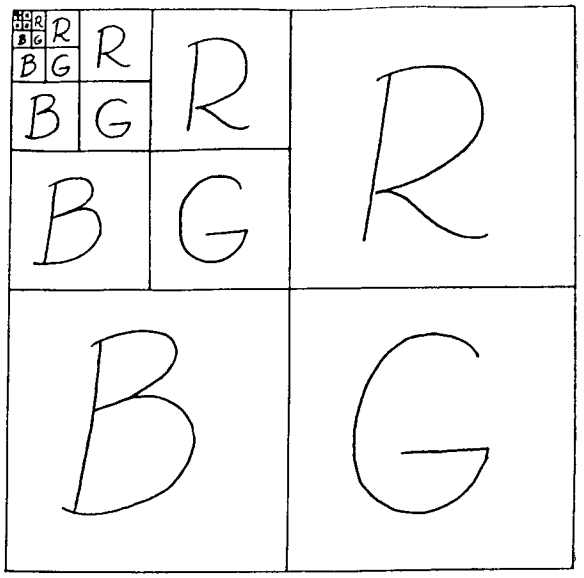


Figure (1)
Structure of a Color Mip Map
Smaller and smaller images diminish into the upper left corner of the map. Each of the images is averaged down from its larger predecessor.

(Below:)
Mip maps are indexed by three coordinates: U, V, and D. U and V are spatial coordinates of the map; D is the variable used to index, and interpolate between, the different levels of the pyramid.

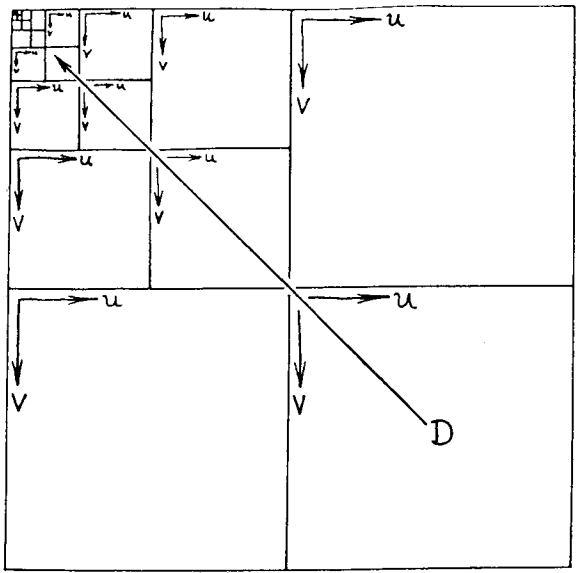


Figure (1) illustrates the memory organization of a color mip map. The image is separated into its red, green, and blue components (R, G, and B in the diagram). Successively filtered and downsampled versions of each component are instanced above and to the left of the originals, in a series of smaller and smaller images, each half the linear dimension (and a quarter the number of

Computer Graphics Volume 17, Number 3 July 1983

samples) of its parent. Successive divisions by four partition the frame buffer equally among the three components, with a single unused pixel remaining in the upper left-hand corner.

The concept behind this memory organization is that corresponding points in different prefiltered maps can be addressed simply by a binary shift of an input U, V coordinate pair. Since the filtering and sampling are performed at scales which are powers of two, indexing the maps is possible with inexpensive binary scaling. In a hardware implementation, the addresses in all the corresponding maps (now separate memories) would be instantly and simultaneously available from the U, V input.

The routines for creating and accessing mip maps at NYIT are based on simple box (Fourier) window prefiltering, bilinear interpolation of pixels within each map instance, and linear interpolation between two maps for each value of D (the pyramid's vertical coordinate). For each of the three components of a color mip map, this requires 8 pixel reads and 7 multiplications. This choice of filters is strictly for the sake of speed. Note that the bilinear interpolation of pixel values at the extreme edges of each map instance must be performed with pixels from the opposite edge(s) of that map, for texture which is periodic. For non-periodic texture, scaling or clipping of the U, V coordinates prevents the intrusion of an inappropriate map or color component into the interpolation.

The box (Fourier) window used to create the mip maps illustrated here, and the tent (Bartlett) window used to interpolate them, are far from ideal; yet probably the most severe compromise made by mip filtering is that it is symmetrical. Each of the prefiltered levels of the map is filtered equally in X and Y. Choosing a value of D trades off aliasing against blurring, which becomes a tricky proposition as a pixel's projection in the texture map deviates from symmetry. Heckbert [8] suggests:

$$d = \max\left(\sqrt{\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial v}{\partial x}\right)^2}\ ,\ \sqrt{\left(\frac{\partial u}{\partial y}\right)^2+\left(\frac{\partial v}{\partial y}\right)^2}\right)$$

where D is proportional to the "diameter" of the area in the texture to be filtered, and the partials of U and V (the texture-map coordinates) with respect to X and Y (the screen coordinates) can be calculated from the surface projection.

Illustrations of mapping performed by the mip technique are the subject of Figures (2) through (10). The NYIT Test Frog in Figure (2) is magnified by simple point sampling in (3), and by interpolation in (4). The hapless amphibian is similarly

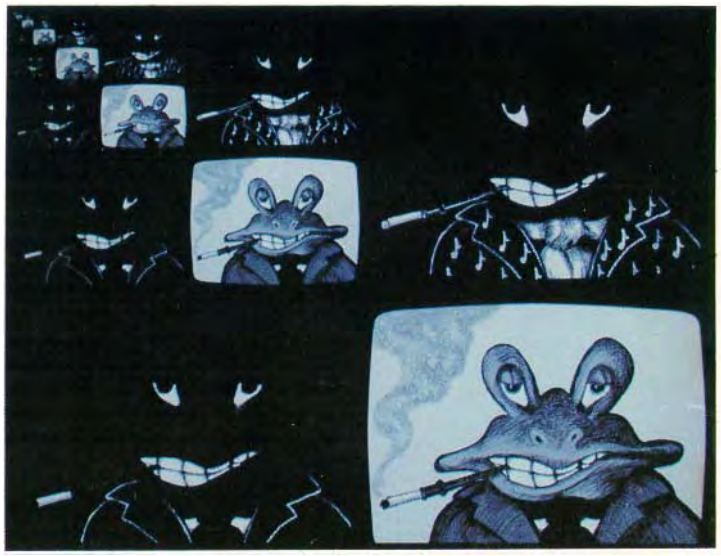


Figure (2)
Mip map of the flexible NYIT Test Frog.

compressed by point sampling in (5) and by mipping in (6).

The more general and interesting case -- continuously variable upsampling and downsampling of the original texture -- is illustrated in (7) on a variety of surfaces. Since the symmetry of mip filtering would be expected to show up badly when texture is compressed in only one dimension, figures (8) through (10) are of especial interest. These pictures, created by Ed Emshwiller at NYIT for his videotape, "Sunstone," were mapped using Alvy Ray Smith's TEXAS animation program, which in turn used MIP to antialias texture. As the panels rotate edge-on, the texture collapses to a line smoothly and without apparent artifacts.

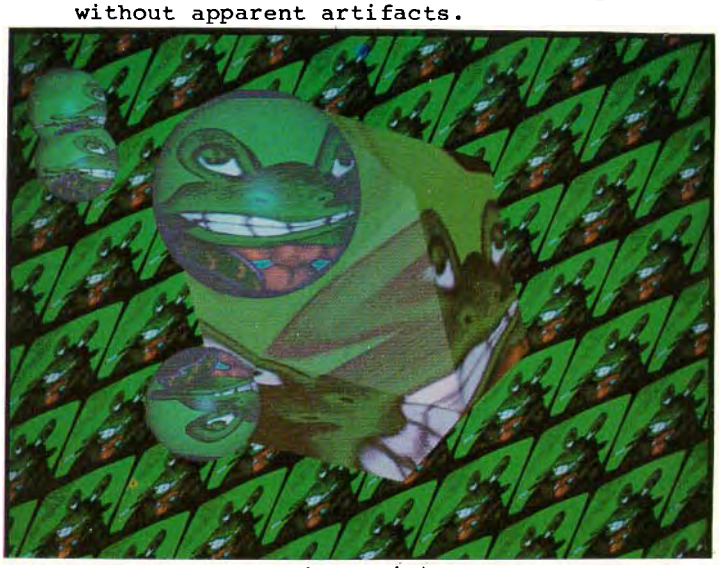


Figure (7)
General mapping: interpolation and pyramidal compression.

Computer Graphics Volume 17, Number 3 July 1983

Figure (3)
Upsampling the frog: magnification by point sampling.

Figure (4)
Upsampling the frog: magnification by bilinear interpolation.

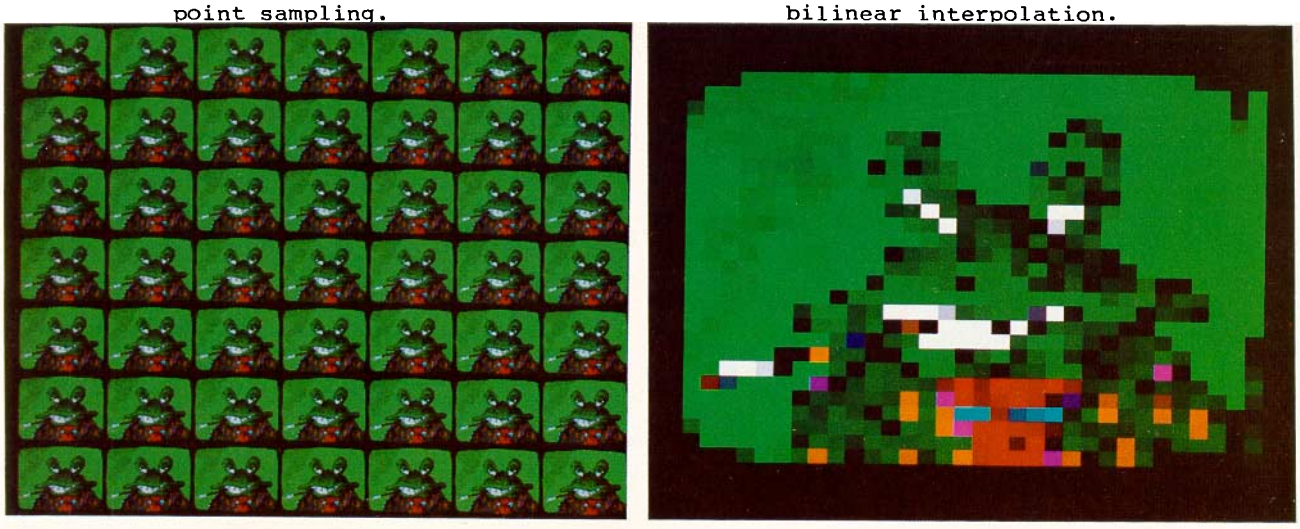


Figure (5)
Downsampling the frog: compression by point sampling (detail, right).

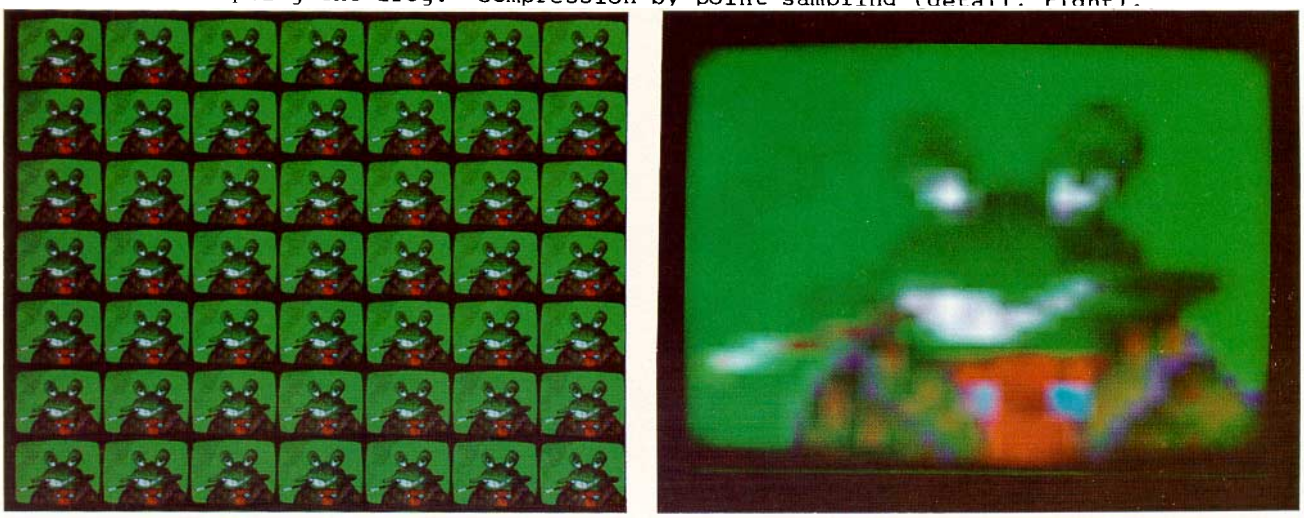
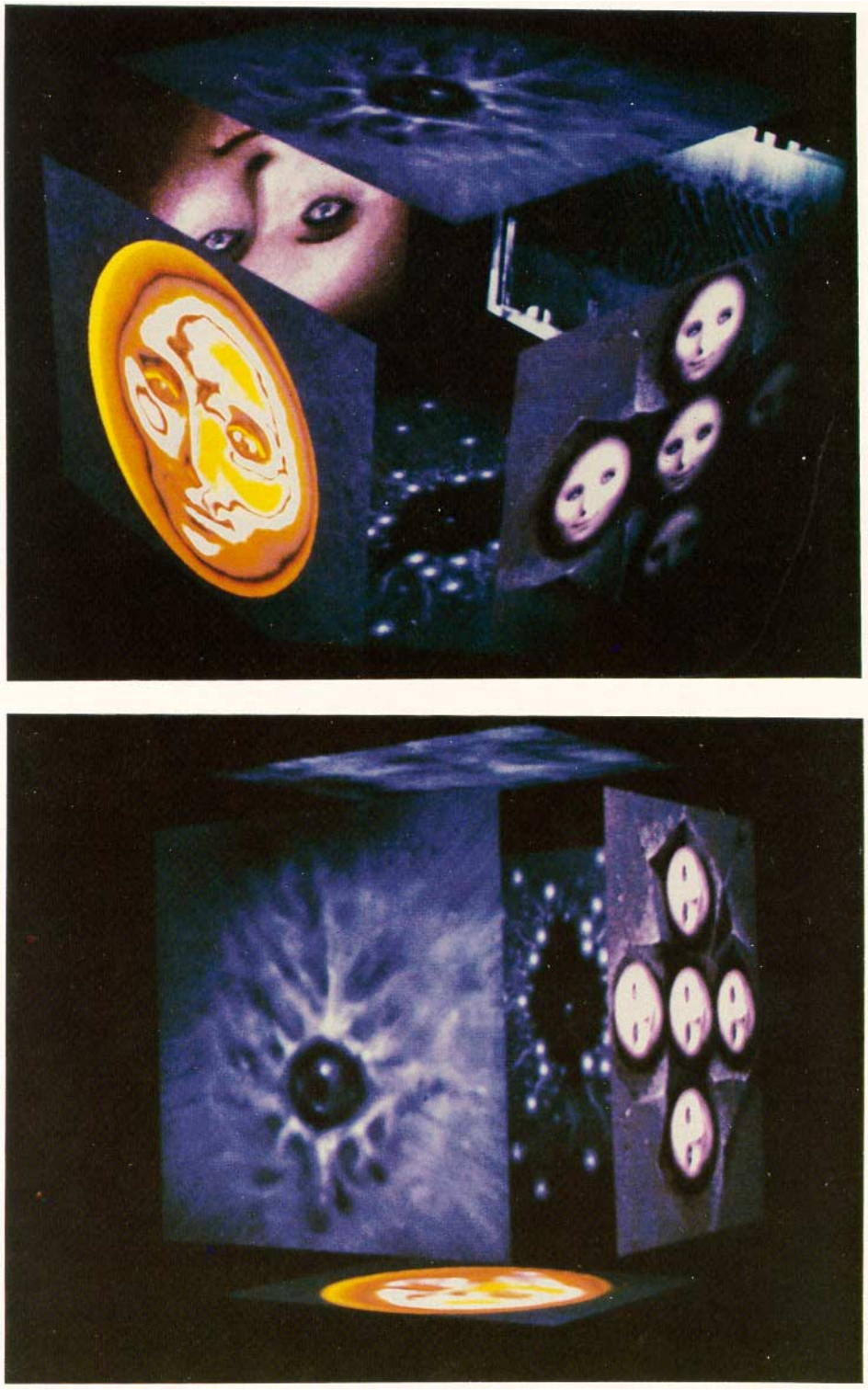


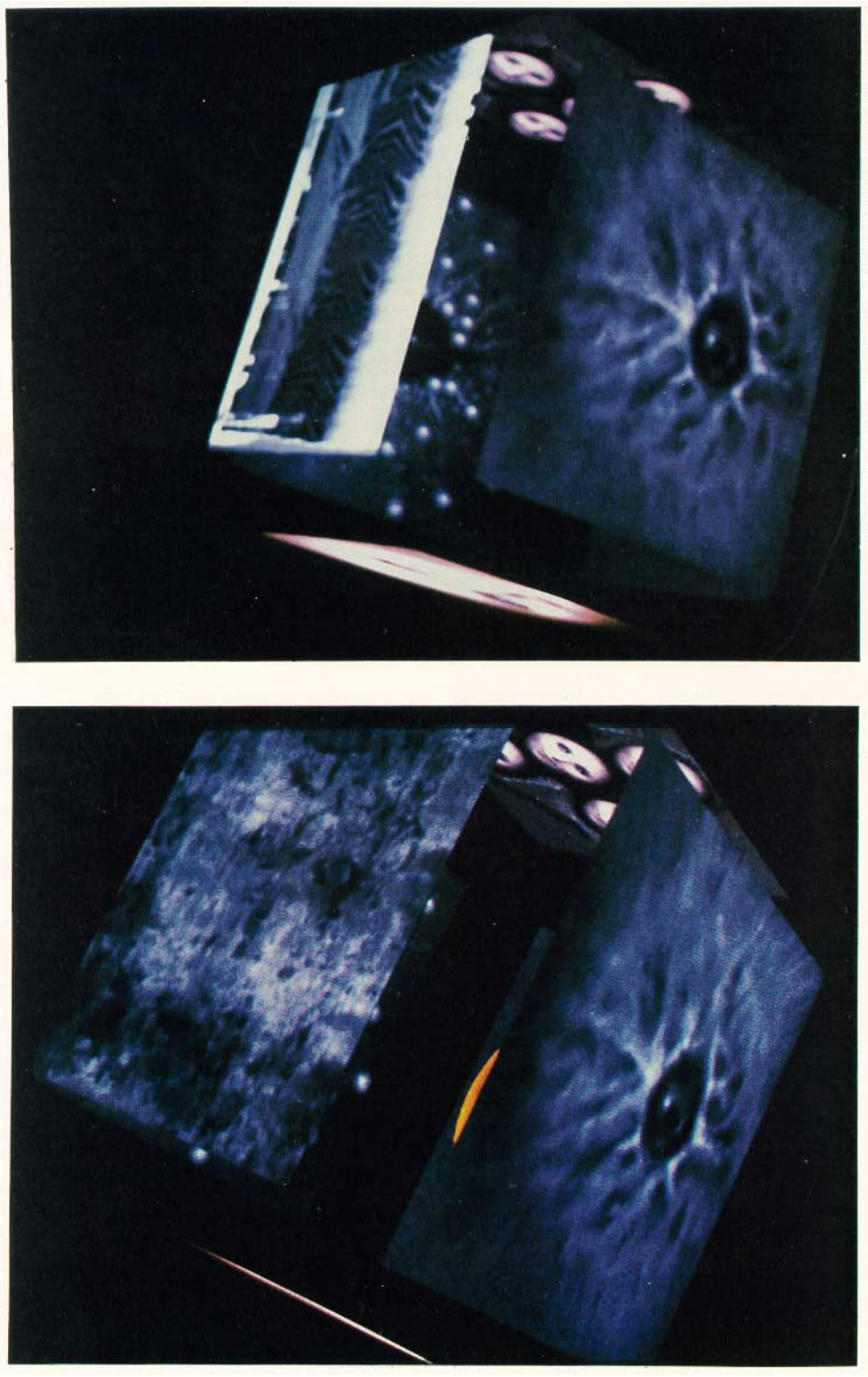
Figure (6)
Downsampling: compression by pyramidal interpolation (detail, right).

Computer Graphics Volume 17, Number 3 July 1983



Figures (8)-(9)
"Sunstone" by Ed Emshwiller, segment animated by Alvy Ray Smith
Pyramidal parametric texture mapping on polygons.

Computer Graphics Volume 17, Number 3 July 1983



Figures (10)-(11)
"Sunstone" by Ed Emshwiller, segment animated by Alvy Ray Smith
Pyramidal parametric texture mapping on polygons.

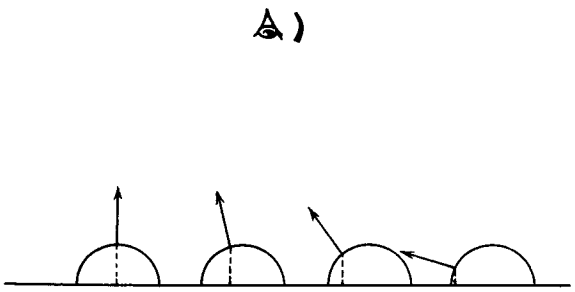
Computer Graphics Volume 17, Number 3 July 1983

4. Highlight Antialiasing

As small or highly curved objects move across a raster, their surface normals may beat erratically with the sampling grid. This causes the shading values to flash annoyingly in motion sequences, a symptom of illumination aliasing. The surface normals essentially point-sample the illumination function.

Figure (12) illustrates samples of the surface normals of a set of parallel cylinders. The cylinders in the diagram are depicted as if from the edge of the image plane; the regularly-spaced vertical line segments are the samples along a single axis. The arrows at the sample points indicate the directions of the surface normals. Depending on the shading formula invoked, there may be very high contrast between samples where the normal is nearly parallel to the sample axis, and samples where the normal points directly at the observer's eye.

Figure (12)



The shading function depends not only on the shape of the surface, but its light reflection properties (characterized by the shading formula), the position of the light source, and the position of the observer's eye. Hanrahan [7] expresses it in honest Greek:

$$\int_x \int_y \varphi(E,N,L) \frac{\partial(u,v)}{\partial(x,y)} \, dxdy$$

where the normal, N, the light sources, L, and the eye, E, are vectors which may each be functions of U and V, and the limits of integration are the X, Y boundaries of the pixel.

Figure (13) illustrates highlight aliasing on a perfectly flat surface. The viewing conventions of the diagram are the same as in Figure (12). "L" is the direction vector of the light source; the surface is a polygon at an angle to the image plane; the dotted bump is a graph of the reflected light, characteristic of a specular surface reflection function. The highlight indicated by the bump falls entirely between the samples. (Note that this is only possible on a flat surface if either the eye or the light is local, a point in space rather than simply a direction vector. Some boring shading formulae exclude the possibility of highlight aliasing on polygons by requiring all flat surfaces to be flat in shading.)

Figure (13)

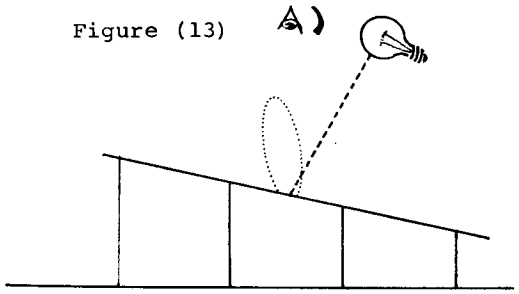
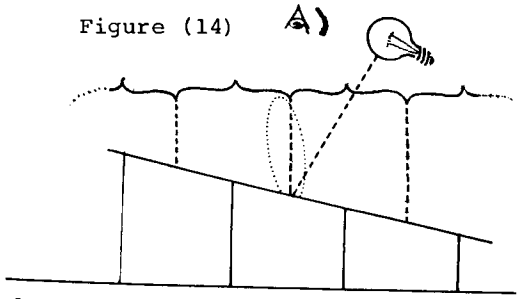


Figure (14)



A first attempt to overcome the limitations of point-sampling the illumination function is to integrate the function over the projected area represented by each sample point. This approach is illustrated in Figure (14). The brackets at each sample represent the area of the surface over which the illumination function is integrated. This procedure is analogous to area-averaging of sampled edges or texture [3].

In order to generalize this approach to curved surfaces, the "sample interval" over which illumination is integrated must be modified according to the local curvature of the surface at a sample. In Figure (15), the area of a surface represented by a pixel has been projected onto a curved surface. The solid angle over which illumination must be integrated is approximated by the volume enclosed by the normals at the pixel corners. The distribution of light within this volume will sum to an estimate of the diffuse reflection over the pixel. If the surface exhibits undulations at the pixel level, however, aliasing will result.

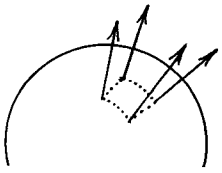


Figure (15)

Computer Graphics Volume 17, Number 3 July 1983

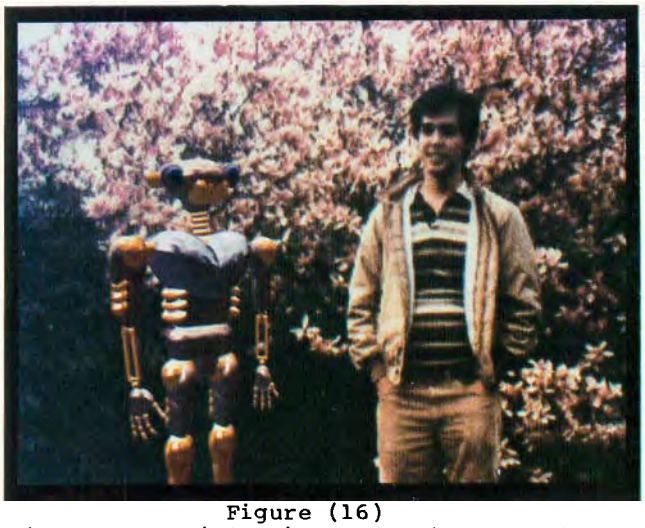


Figure (16)
Michael Chou (right) poses with an imaginary companion. Reflectance maps can enhance the realism of synthetic shading.

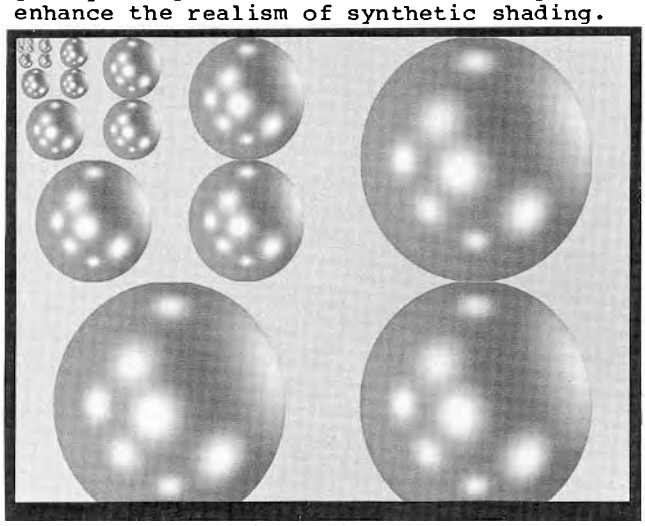


Figure (17)
A pyramidal parametric reflectance map, containing 9 light sources. The region outside the "sphere" is unused.

We might divide the surface up into regions of relatively low curvature (as is done in some patch rendering algorithms), and rely on "edge antialiasing" to integrate the different surfaces within a pixel. Alternatively, we may develop some mechanism for limiting the local curvature of surfaces before rendering. This possibility is explored in the next section.

If we represent the illumination of a scene as a two-dimensional map, highlights can be effectively antialiased in much the same way as textures. Blinn and Newell [1] demonstrated specular reflection using an illumination map. The map was an image of the environment (a spherical projection of the scene, indexed by the X and Y components of the surface normals) which could be used to cast reflections onto specular surfaces. The impression of mirrored facets and chrome objects which can be achieved with this method is striking; Figure (16) provides an illustration. Reflectance mapping is not, however, accurate for local reflections. To achieve similar results with three dimensional accuracy requires ray-tracing.

A pyramidal parametric illumination map permits convenient antialiasing of highlights as long as a good measure of local surface curvature is available. The value of "D" used to index the map is proportional to the solid angle subtended by the surface over the pixel being computed; this may be estimated by the same formula used to compute D for ordinary texture mapping. Nine light sources of varying brightness glint raggedly from the test object in Figure (18); the reflectance map in Figure (17) provided the illumination. In Figure (19), convincing highlight antialiasing results from the full pyramidal parametric treatment.

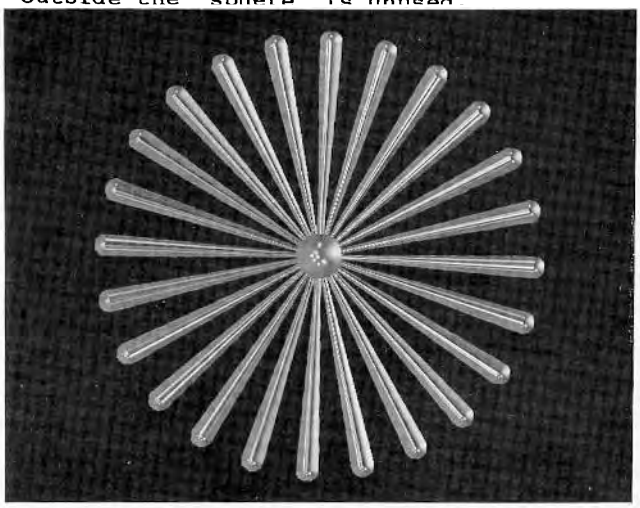


Figure (18) Before

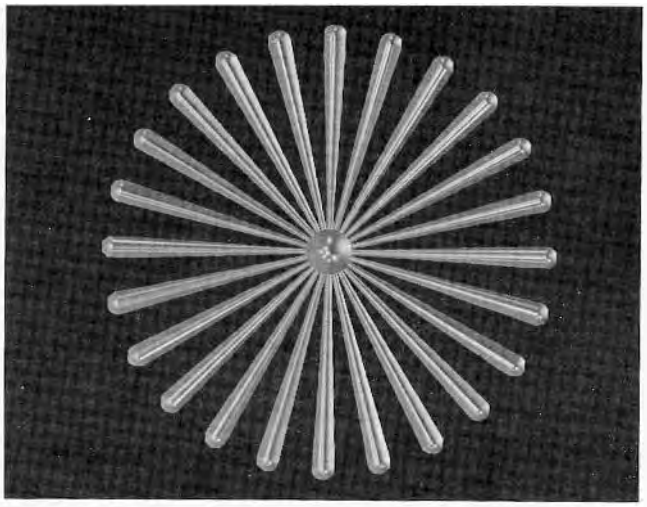
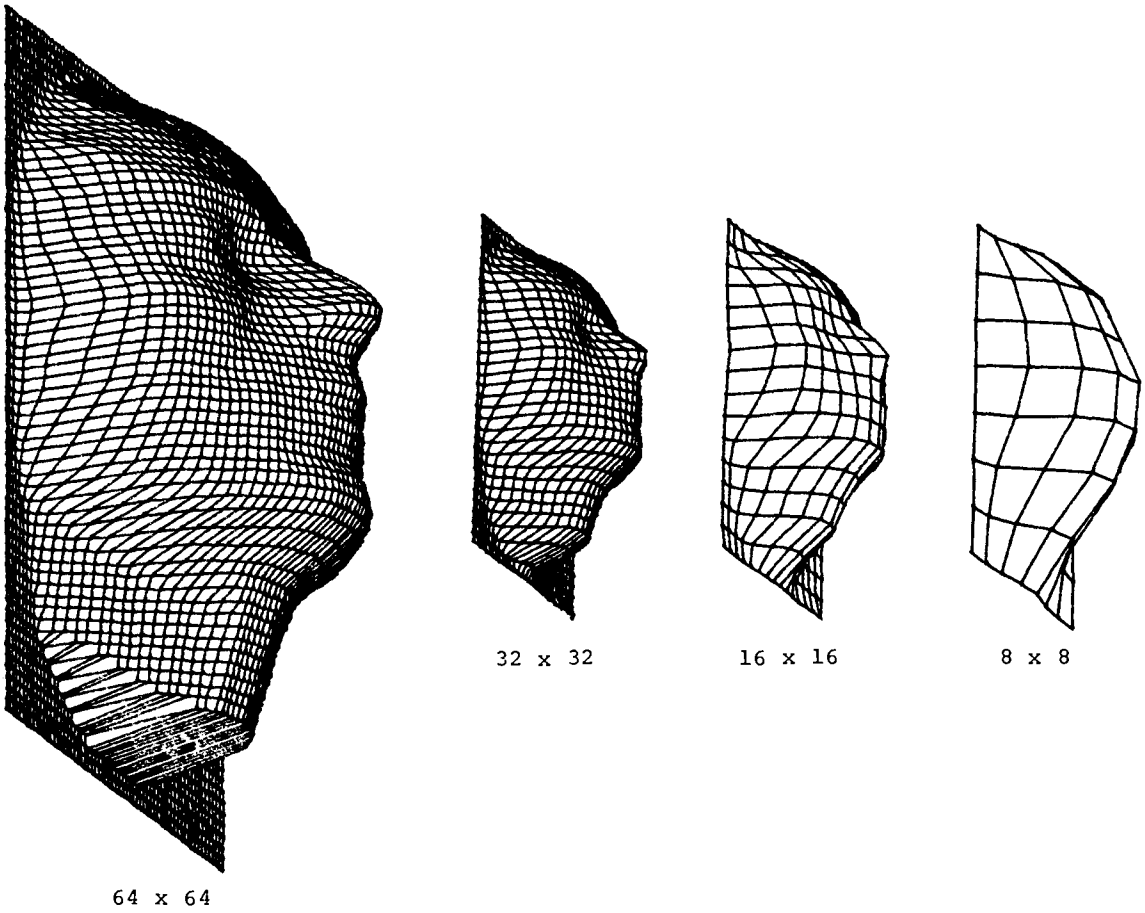


Figure (19) After

Computer Graphics Volume 17, Number 3 July 1983



Figures (20-23) Different resolution meshes.

5. Levels of Detail in Surface Representation

In addition to bandlimiting texture and illumination functions for mapping onto a surface, pyramidal parametrics may be used to limit the level of detail with which the surface itself is represented. The goal is to represent an object for graphic display as economically as its projection on the image plane permits, without boiling and sparkling aliasing artifacts as the projection changes.

The expense of computing and shading each pixel dominates the cost of many algorithms for rendering higher-order surfaces. For meshes of polygons or patch control points which project onto a small portion of the image, however, the vertex (or control-point) expense dominates. In these situations it is desirable to reduce the number of points used to represent the object.

A pyramidal parametric data structure the components of which are spatial coordinates (the X-Y-Z of the vertices of a rectangular mesh, for example, as opposed to the R-G-B of a texture or illumination map) provides a continuously-variable filtered instance of the surface for sampling at any desired degree of resolution.

Figures (20) through (23) illustrate a simple surface based on a human face model developed by Fred Parke at the University of Utah. As the sampling density varies, so does the filtering of the surface. These faces are filtered and sampled by the same methods previously discussed for texture and reflectance maps. Pyramidal parametric representations such as these appear promising for reducing aliasing effects as well as systematically sampling very large data bases over a wide range of scales and viewing angles.

6. Conclusions

Pyramidal data structures are of proven value in image analysis and have interesting application to image bandwidth compression and transmission. "Pyramidal parametrics," pyramidal data structures with intra- and inter-level interpolation, are here proposed for use in image synthesis. By continuously varying the detail with which data are resolved, pyramidal parametrics provide economical approximate solutions to filtering problems in mapping texture and illumination onto surfaces, and preliminary experiments suggest they may provide flexible surface representations as well.

7. Acknowledgments

I would like to acknowledge Ed Catmull, the first (to my knowledge) to apply multiple prefiltered images to texture mapping: the method was applied to the bicubic patches in his thesis, although it was not described. Credit is also due Tom Duff, who wrote both recursive and scan-order routines for creating mip maps which preserved numerical precision over all map instances; Dick Lundin, who wrote the first assembly-coded mip map accessing routines; Ephraim Cohen, who wrote the second; Rick Ace, who translated Ephraim's PDP-11 versions for the VAX assembler; Paul Heckbert, for refining and speeding up both creation and accessing routines, and investigating various estimates of "D"; Michael Chou, for implementing highlight antialiasing and high-resolution reflectance mapping on quadric surfaces.

I owe special thanks to Jules Bloomenthal, Michael Chou, Pat Hanrahan, and Paul Heckbert for critical reading and numerous helpful suggestions in the course of preparing this text. Photographic support was provided by Michael Lehman.

8. References

[1] Blinn, J., and Newell, M., "Texture and Reflection on Computer Generated Images," CACM, Vol. 19, #10, Oct. 1976, pp. 542-547.

[2] Bui-Tuong Phong, "Illumination for Computer Generated Pictures," PhD. dissertation, Department of Computer Science, University of Utah, December 1978.

[3] Crow, F.C., "The Aliasing Problem in Computer Synthesized Shaded Images," PhD. dissertation, Department of Computer Science, University of Utah, Tech. Report UTEC-CSc-76-015, March 1976.

[4] Dungan, W., Stenger, A., and Sutty, G., "Texture Tile Considerations for Raster Graphics," SIGGRAPH 1978 Proceedings, Vol. 12, #3, August 1978.

[5] Eastman, Charles M., "Representations for Space Planning," CACM, Vol. 13, #4, April 1970.

[6] Feibush, E.A., Levoy, M., and Cook, R.L., "Synthetic Texturing Using Digital Filters," Computer Graphics, Vol. 14, July, 1980.

[7] Hanrahan, Pat, private communication, 1983.

[8] Heckbert, Paul, "Texture Mapping Polygons in Perspective," NYIT Computer Graphics Lab Tech. Memo #13, April, 1983.

[9] Klinger, A., and Dyer, C.R., "Experiments on Picture Representation Using Regular Decomposition," Computer Graphics and Image Processing, #5, March, 1976.

[10] Knowlton, K., "Progressive Transmission of Gray-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes," Proceedings of the IEEE, Vol. 68, #7, July 1980, pp. 885-896.

[11] Meagher, D., "Octree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary 3D Objects by Computer," IPL-TR-80-111, Image Processing Lab, Electrical and Systems Engineering Dept., Rensselaer Polytechnic Institute, October 1980.

[12] Tanimoto, S.L., and Klinger, A., Structured Computer Vision, Academic Press, New York, 1980.

[13] Tanimoto, S.L., and Pavlidis, T., "A Hierarchical Data Structure for Picture Processing," Computer Graphics and Image Processing, Vol. 4, #2, June 1975.

[14] Tanimoto, S.L., "Image Processing with Gross Information First," Computer Graphics and Image Processing 9, 1979.

[15] Warnock, J.E., "A Hidden-Line Algorithm for Halftone Picture Representation," Department of Computer Science, University of Utah, TR 4-15, 1969.

[16] Williams, L., "Pyramidal Parametrics," SIGGRAPH tutorial notes, "Advanced Image Synthesis," 1981.

[17] Yau, M.M., and Srihari, S.N., "Recursive Generation of Hierarchical Data Structures for Multidimensional Digital Images," Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing, August 1981.

APPENDIX M

Next Up Previous Contents Index
Next: 3.8.2 Texture Magnification **Up:** 3.8.1 Texture Minification **Previous:** 3.8.1 Texture Minification

Mipmapping

`TEXTURE_MIN_FILTER` values `NEAREST_MIPMAP_NEAREST`, `NEAREST_MIPMAP_LINEAR`, `LINEAR_MIPMAP_NEAREST`, and `LINEAR_MIPMAP_LINEAR` each require the use of a *mipmap*. A mipmap is an ordered set of arrays representing the same image; each array has a resolution lower than the previous one. If the texture has dimensions $2^n \times 2^m$, then there are $\max\{n, m\} + 1$ mipmap arrays. The first array is the original texture with dimensions $2^n \times 2^m$. Each subsequent array has dimensions $2^{(k-1)} \times 2^{(l-1)}$ where $2^k \times 2^l$ are the dimensions of the previous array. This is the case as long as both **k>0** and **l>0**. Once either **k=0** or **l=0**, each subsequent array has dimension $1 \times 2^{(l-1)}$ or $2^{(k-1)} \times 1$, respectively, until the last array is reached with dimension 1×1.

Each array in a mipmap is transmitted to the GL using **TexImage2D** or **TexImage1D** ; the array being set is indicated with the *level-of-detail* argument. Level-of-detail numbers proceed from **0** for the original texture array through $p = \max\{n, m\}$ with each unit increase indicating an array of half the dimensions of the previous one as already described. If texturing is enabled (and `TEXTURE_MIN_FILTER` is one that requires a mipmap) at the time a primitive is rasterized and if the set of arrays **0** through **p** is incomplete, based on the dimensions of array **0**, then it is as if texture mapping were disabled. The set of arrays **0** through **p** is incomplete if the internal formats of all the mipmap arrays were not specified with the same symbolic constant, or if the border widths of the mipmap arrays are not the same, or if the dimensions of the mipmap arrays do not follow the sequence described above. Arrays indexed greater than **p** are insignificant.

The mipmap is used in conjunction with the level of detail to approximate the application of an appropriately filtered texture to a fragment. Let $p = \max\{n, m\}$ and let **c** be the value of λ at which the transition from minification to magnification occurs (since this discussion pertains to minification, we are concerned only with values of λ where $\lambda > c$). For `NEAREST_MIPMAP_NEAREST`, if $c < \lambda \leq 0.5$ then the mipmap array with level-of-detail of 0 is selected. Otherwise, the **d**th mipmap array is selected when $d - \frac{1}{2} < \lambda \leq d + \frac{1}{2}$ as long as $1 \leq d \leq p$. If $\lambda > p + \frac{1}{2}$, then the **p**th mipmap array is selected. The rules for `NEAREST` are then applied to the selected array.

The same mipmap array selection rules apply for `LINEAR_MIPMAP_NEAREST` as for `NEAREST_MIPMAP_NEAREST`, but the rules for `LINEAR` are applied to the selected array.

For `NEAREST_MIPMAP_LINEAR`, the level **d-1** and the level **d** mipmap arrays are selected, where $d - 1 \leq \lambda < d$, unless $\lambda \geq p$, in which case the **p**th mipmap array is used for both arrays. The rules

for NEAREST are then applied to each of these arrays, yielding two corresponding texture values τ_{d-1} and τ_d. The final texture value is then found as

$$\tau = [1 - \mathrm{frac}(\lambda)]\tau_{d-1} + \mathrm{frac}(\lambda)\tau_d.$$

LINEAR_MIPMAP_LINEAR has the same effect as NEAREST_MIPMAP_LINEAR except that the rules for LINEAR are applied for each of the two mipmap arrays to generate τ_{d-1} and τ_d.

Next Up Previous Contents Index
Next: 3.8.2 Texture Magnification **Up:** 3.8.1 Texture Minification **Previous:** 3.8.1 Texture Minification

---

David Blythe
Sat Mar 29 02:23:21 PST 1997

Progressive Meshes

Hugues Hoppe
Microsoft Research

ABSTRACT

Highly detailed geometric models are rapidly becoming commonplace in computer graphics. These models, often represented as complex triangle meshes, challenge rendering performance, transmission bandwidth, and storage capacities. This paper introduces the *progressive mesh* (PM) representation, a new scheme for storing and transmitting arbitrary triangle meshes. This efficient, lossless, continuous-resolution representation addresses several practical problems in graphics: smooth geomorphing of level-of-detail approximations, progressive transmission, mesh compression, and selective refinement.

In addition, we present a new mesh simplification procedure for constructing a PM representation from an arbitrary mesh. The goal of this optimization procedure is to preserve not just the geometry of the original mesh, but more importantly its overall appearance as defined by its discrete and scalar appearance attributes such as material identifiers, color values, normals, and texture coordinates. We demonstrate construction of the PM representation and its applications using several practical models.

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - surfaces and object representations.

Additional Keywords: mesh simplification, level of detail, shape interpolation, progressive transmission, geometry compression.

1 INTRODUCTION

Highly detailed geometric models are necessary to satisfy a growing expectation for realism in computer graphics. Within traditional modeling systems, detailed models are created by applying versatile modeling operations (such as extrusion, constructive solid geometry, and freeform deformations) to a vast array of geometric primitives. For efficient display, these models must usually be tessellated into polygonal approximations—meshes. Detailed meshes are also obtained by scanning physical objects using range scanning systems [5]. In either case, the resulting complex meshes are expensive to store, transmit, and render, thus motivating a number of practical problems:

- *Mesh simplification*: The meshes created by modeling and scanning systems are seldom optimized for rendering efficiency, and can frequently be replaced by nearly indistinguishable approximations with far fewer faces. At present, this process often requires significant user intervention. Mesh simplification tools can hope to automate this painstaking task, and permit the porting of a single model to platforms of varying performance.
- *Level-of-detail (LOD) approximation*: To further improve rendering performance, it is common to define several versions of a model at various levels of detail [3, 8]. A detailed mesh is used when the object is close to the viewer, and coarser approximations are substituted as the object recedes. Since instantaneous switching between LOD meshes may lead to perceptible "popping", one would like to construct smooth visual transitions, *geomorphs*, between meshes at different resolutions.
- *Progressive transmission*: When a mesh is transmitted over a communication line, one would like to show progressively better approximations to the model as data is incrementally received. One approach is to transmit successive LOD approximations, but this requires additional transmission time.
- *Mesh compression*: The problem of minimizing the storage space for a model can be addressed in two orthogonal ways. One is to use mesh simplification to reduce the number of faces. The other is mesh compression: minimizing the space taken to store a particular mesh.
- *Selective refinement*: Each mesh in a LOD representation captures the model at a uniform (view-independent) level of detail. Sometimes it is desirable to adapt the level of refinement in selected regions. For instance, as a user flies over a terrain, the terrain mesh need be fully detailed only near the viewer, and only within the field of view.

In addressing these problems, this paper makes two major contributions. First, it introduces the *progressive mesh* (PM) representation. In PM form, an arbitrary mesh $\hat{M}$ is stored as a much coarser mesh M^0 together with a sequence of n detail records that indicate how to incrementally refine M^0 exactly back into the original mesh $\hat{M} = M^n$. Each of these records stores the information associated with a *vertex split*, an elementary mesh transformation that adds an additional vertex to the mesh. The PM representation of $\hat{M}$ thus defines a continuous sequence of meshes $M^0, M^1, \ldots, M^n$ of increasing accuracy, from which LOD approximations of any desired complexity can be efficiently retrieved. Moreover, geomorphs can be efficiently constructed between any two such meshes. In addition, we show that the PM representation naturally supports progressive transmission, offers a concise encoding of $\hat{M}$ itself, and permits selective refinement. In short, progressive meshes offer an efficient, lossless, continuous-resolution representation.

The other contribution of this paper is a new simplification procedure for constructing a PM representation from a given mesh $\hat{M}$. Unlike previous simplification methods, our procedure seeks to preserve not just the geometry of the mesh surface, but more importantly its overall appearance, as defined by the discrete and scalar attributes associated with its surface.

Email: hhoppe@microsoft.com
Web: http://www.research.microsoft.com/research/graphics/hoppe/

2 MESHES IN COMPUTER GRAPHICS

Models in computer graphics are often represented using triangle meshes.[1] Geometrically, a triangle mesh is a piecewise linear surface consisting of triangular faces pasted together along their edges. As described in [9], the mesh geometry can be denoted by a tuple (K, V), where K is a *simplicial complex* specifying the connectivity of the mesh simplices (the adjacency of the vertices, edges, and faces), and $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ is the set of vertex positions defining the shape of the mesh in $\mathbf{R}^3$. More precisely (cf. [9]), we construct a parametric domain $|K| \subset \mathbf{R}^m$ by identifying each vertex of K with a canonical basis vector of $\mathbf{R}^m$, and define the mesh as the image $\phi_V(|K|)$ where $\phi_V : \mathbf{R}^m \to \mathbf{R}^3$ is a linear map.

Often, surface appearance attributes other than geometry are also associated with the mesh. These attributes can be categorized into two types: *discrete* attributes and *scalar* attributes.

Discrete attributes are usually associated with faces of the mesh. A common discrete attribute, the *material identifier*, determines the shader function used in rendering a face of the mesh [18]. For instance, a trivial shader function may involve simple look-up within a specified texture map.

Many scalar attributes are often associated with a mesh, including diffuse color (r, g, b), normal (n_x, n_y, n_z), and texture coordinates (u, v). More generally, these attributes specify the local parameters of shader functions defined on the mesh faces. In simple cases, these scalar attributes are associated with vertices of the mesh. However, to represent discontinuities in the scalar fields, and because adjacent faces may have different shading functions, it is common to associate scalar attributes not with vertices, but with corners of the mesh [1]. A *corner* is defined as a (vertex,face) tuple. Scalar attributes at a corner (v, f) specify the shading parameters for face f at vertex v. For example, along a *crease* (a curve on the surface across which the normal field is not continuous), each vertex has two distinct normals, one associated with the corners on each side of the crease.

We express a mesh as a tuple $M = (K, V, D, S)$ where V specifies its geometry, D is the set of discrete attributes d_f associated with the faces $f = \{j, k, l\} \in K$, and S is the set of scalar attributes $s_{(v,f)}$ associated with the corners (v, f) of K.

The attributes D and S give rise to discontinuities in the visual appearance of the mesh. An edge $\{v_j, v_k\}$ of the mesh is said to be *sharp* if either (1) it is a boundary edge, or (2) its two adjacent faces f_l and f_r have different discrete attributes (i.e. $d_{f_l} \neq d_{f_r}$), or (3) its adjacent corners have different scalar attributes (i.e. $s_{(v_j,f_l)} \neq s_{(v_j,f_r)}$ or $s_{(v_k,f_l)} \neq s_{(v_k,f_r)}$). Together, the set of sharp edges define a set of *discontinuity curves* over the mesh (e.g. the yellow curves in Figure 12).

3 PROGRESSIVE MESH REPRESENTATION

3.1 Overview

Hoppe et al. [9] describe a method, *mesh optimization*, that can be used to approximate an initial mesh $\hat{M}$ by a much simpler one. Their optimization algorithm, reviewed in Section 4.1, traverses the space of possible meshes by successively applying a set of 3 mesh transformations: edge collapse, edge split, and edge swap.

We have discovered that in fact a single one of those transformations, *edge collapse*, is sufficient for effectively simplifying meshes. As shown in Figure 1, an edge collapse transformation $ecol(\{v_s, v_t\})$ unifies 2 adjacent vertices v_s and v_t into a single vertex v_s. The vertex v_t and the two adjacent faces $\{v_s, v_t, v_l\}$ and $\{v_t, v_s, v_r\}$ vanish in the process. A position $\mathbf{v}_s$ is specified for the new unified vertex.

[1] We assume in this paper that more general meshes, such as those containing n-sided faces and faces with holes, are first converted into triangle meshes by triangulation. The PM representation could be generalized to handle the more general meshes directly, at the expense of more complex data structures.

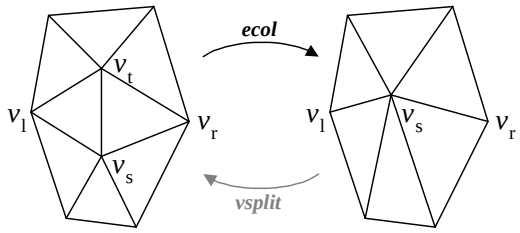


Figure 1: Illustration of the edge collapse transformation.

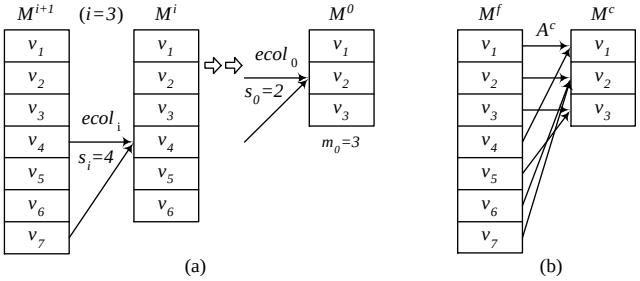


Figure 2: (a) Sequence of edge collapses; (b) Resulting vertex correspondence.

Thus, an initial mesh $\hat{M} = M^n$ can be simplified into a coarser mesh M^0 by applying a sequence of n successive edge collapse transformations:

$$(\hat{M} = M^n) \xrightarrow{ecol_{n-1}} \ldots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0 .$$

The particular sequence of edge collapse transformations must be chosen carefully, since it determines the quality of the approximating meshes $M^i, i < n$. A scheme for choosing these edge collapses is presented in Section 4.

Let m_0 be the number of vertices in M^0, and let us label the vertices of mesh M^i as $V^i = \{v_1, \ldots, v_{m_0+i}\}$, so that edge $\{v_{s_i}, v_{m_0+i+1}\}$ is collapsed by $ecol_i$ as shown in Figure 2a. As vertices may have different positions in the different meshes, we denote the position of v_j in M^i as $\mathbf{v}_j^i$.

A key observation is that an edge collapse transformation is invertible. Let us call that inverse transformation a *vertex split*, shown as *vsplit* in Figure 1. A vertex split transformation $vsplit(s, l, r, t, A)$ adds near vertex v_s a new vertex v_t and two new faces $\{v_s, v_t, v_l\}$ and $\{v_t, v_s, v_r\}$. (If the edge $\{v_s, v_t\}$ is a boundary edge, we let $v_r = 0$ and only one face is added.) The transformation also updates the attributes of the mesh in the neighborhood of the transformation. This attribute information, denoted by A, includes the positions $\mathbf{v}_s$ and $\mathbf{v}_t$ of the two affected vertices, the discrete attributes $d_{\{v_s,v_t,v_l\}}$ and $d_{\{v_t,v_s,v_r\}}$ of the two new faces, and the scalar attributes of the affected corners ($s_{(v_s,\cdot)}$, $s_{(v_t,\cdot)}$, $s_{(v_l,\{v_s,v_t,v_l\})}$, and $s_{(v_r,\{v_t,v_s,v_r\})}$).

Because edge collapse transformations are invertible, we can therefore represent an arbitrary triangle mesh $\hat{M}$ as a simple mesh M^0 together with a sequence of n *vsplit* records:

$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \ldots \xrightarrow{vsplit_{n-1}} (M^n = \hat{M})$$

where each record is parametrized as $vsplit_i(s_i, l_i, r_i, A_i)$. We call $(M^0, \{vsplit_0, \ldots, vsplit_{n-1}\})$ a *progressive mesh* (PM) representation of $\hat{M}$.

As an example, the mesh $\hat{M}$ of Figure 5d (13,546 faces) was simplified down to the coarse mesh M^0 of Figure 5a (150 faces) using

6,698 edge collapse transformations. Thus its PM representation consists of M^0 together with a sequence of $n = 6698$ *vsplit* records. From this PM representation, one can extract approximating meshes with any desired number of faces (actually, within ± 1) by applying to M^0 a prefix of the *vsplit* sequence. For example, Figure 5 shows approximating meshes with 150, 500, and 1000 faces.

3.2 Geomorphs

A nice property of the vertex split transformation (and its inverse, edge collapse) is that a smooth visual transition (a *geomorph*) can be created between the two meshes M^i and M^{i+1} in $M^i \xrightarrow{vsplit_i} M^{i+1}$. For the moment let us assume that the meshes contain no attributes other than vertex positions. With this assumption the vertex split record is encoded as $vsplit_i(s_i, l_i, r_i, A_i = (\mathbf{v}_{s_i}^{i+1}, \mathbf{v}_{m_0+i+1}^{i+1}))$. We construct a geomorph $M^G(\alpha)$ with blend parameter $0 \le \alpha \le 1$ such that $M^G(0)$ looks like M^i and $M^G(1)$ looks like M^{i+1}—in fact $M^G(1) = M^{i+1}$—by defining a mesh

$$M^G(\alpha) = (K^{i+1}, V^G(\alpha))$$

whose connectivity is that of M^{i+1} and whose vertex positions linearly interpolate from $v_{s_i} \in M^i$ to the split vertices $v_{s_i}, v_{m_0+i+1} \in M^{i+1}$:

$$\mathbf{v}_j^G(\alpha) = \begin{cases} (\alpha)\mathbf{v}_j^{i+1} + (1-\alpha)\mathbf{v}_{s_i}^i & , \; j \in \{s_i, m_0+i+1\} \\ \mathbf{v}_j^{i+1} = \mathbf{v}_j^i & , \; j \notin \{s_i, m_0+i+1\} \end{cases}$$

Using such geomorphs, an application can smoothly transition from a mesh M^i to meshes M^{i+1} or M^{i-1} without any visible "snapping" of the meshes.

Moreover, since individual *ecol* transformations can be transitioned smoothly, so can the composition of any sequence of them. Geomorphs can therefore be constructed between *any* two meshes of a PM representation. Indeed, given a finer mesh M^f and a coarser mesh M^c, $0 \le c < f \le n$, there exists a natural correspondence between their vertices: each vertex of M^f is related to a unique ancestor vertex of M^c by a surjective map A^c obtained by composing a sequence of *ecol* transformations (Figure 2b). More precisely, each vertex v_j of M^f corresponds with the vertex $v_{A^c(j)}$ in M^c where

$$A^c(j) = \begin{cases} j & , \; j \le m_0 + c \\ A^c(s_{j-m_0-1}) & , \; j > m_0 + c \end{cases} .$$

(In practice, this ancestor information A^c is gathered in a forward fashion as the mesh is refined.) This correspondence allows us to define a geomorph $M^G(\alpha)$ such that $M^G(0)$ looks like M^c and $M^G(1)$ equals M^f. We simply define $M^G(\alpha) = (K^f, V^G(\alpha))$ to have the connectivity of M^f and the vertex positions

$$\mathbf{v}_j^G(\alpha) = (\alpha)\mathbf{v}_j^f + (1-\alpha)\mathbf{v}_{A^c(j)}^c .$$

So far we have outlined the construction of geomorphs between PM meshes containing only position attributes. We can in fact construct geomorphs for meshes containing both discrete and scalar attributes.

Discrete attributes by their nature cannot be smoothly interpolated. Fortunately, these discrete attributes are associated with faces of the mesh, and the "geometric" geomorphs described above smoothly introduce faces. In particular, observe that the faces of M^c are a proper subset of the faces of M^f, and that those faces of M^f missing from M^c are invisible in $M^G(0)$ because they have been collapsed to degenerate (zero area) triangles. Other geomorphing schemes [10, 11, 17] define well-behaved (invertible) parametrizations between meshes at different levels of detail, but these do not permit the construction of geomorphs between meshes with different discrete attributes.

Scalar attributes defined on corners can be smoothly interpolated much like the vertex positions. There is a slight complication in that a corner (v, f) in a mesh M is not naturally associated with any "ancestor corner" in a coarser mesh M^c if f is not a face of M^c. We can still attempt to infer what attribute value (v, f) would have in M^c as follows. We examine the mesh M^{i+1} in which f is first introduced, locate a neighboring corner (v, f') in M^{i+1} whose attribute value is the same, and recursively backtrack from it to a corner in M^c. If there is no neighboring corner in M^{i+1} with an identical attribute value, then the corner (v, f) has no equivalent in M^c and we therefore keep its attribute value constant through the geomorph.

The interpolating function on the scalar attributes need not be linear; for instance, normals are best interpolated over the unit sphere, and colors may be interpolated in a color space other than RGB.

Figure 6 demonstrates a geomorph between two meshes M^{175} (500 faces) and M^{425} (1000 faces) retrieved from the PM representation of the mesh in Figure 5d.

3.3 Progressive transmission

Progressive meshes are a natural representation for progressive transmission. The compact mesh M^0 is transmitted first (using a conventional uni-resolution format), followed by the stream of $vsplit_i$ records. The receiving process incrementally rebuilds $\hat{M}$ as the records arrive, and animates the changing mesh. The changes to the mesh can be geomorphed to avoid visual discontinuities. The original mesh $\hat{M}$ is recovered exactly after all n records are received, since PM is a lossless representation.

The computation of the receiving process should be balanced between the reconstruction of $\hat{M}$ and interactive display. With a slow communication line, a simple strategy is to display the current mesh whenever the input buffer is found to be empty. With a fast communication line, we find that a good strategy is to display meshes whose complexities increase exponentially. (Similar issues arise in the display of images transmitted using progressive JPEG.)

3.4 Mesh compression

Even though the PM representation encodes both $\hat{M}$ and a continuous family of approximations, it is surprisingly space-efficient, for two reasons. First, the locations of the vertex split transformations can be encoded concisely. Instead of storing all three vertex indices (s_i, l_i, r_i) of $vsplit_i$, one need only store s_i and approximately 5 bits to select the remaining two vertices among those adjacent to v_{s_i}.[2] Second, because a vertex split has local effect, one can expect significant coherence in mesh attributes through each transformation. For instance, when vertex $v_{s_i}^i$ is split into $v_{s_i}^{i+1}$ and $v_{m_0+i+1}^{i+1}$, we can predict the positions $\mathbf{v}_{s_i}^{i+1}$ and $\mathbf{v}_{m_0+i+1}^{i+1}$ from $\mathbf{v}_{s_i}^i$, and use delta-encoding to reduce storage. Scalar attributes of corners in M^{i+1} can similarly be predicted from those in M^i. Finally, the material identifiers $d_{\{v_s, v_t, v_l\}}$ and $d_{\{v_t, v_s, v_r\}}$ of the new faces in mesh M^{i+1} can often be predicted from those of adjacent faces in M^i using only a few control bits.

As a result, the size of a carefully designed PM representation should be competitive with that obtained from methods for compressing uni-resolution meshes. Our current prototype implementation was not designed with this goal in mind. However, we analyze the compression of the connectivity K, and report results on the compression of the geometry V. In the following analysis, we assume for simplicity that $m_0 = 0$ since typically $m_0 \ll n$.

A common representation for the mesh connectivity K is to list the three vertex indices for each face. Since the number of vertices is n and the number of faces approximately $2n$, such a list requires $6\lceil \log_2(n) \rceil n$ bits of storage. Using a buffer of 2 vertices, *generalized triangle strip* representations reduce this number to about

[2] On average, v_{s_i} has 6 neighbors, and the number of permutations $P_2^6 = 30$ can be encoded in $\lceil \log_2(P_2^6) \rceil = 5$ bits.

$(\lceil \log_2(n) \rceil + 2k)n$ bits, where vertices are back-referenced once on average and $k \simeq 2$ bits capture the vertex replacement codes [6]. By increasing the vertex buffer size to 16, Deering's *generalized triangle mesh* representation [6] further reduces storage to about $(\frac{1}{8}\lceil \log_2(n) \rceil + 8)n$ bits. Turan [16] shows that planar graphs (and hence the connectivity of closed genus 0 meshes) can be encoded in $12n$ bits. Recent work by Taubin and Rossignac [15] addresses more general meshes. With the PM representation, each $vsplit_i$ requires specification of s_i and its two neighbors, for a total storage of about $(\lceil \log_2(n) \rceil + 5)n$ bits. Although not as concise as [6, 15], this is comparable to generalized triangle strips.

A traditional representation of the mesh geometry V requires storage of $3n$ coordinates, or $96n$ bits with IEEE single-precision floating point. Like Deering [6], we assume that these coordinates can be quantized to 16-bit fixed precision values without significant loss of visual quality, thus reducing storage to $48n$ bits. Deering is able to further compress this storage by delta-encoding the quantized coordinates and Huffman compressing the variable-length deltas. For 16-bit quantization, he reports storage of $35.8n$ bits, which includes both the deltas and the Huffman codes. Using a similar approach with the PM representation, we encode V in $31n$ to $50n$ bits as shown in Table 1. To obtain these results, we exploit a property of our optimization algorithm (Section 4.3): when considering the collapse of an edge $\{v_s, v_t\}$, it considers three starting points for the resulting vertex position $\mathbf{v}_n$: $\{\mathbf{v}_s, \mathbf{v}_t, \frac{\mathbf{v}_s+\mathbf{v}_t}{2}\}$. Depending on the starting point chosen, we delta-encode either $\{\mathbf{v}_s - \mathbf{v}_n, \mathbf{v}_t - \mathbf{v}_n\}$ or $\{\frac{\mathbf{v}_s+\mathbf{v}_t}{2} - \mathbf{v}_n, \frac{\mathbf{v}_t-\mathbf{v}_s}{2}\}$, and use separate Huffman tables for all four quantities.

To further improve compression, we could alter the construction algorithm to forego optimization and let $\mathbf{v}_n \in \{\mathbf{v}_s, \mathbf{v}_t, \frac{\mathbf{v}_s+\mathbf{v}_t}{2}\}$. This would degrade the accuracy of the approximating meshes somewhat, but allows encoding of V in $30n$ to $37n$ bits in our examples. Arithmetic coding [19] of delta lengths does not improve results significantly, reflecting the fact that the Huffman trees are well balanced. Further compression improvements may be achievable by adapting both the quantization level and the delta length models as functions of the *vsplit* record index i, since the magnitude of successive changes tends to decrease.

3.5 Selective refinement

The PM representation also supports selective refinement, whereby detail is added to the model only in desired areas. Let the application supply a callback function REFINE(v) that returns a Boolean value indicating whether the neighborhood of the mesh about v should be further refined. An initial mesh M^c is selectively refined by iterating through the list $\{vsplit_c, \ldots, vsplit_{n-1}\}$ as before, but only performing $vsplit_i(s_i, l_i, r_i, A_i)$ if

(1) all three vertices $\{v_{s_i}, v_{l_i}, v_{r_i}\}$ are present in the mesh, and

(2) REFINE(v_{s_i}) evaluates to TRUE.

(A vertex v_j is absent from the mesh if the prior vertex split that would have introduced it, $vsplit_{j-m_0-1}$, was not performed due to the above conditions.)

As an example, to obtain selective refinement of the model within a view frustum, REFINE(v) is defined to be TRUE if either v or any of its neighbors lies within the frustum. As seen in Figure 7a, condition (1) described above is suboptimal. The problem is that a vertex v_{s_i} within the frustum may fail to be split because its expected neighbor v_{l_i} lies just outside the frustum and was not previously created. The problem is remedied by using a less stringent version of condition (1). Let us define the *closest living ancestor* of a vertex v_j to be the vertex with index

$$A'(j) = \begin{cases} j & \text{, if } v_j \text{ exists in the mesh} \\ A'(s_{j-m_0-1}) & \text{, otherwise} \end{cases}$$

The new condition becomes:

(1') v_{s_i} is present in the mesh (i.e. $A'(s_i) = s_i$) and the vertices $v_{A'(l_i)}$ and $v_{A'(r_i)}$ are both adjacent to v_{s_i}.

As when constructing the geomorphs, the ancestor information A' is carried efficiently as the *vsplit* records are parsed. If conditions (1') and (2) are satisfied, $vsplit(s_i, A'(l_i), A'(r_i), A_i)$ is applied to the mesh. A mesh selectively refined with this new strategy is shown in Figure 7b. This same strategy was also used for Figure 10. Note that it is still possible to create geomorphs between M^c and selectively refined meshes thus created.

An interesting application of selective refinement is the transmission of view-dependent models over low-bandwidth communication lines. As the receiver's view changes over time, the sending process need only transmit those *vsplit* records for which REFINE evaluates to TRUE, and of those only the ones not previously transmitted.

4 PROGRESSIVE MESH CONSTRUCTION

The PM representation of an arbitrary mesh $\hat{M}$ requires a sequence of edge collapses transforming $\hat{M} = M^n$ into a base mesh M^0. The quality of the intermediate approximations $M^i, i < n$ depends largely on the algorithm for selecting which edges to collapse and what attributes to assign to the affected neighborhoods, for instance the positions $\mathbf{v}^i_{s_i}$.

There are many possible PM construction algorithms with varying trade-offs of speed and accuracy. At one extreme, a crude and fast scheme for selecting edge collapses is to choose them completely at random. (Some local conditions must be satisfied for an edge collapse to be legal, i.e. manifold preserving [9].) More sophisticated schemes can use heuristics to improve the edge selection strategy, for example the "distance to plane" metric of Schroeder et al. [14]. At the other extreme, one can attempt to find approximating meshes that are optimal with respect to some appearance metric, for instance the E_{dist} geometric metric of Hoppe et al. [9].

Since PM construction is a preprocess that can be performed offline, we chose to design a simplification procedure that invests some time in the selection of edge collapses. Our procedure is similar to the mesh optimization method introduced by Hoppe et al. [9], which is outlined briefly in Section 4.1. Section 4.2 presents an overview of our procedure, and Sections 4.3–4.6 present the details of our optimization scheme for preserving both the shape of the mesh and the scalar and discrete attributes which define its appearance.

4.1 Background: mesh optimization

The goal of mesh optimization [9] is to find a mesh $M = (K, V)$ that both accurately fits a set X of points $\mathbf{x}_i \in \mathbf{R}^3$ and has a small number of vertices. This problem is cast as minimization of an energy function

$$E(M) = E_{dist}(M) + E_{rep}(M) + E_{spring}(M) \ .$$

The first two terms correspond to the two goals of accuracy and conciseness: the *distance energy* term

$$E_{dist}(M) = \sum_i d^2(\mathbf{x}_i, \phi_V(|K|))$$

measures the total squared distance of the points from the mesh, and the *representation energy* term $E_{rep}(M) = c_{rep}m$ penalizes the number m of vertices in M. The third term, the *spring energy* $E_{spring}(M)$ is introduced to regularize the optimization problem. It corresponds to placing on each edge of the mesh a spring of rest length zero and tension κ:

$$E_{spring}(M) = \sum_{\{j,k\} \in K} \kappa \|\mathbf{v}_j - \mathbf{v}_k\|^2 \ .$$

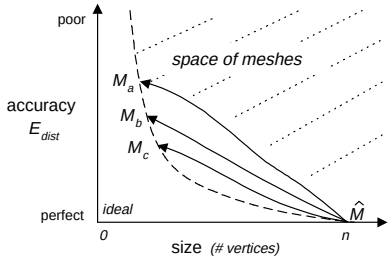


Figure 3: Illustration of the paths taken by mesh optimization using three different settings of c_{rep}.

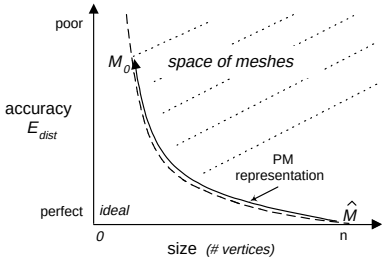


Figure 4: Illustration of the path taken by the new mesh simplification procedure in a graph plotting accuracy vs. mesh size.

The energy function $E(M)$ is minimized using a nested optimization method:

- *Outer loop*: The algorithm optimizes over K, the connectivity of the mesh, by randomly attempting a set of three possible mesh transformations: edge collapse, edge split, and edge swap. This set of transformations is complete, in the sense that any simplicial complex K of the same topological type as $\hat{K}$ can be reached through a sequence of these transformations. For each candidate mesh transformation, $K \rightarrow K'$, the continuous optimization described below computes $E_{K'}$, the minimum of E subject to the new connectivity K'. If $\Delta E = E_{K'} - E_K$ is found to be negative, the mesh transformation is applied (akin to a zero-temperature simulated annealing method).
- *Inner loop*: For each candidate mesh transformation, the algorithm computes $E_{K'} = \min_V E_{dist}(V) + E_{spring}(V)$ by optimizing over the vertex positions V. For the sake of efficiency, the algorithm in fact optimizes only one vertex position $\mathbf{v}_s$, and considers only the subset of points X that project onto the neighborhood affected by $K \rightarrow K'$. To avoid surface self-intersections, the edge collapse is disallowed if the maximum dihedral angle of edges in the resulting neighborhood exceeds some threshold.

Hoppe et al. [9] find that the regularizing spring energy term $E_{spring}(M)$ is most important in the early stages of the optimization, and achieve best results by repeatedly invoking the nested optimization method described above with a schedule of decreasing spring constants κ.

Mesh optimization is demonstrated to be an effective tool for mesh simplification. Given an initial mesh $\hat{M}$ to approximate, a dense set of points X is sampled both at the vertices of $\hat{M}$ and randomly over its faces. The optimization algorithm is then invoked with $\hat{M}$ as the starting mesh. Varying the setting of the representation constant c_{rep} results in optimized meshes with different trade-offs of accuracy and size. The paths taken by these optimizations are shown illustratively in Figure 3.

4.2 Overview of the simplification algorithm

As in mesh optimization [9], we also define an explicit energy metric $E(M)$ to measure the accuracy of simplified meshes $M = (K, V, D, S)$ with respect to the original $\hat{M}$, and we also modify the mesh M starting from $\hat{M}$ while minimizing $E(M)$.

Our energy metric has the following form:

$$E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M) \ .$$

The first two terms, $E_{dist}(M)$ and $E_{spring}(M)$ are identical to those in [9]. The next two terms of $E(M)$ are added to preserve attributes associated with M: $E_{scalar}(M)$ measures the accuracy of its scalar attributes (Section 4.4), and $E_{disc}(M)$ measures the geometric accuracy of its discontinuity curves (Section 4.5). (To achieve scale invariance of the terms, the mesh is uniformly scaled to fit in a unit cube.)

Our scheme for optimizing over the connectivity K of the mesh is rather different from [9]. We have discovered that a mesh can be effectively simplified using edge collapse transformations alone. The edge swap and edge split transformations, useful in the context of surface reconstruction (which motivated [9]), are not essential for simplification. Although in principle our simplification algorithm can no longer traverse the entire space of meshes considered by mesh optimization, we find that the meshes generated by our algorithm are just as good. In fact, because of the priority queue approach described below, our meshes are usually better. Moreover, considering only edge collapses simplifies the implementation, improves performance, and most importantly, gives rise to the PM representation (Section 3).

Rather than randomly attempting mesh transformations as in [9], we place all (legal) candidate edge collapse transformations into a priority queue, where the priority of each transformation is its estimated energy cost ΔE. In each iteration, we perform the transformation at the front of the priority queue (with lowest ΔE), and recompute the priorities of edges in the neighborhood of this transformation. As a consequence, we eliminate the need for the awkward parameter c_{rep} as well as the energy term $E_{rep}(M)$. Instead, we can explicitly specify the number of faces desired in an optimized mesh. Also, a single run of the optimization can generate several such meshes. Indeed, it generates a continuous-resolution family of meshes, namely the PM representation of $\hat{M}$ (Figure 4).

For each edge collapse $K \rightarrow K'$, we compute its cost $\Delta E = E_{K'} - E_K$ by solving a continuous optimization

$$E_{K'} = \min_{V,S} E_{dist}(V) + E_{spring}(V) + E_{scalar}(V, S) + E_{disc}(V)$$

over both the vertex positions V and the scalar attributes S of the mesh with connectivity K'. This minimization is discussed in the next three sections.

4.3 Preserving surface geometry ($E_{dist} + E_{spring}$)

As in [9], we "record" the geometry of the original mesh $\hat{M}$ by sampling from it a set of points X. At a minimum, we sample a point at each vertex of $\hat{M}$. If requested by the user, additional points are sampled randomly over the surface of $\hat{M}$. The energy terms $E_{dist}(M)$ and $E_{spring}(M)$ are defined as in Section 4.1.

For a mesh of fixed connectivity, our method for optimizing the vertex positions to minimize $E_{dist}(V) + E_{spring}(V)$ closely follows that of [9]. Evaluating $E_{dist}(V)$ involves computing the distance of each point $\mathbf{x}_i$ to the mesh. Each of these distances is itself a minimization problem

$$d^2(\mathbf{x}_i, \phi_V(|K|)) = \min_{\mathbf{b}_i \in |K|} \|\mathbf{x}_i - \phi_V(\mathbf{b}_i)\|^2 \qquad (1)$$

where the unknown $\mathbf{b}_i$ is the parametrization of the projection of $\mathbf{x}_i$ on the mesh. The nonlinear minimization of $E_{dist}(V) + E_{spring}(V)$ is performed using an iterative procedure alternating between two steps:

1. For fixed vertex positions V, compute the optimal parametrizations $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_{|X|}\}$ by projecting the points X onto the mesh.
2. For fixed parametrizations B, compute the optimal vertex positions V by solving a sparse linear least-squares problem.

As in [9], when considering $ecol(\{v_s, v_t\})$, we optimize only one vertex position, $\mathbf{v}_s^i$. We perform three different optimizations with different starting points, $\mathbf{v}_s^i = (1-\alpha)\mathbf{v}_s^{i+1} + (\alpha)\mathbf{v}_t^{i+1}$ for $\alpha = \{0, \frac{1}{2}, 1\}$, and accept the best one.

Instead of defining a global spring constant κ for E_{spring} as in [9], we adapt κ each time an edge collapse transformation is considered. Intuitively, the spring energy is most important when few points project onto a neighborhood of faces, since in this case finding the vertex positions minimizing $E_{dist}(V)$ may be an under-constrained problem. Thus, for each edge collapse transformation considered, we set κ as a function of the ratio of the number of points to the number of faces in the neighborhood.[3] With this adaptive scheme, the influence of $E_{spring}(M)$ decreases gradually and adaptively as the mesh is simplified, and we no longer require the expensive schedule of decreasing spring constants.

4.4 Preserving scalar attributes (E_{scalar})

As described in Section 2, we represent piecewise continuous scalar fields by defining scalar attributes S at the mesh corners. We now present our scheme for preserving these scalar fields through the simplification process. For exposition, we find it easier to first present the case of continuous scalar fields, in which the corner attributes at a vertex are identical. The generalization to piecewise continuous fields is discussed shortly.

Optimizing scalar attributes at vertices Let the original mesh $\hat{M}$ have at each vertex v_j not only a position $\mathbf{v}_j \in \mathbf{R}^3$ but also a scalar attribute $\underline{\mathbf{v}}_j \in \mathbf{R}^d$. To capture scalar attributes, we sample at each point $\mathbf{x}_i \in X$ the attribute value $\underline{\mathbf{x}}_i \in \mathbf{R}^d$. We would then like to generalize the distance metric E_{dist} to also measure the deviation of the sampled attribute values $\underline{X}$ from those of M.

One natural way to achieve this is to redefine the distance metric to measure distance in $\mathbf{R}^{3+d}$:

$$d^2((\mathbf{x}_i \;\; \underline{\mathbf{x}}_i), M(K, V, \underline{V})) = \min_{\mathbf{b}_i \in |K|} \left\| (\mathbf{x}_i \;\; \underline{\mathbf{x}}_i) - (\phi_V(\mathbf{b}_i) \;\; \phi_{\underline{V}}(\mathbf{b}_i)) \right\|^2 .$$

This new distance functional could be minimized using the iterative approach of Section 4.3. However, it would be expensive since finding the optimal parametrization $\mathbf{b}_i$ of each point $\mathbf{x}_i$ would require projection in $\mathbf{R}^{3+d}$, and would be non-intuitive since these parametrizations would not be geometrically based.

Instead we opted to determine the parametrizations $\mathbf{b}_i$ using only geometry with equation (1), and to introduce a separate energy term E_{scalar} to measure attribute deviation based on these parametrizations:

$$E_{scalar}(\underline{V}) = (c_{scalar})^2 \sum_i \left\| \underline{\mathbf{x}}_i - \phi_{\underline{V}}(\mathbf{b}_i) \right\|^2$$

where the constant c_{scalar} assigns a relative weight between the scalar attribute errors (E_{scalar}) and the geometric errors (E_{dist}).

Thus, to minimize $E(V, \underline{V}) = E_{dist}(V) + E_{spring}(V) + E_{scalar}(\underline{V})$, our algorithm first finds the vertex position $\mathbf{v}_s$ minimizing $E_{dist}(V) + E_{spring}(V)$ by alternately projecting the points onto the mesh (obtaining the parametrizations $\mathbf{b}_i$) and solving a linear least-squares problem (Section 4.1). Then, using those same parametrizations $\mathbf{b}_i$, it finds the vertex attribute $\underline{\mathbf{v}}_s$ minimizing E_{scalar} by solving a single linear least-squares problem. Hence introducing E_{scalar} into the optimization causes negligible performance overhead.

Since ΔE_{scalar} contributes to the estimated cost ΔE of an edge collapse, we obtain simplified meshes whose faces naturally adapt to the attribute fields, as shown in Figures 8 and 11.

Optimizing scalar attributes at corners Our scheme for optimizing the scalar corner attributes S is a straightforward generalization of the scheme just described. Instead of solving for a single unknown attribute value $\underline{\mathbf{v}}_s$, the algorithm partitions the corners around v_s into continuous sets (based on equivalence of corner attributes) and for each continuous set solves independently for its optimal attribute value.

Range constraints Some scalar attributes have constrained ranges. For instance, the components (r, g, b) of color are typically constrained to lie between 0 and 1. Least-squares optimization may yield color values outside this range. In these cases we clip the optimized values to the given range. For least-squares minimization of a Euclidean norm at a single vertex, this is in fact optimal.

Normals Surface normals (n_x, n_y, n_z) are typically constrained to have unit length, and thus their domain is non-Cartesian. Optimizing over normals would therefore require minimization of a nonlinear functional with nonlinear constraints. We decided to instead simply carry the normals through the simplification process. Specifically, we compute the new normals at vertex $v_{s_i}^i$ by interpolating between the normals at vertices $v_{s_i}^{i+1}$ and $v_{m_0+i+1}^{i+1}$ using the α value that resulted in the best vertex position $\mathbf{v}_{s_i}^i$ in Section 4.3. Fortunately, the absolute directions of normals are less visually important than their discontinuities, and we have a scheme for preserving such discontinuities, as described in the next section.

4.5 Preserving discontinuity curves (E_{disc})

Appearance attributes give rise to a set of discontinuity curves on the mesh, both from differences between discrete face attributes (e.g. material boundaries), and from differences between scalar corner attributes (e.g. creases and shadow boundaries). As these discontinuity curves form noticeable features, we have found it useful to preserve them both topologically and geometrically.

We can detect when a candidate edge collapse would modify the topology of the discontinuity curves using some simple tests on the presence of sharp edges in its neighborhood. Let $sharp(v_j, v_k)$ denote that an edge $\{v_j, v_k\}$ is sharp, and let $\#sharp(v_j)$ be the number of sharp edges adjacent to a vertex v_j. Then, referring to Figure 1, $ecol(\{v_s, v_t\})$ modifies the topology of discontinuity curves if either:

- $sharp(v_s, v_l)$ and $sharp(v_t, v_l)$, or
- $sharp(v_s, v_r)$ and $sharp(v_t, v_r)$, or
- $\#sharp(v_s) \geq 1$ and $\#sharp(v_t) \geq 1$ and not $sharp(v_s, v_t)$, or
- $\#sharp(v_s) \geq 3$ and $\#sharp(v_t) \geq 3$ and $sharp(v_s, v_t)$, or
- $sharp(v_s, v_t)$ and $\#sharp(v_s) = 1$ and $\#sharp(v_t) \neq 2$, or
- $sharp(v_s, v_t)$ and $\#sharp(v_t) = 1$ and $\#sharp(v_s) \neq 2$.

If an edge collapse would modify the topology of discontinuity curves, we either disallow it, or penalize it as discussed in Section 4.6.

To preserve the geometry of the discontinuity curves, we sample an additional set of points X_{disc} from the sharp edges of $\hat{M}$, and define an additional energy term E_{disc} equal to the total squared distances of each of these points to the discontinuity curve from which it was sampled. Thus E_{disc} is defined just like E_{dist}, except that the points X_{disc} are constrained to project onto a set of sharp edges in the mesh. In effect, we are solving a curve fitting problem embedded within the surface fitting problem. Since all boundaries of the surface are defined to be discontinuity curves, our procedure preserves bound-

[3] The neighborhood of an edge collapse transformation is the set of faces shown in Figure 1. Using C notation, we set $\kappa = r < 4 \;?\; 10^{-2} : r < 8 \;?\; 10^{-4} : 10^{-8}$ where r is the ratio of the number of points to faces in the neighborhood.

ary geometry more accurately than [9]. Figure 9 demonstrates the importance of using the E_{disc} energy term in preserving the material boundaries of a mesh with discrete face attributes.

4.6 Permitting changes to topology of discontinuity curves

Some meshes contain numerous discontinuity curves, and these curves may delimit features that are too small to be visible when viewed from a distance. In such cases we have found that strictly preserving the topology of the discontinuity curves unnecessarily curtails simplification. We have therefore adopted a hybrid strategy, which is to permit changes to the topology of the discontinuity curves, but to penalize such changes. When a candidate edge collapse $ecol(\{v_s, v_t\})$ changes the topology of the discontinuity curves, we add to its cost ΔE the value $|X_{disc,\{v_s,v_t\}}| \cdot \|\mathbf{v}_s - \mathbf{v}_t\|^2$ where $|X_{disc,\{v_s,v_t\}}|$ is the number of points of X_{disc} projecting onto $\{v_s, v_t\}$. That simple strategy, although ad hoc, has proven very effective. For example, it allows the dark gray window frames of the "cessna" (visible in Figure 9) to vanish in the simplified meshes (Figures 5a–c).

Table 1: Parameter settings and quantitative results.

Object	Original $\hat{M}$		Base M^0		User param.		$\lvert X_{disc}\rvert$	V $\frac{\text{bits}}{n}$	Time
	$m_0 + n$	#faces	m_0	#faces	$\lvert X\rvert-(m_0+n)$	c_{color}			mins
cessna	6,795	13,546	97	150	100,000	-	46,811	46	23
terrain	33,847	66,960	3	1	0	-	3,796	46	16
mandrill	40,000	79,202	3	1	0	0.1	4,776	31	19
radiosity	78,923	150,983	1,192	1,191	200,000	0.01	74,316	37	106
fandisk	6,475	12,946	27	50	10,000	-	5,924	50	19

5 RESULTS

Table 1 shows, for the meshes in Figures 5–12, the number of vertices and faces in both $\hat{M}$ and M^0. In general, we let the simplification proceed until no more legal edge collapse transformations are possible. For the "cessna", we stopped at 150 faces to obtain a visually aesthetic base mesh. As indicated, the only user-specified parameters are the number of additional points (besides the $m_0 + n$ vertices of $\hat{M}$) sampled to increase fidelity, and the c_{scalar} constants relating the scalar attribute accuracies to the geometric accuracy. The only scalar attribute we optimized is color, and its c_{scalar} constant is denoted as c_{color}. The number $|X_{disc}|$ of points sampled from sharp edges is set automatically so that the densities of X and X_{disc} are proportional.[4] Execution times were obtained on a 150MHz Indigo2 with 128MB of memory.

Construction of the PM representation proceeds in three steps. First, as the simplification algorithm applies a sequence $ecol_{n-1} \ldots ecol_0$ of transformations to the original mesh, it writes to a file the sequence $vsplit_{n-1} \ldots vsplit_0$ of corresponding inverse transformations. When finished, the algorithm also writes the resulting base mesh M^0. Next, we reverse the order of the *vsplit* records. Finally, we renumber the vertices and faces of $(M^0, vsplit_0 \ldots vsplit_{n-1})$ to match the indexing scheme of Section 3.1 in order to obtain a concise format.

Figure 6 shows a single geomorph between two meshes M^{175} and M^{425} of a PM representation. For interactive LOD, it is useful to select a sequence of meshes from the PM representation, and to construct successive geomorphs between them. We have obtained good results by selecting meshes whose complexities grow exponentially, as in Figure 5. During execution, an application can adjust the granularity of these geomorphs by sampling additional meshes from the PM representation, or freeing some up.

In Figure 10, we selectively refined a terrain (grid of 181×187 vertices) using a new REFINE(v) function that keeps more detail near silhouette edges and near the viewer. More precisely, for the faces F_v adjacent to v, we compute the signed projected screen areas $\{a_f : f \in F_v\}$. We let REFINE(v) return TRUE if

(1) any face $f \in F_v$ lies within the view frustum, and either

(2a) the signs of a_f are not all equal (i.e. v lies near a silhouette edge) or

(2b) $\sum_{f \in F_v} a_f > thresh$ for a screen area threshold $thresh = 0.16^2$ (where total screen area is 1).

6 RELATED WORK

Mesh simplification methods A number of schemes construct a discrete sequence of approximating meshes by repeated application of a simplification procedure. Turk [17] sprinkles a set of points on a mesh, with density weighted by estimates of local curvature, and then retriangulates based on those points. Both Schroeder et al. [14] and Cohen et al. [4] iteratively remove vertices from the mesh and retriangulate the resulting holes. Cohen et al. are able to bound the maximum error of the approximation by restricting it to lie between two offset surfaces. Hoppe et al. [9] find accurate approximations through a general mesh optimization process (Section 4.1). Rossignac and Borrel [12] merge vertices of a model using spatial binning. A unique aspect of their approach is that the topological type of the model may change in the process. Their method is extremely fast, but since it ignores geometric qualities like curvature, the resulting approximations can be far from optimal. Some of the above methods [12, 17] permit the construction of geomorphs between successive simplified meshes.

Multiresolution analysis (MRA) Lounsbery et al. [10, 11] generalize the concept of multiresolution analysis to surfaces of arbitrary topological type. Eck et al. [7] describe how MRA can be applied to the approximation of an arbitrary mesh. Certain et al. [2] extend MRA to capture color, and present a multiresolution Web viewer supporting progressive transmission. MRA has many similarities with the PM scheme, since both store a simple base mesh together with a stream of detail records, and both produce a continuous-resolution representation. It is therefore worthwhile to highlight their differences:

Advantages of PM over MRA:

- MRA requires that the detail terms (wavelets) lie on a domain with subdivision connectivity, and as a result an arbitrary initial mesh $\hat{M}$ can only be recovered to within an ϵ tolerance. In contrast, the PM representation is lossless since $M^n = \hat{M}$.
- Because the approximating meshes $M^i, i < n$ in a PM may have arbitrary connectivity, they can be much better approximations than their MRA counterparts (Figure 12).
- The MRA representation cannot deal effectively with surface creases, unless those creases lie parametrically along edges of the base mesh (Figure 12). PM's can introduce surface creases anywhere and at any level of detail.
- PM's capture continuous, piecewise-continuous, and discrete appearance attributes. MRA schemes can represent discontinuous functions using a piecewise-constant basis (such as the Haar basis as used in [2, 13]), but the resulting approximations have too many discontinuities since none of the basis functions meet continuously. Also, it is not clear how MRA could be extended to capture discrete attributes.

[4] We set $|X_{disc}|$ such that $|X_{disc}|/perim = c(|X|/area)^{\frac{1}{2}}$ where $perim$ is the total length of all sharp edges in $\hat{M}$, $area$ is total area of all faces, and the constant $c = 4.0$ is chosen empirically.

Advantages of MRA over PM:

- The MRA framework provides a parametrization between meshes at various levels of detail, thus making possible multiresolution surface editing. PM's also offer such a parametrization, but it is not smooth, and therefore multiresolution editing may be non-intuitive.
- Eck et al. [7] construct MRA approximations with guaranteed maximum error bounds to $\hat{M}$. Our PM construction algorithm does not provide such bounds, although one could envision using simplification envelopes [4] to achieve this.
- MRA allows geometry and color to be compressed independently [2].

Other related work There has been relatively little work in simplifying arbitrary surfaces with functions defined over them. One special instance is image compression, since an image can be thought of as a set of scalar color functions defined on a quadrilateral surface. Another instance is the framework of Schröder and Sweldens [13] for simplifying functions defined over the sphere. The PM representation, like the MRA representation, is a generalization in that it supports surfaces of arbitrary topological type.

7 SUMMARY AND FUTURE WORK

We have introduced the progressive mesh representation and shown that it naturally supports geomorphs, progressive transmission, compression, and selective refinement. In addition, as a PM construction method, we have presented a new mesh simplification procedure designed to preserve not just the geometry of the original mesh, but also its overall appearance.

There are a number of avenues for future work, including:

- Development of an explicit metric and optimization scheme for preserving surface normals.
- Experimentation with PM editing.
- Representation of articulated or animated models.
- Application of the work to progressive subdivision surfaces.
- Progressive representation of more general simplicial complexes (not just 2-d manifolds).
- Addition of spatial data structures to permit efficient selective refinement.

We envision many practical applications for the PM representation, including streaming of 3D geometry over the Web, efficient storage formats, and continuous LOD in computer graphics applications. The representation may also have applications in finite element methods, as it can be used to generate coarse meshes for multigrid analysis.

ACKNOWLEDGMENTS

I wish to thank Viewpoint Datalabs for providing the "cessna" mesh, Pratt & Whitney for the gas turbine engine component ("fandisk"), Softimage for the "terrain" mesh, and especially Steve Drucker for creating several radiosity models using Lightscape. Thanks also to Michael Cohen, Steven "Shlomo" Gortler, and Jim Kajiya for their enthusiastic support, and to Rick Szeliski for helpful comments on the paper. Mark Kenworthy first coined the term "geomorph" in '92 to distinguish them from image morphs.

REFERENCES

[1] APPLE COMPUTER, INC. *3D graphics programming with QuickDraw 3D.* Addison Wesley, 1995.

[2] CERTAIN, A., POPOVIC, J., DUCHAMP, T., SALESIN, D., STUETZLE, W., AND DEROSE, T. Interactive multiresolution surface viewing. *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996).

[3] CLARK, J. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM 19,* 10 (Oct. 1976), 547–554.

[4] COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, F., AND WRIGHT, W. Simplification envelopes. *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996).

[5] CURLESS, B., AND LEVOY, M. A volumetric method for building complex models from range images. *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996).

[6] DEERING, M. Geometry compression. *Computer Graphics (SIGGRAPH '95 Proceedings)* (1995), 13–20.

[7] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution analysis of arbitrary meshes. *Computer Graphics (SIGGRAPH '95 Proceedings)* (1995), 173–182.

[8] FUNKHOUSER, T., AND SÉQUIN, C. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (SIGGRAPH '93 Proceedings)* (1995), 247–254.

[9] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh optimization. *Computer Graphics (SIGGRAPH '93 Proceedings)* (1993), 19–26.

[10] LOUNSBERY, J. M. *Multiresolution analysis for surfaces of arbitrary topological type.* PhD thesis, Dept. of Computer Science and Engineering, U. of Washington, 1994.

[11] LOUNSBERY, M., DEROSE, T., AND WARREN, J. Multiresolution analysis for surfaces of arbitrary topological type. Submitted for publication. (TR 93-10-05b, Dept. of Computer Science and Engineering, U. of Washington, January 1994.).

[12] ROSSIGNAC, J., AND BORREL, P. Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*, B. Falcidieno and T. L. Kunii, Eds. Springer-Verlag, 1993, pp. 455–465.

[13] SCHRÖDER, P., AND SWELDENS, W. Spherical wavelets: Efficiently representing functions on the sphere. *Computer Graphics (SIGGRAPH '95 Proceedings)* (1995), 161–172.

[14] SCHROEDER, W., ZARGE, J., AND LORENSEN, W. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proceedings) 26,* 2 (1992), 65–70.

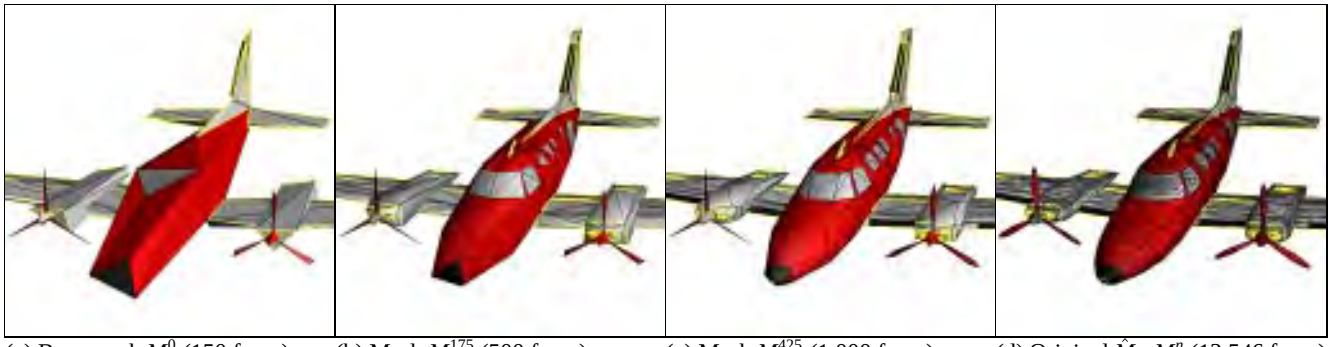
[15] TAUBIN, G., AND ROSSIGNAC, J. Geometry compression through topological surgery. Research Report RC-20340, IBM, January 1996.

[16] TURAN, G. Succinct representations of graphs. *Discrete Applied Mathematics 8* (1984), 289–294.

[17] TURK, G. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings) 26,* 2 (1992), 55–64.

[18] UPSTILL, S. *The RenderMan Companion*. Addison-Wesley, 1990.

[19] WITTEN, I., NEAL, R., AND CLEARY, J. Arithmetic coding for data compression. *Communications of the ACM 30,* 6 (June 1987), 520–540.

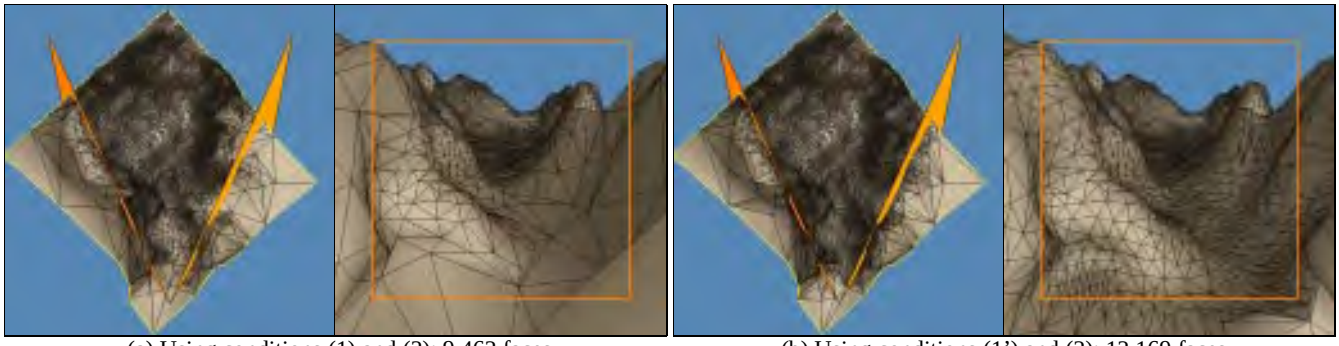


(a) Base mesh M^0 (150 faces) (b) Mesh M^{175} (500 faces) (c) Mesh M^{425} (1,000 faces) (d) Original $\hat{M} = M^n$ (13,546 faces)

Figure 5: The PM representation of an arbitrary mesh $\hat{M}$ captures a continuous-resolution family of approximating meshes $M^0 \ldots M^n = \hat{M}$.

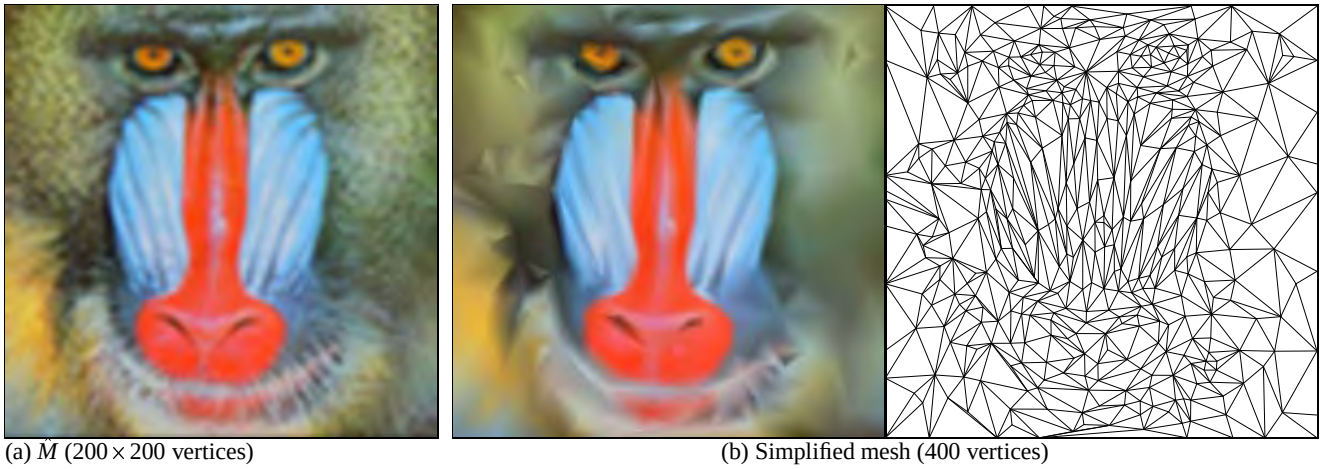
(a) $\alpha = 0.00$ (b) $\alpha = 0.25$ (c) $\alpha = 0.50$ (d) $\alpha = 0.75$ (e) $\alpha = 1.00$

Figure 6: Example of a geomorph $M^G(\alpha)$ defined between $M^G(0) \doteq M^{175}$ (with 500 faces) and $M^G(1) = M^{425}$ (with 1,000 faces).



(a) Using conditions (1) and (2); 9,462 faces (b) Using conditions (1') and (2); 12,169 faces

Figure 7: Example of selective refinement within the view frustum (indicated in orange).



(a) $\hat{M}$ (200×200 vertices) (b) Simplified mesh (400 vertices)

Figure 8: Demonstration of minimizing E_{scalar}: simplification of a mesh with trivial geometry (a square) but complex scalar attribute field. ($\hat{M}$ is a mesh with regular connectivity whose vertex colors correspond to the pixels of an image.)

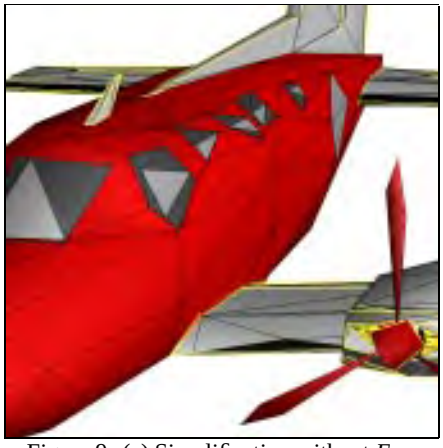


Figure 9: (a) Simplification without E_{disc}

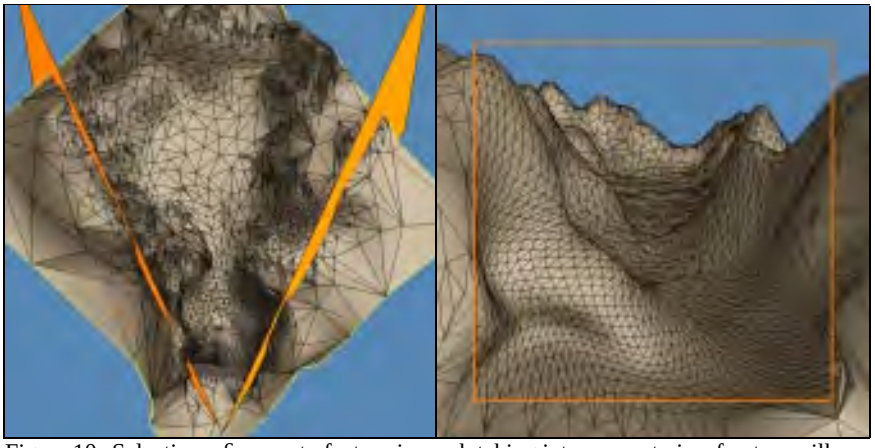
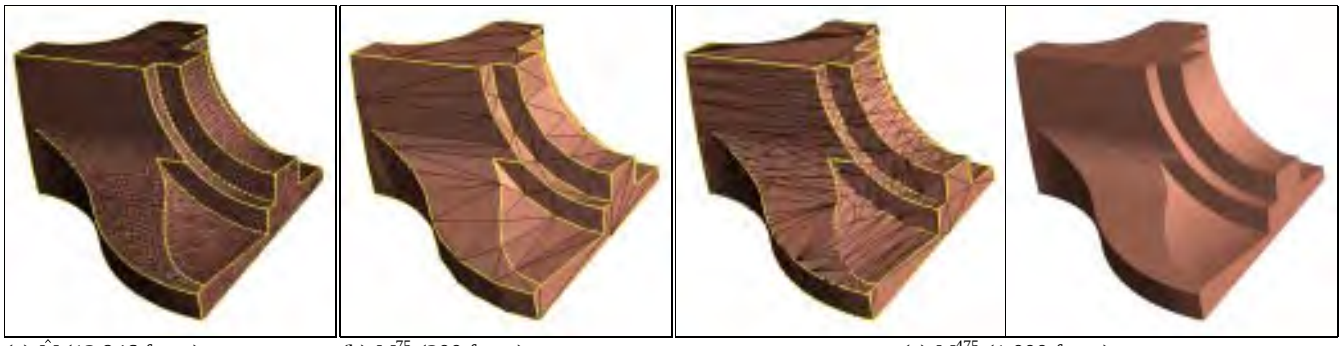
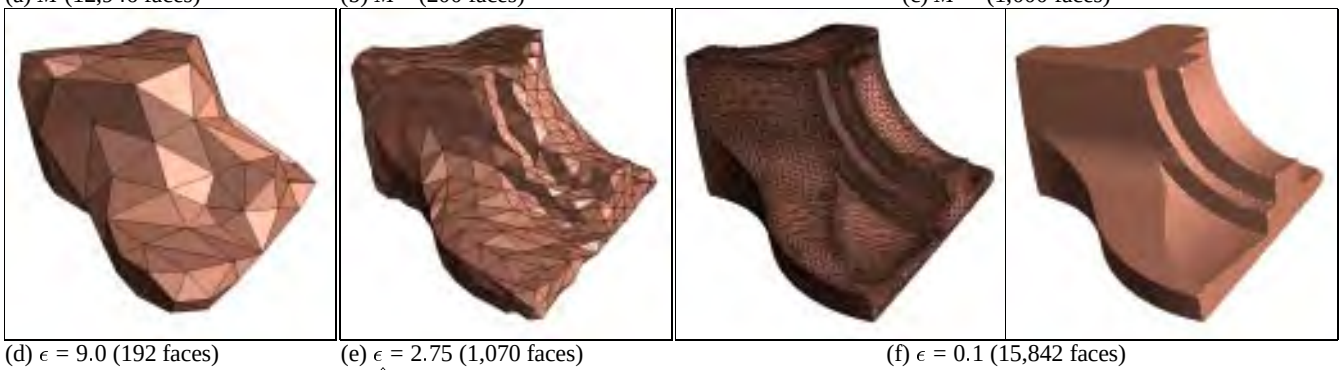


Figure 10: Selective refinement of a terrain mesh taking into account view frustum, silhouette regions, and projected screen size of faces (7,438 faces).

Figure 11: Simplification of a radiosity solution; left: original mesh (150,983 faces); right: simplified mesh (10,000 faces).



(a) $\hat{M}$ (12,946 faces) (b) M^{75} (200 faces) (c) M^{475} (1,000 faces)



(d) ϵ = 9.0 (192 faces) (e) ϵ = 2.75 (1,070 faces) (f) ϵ = 0.1 (15,842 faces)

Figure 12: Approximations of a mesh $\hat{M}$ using (b–c) the PM representation, and (d–f) the MRA scheme of Eck et al. [7]. As demonstrated, MRA cannot recover $\hat{M}$ exactly, cannot deal effectively with surface creases, and produces approximating meshes of inferior quality.

US005798770A

United States Patent [19]

Baldwin

[11] Patent Number: 5,798,770

[45] Date of Patent: Aug. 25, 1998

[54] **GRAPHICS RENDERING SYSTEM WITH RECONFIGURABLE PIPELINE SEQUENCE**

[75] Inventor: **David Robert Baldwin**, Weybridge, United Kingdom

[73] Assignee: **3DLabs Inc. Ltd.**, Hamilton, Bermuda

[21] Appl. No.: **640,620**

[22] Filed: **May 1, 1996**

Related U.S. Application Data

[60] Provisional application No. 60/008,803 Dec. 18, 1995.

[63] Continuation-in-part of Ser. No. 410,345, Mar. 24, 1995.

[51] **Int. Cl.**[6] **G06T 1/20**

[52] **U.S. Cl.** **345/506**; 345/519; 345/509

[58] **Field of Search** 395/506, 502, 395/507, 509, 519, 122, 130, 132, 125, 503; 345/506, 507, 502, 509, 519, 422, 430–432, 425, 503

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,866,637	9/1989	Gonzalez-Lopez	395/506
5,392,391	2/1995	Caulk, Jr. et al.	395/503

OTHER PUBLICATIONS

Foley *et al.*, "Computer Graphics, Principles and Practice", 2 ed in C,1996, Chapter 18, pp. 855–920.

Kogge, P.M., "The Microprogramming of Pipelined Processors", 1977, Proc. 4th Ann. Conf Parallel Procesing, IEEE, March, pp. 63–69.

Computer Graphics, vol. 22, No. 4, "A display system for the Stellar graphics Supercomputer Model GS1000". Brian Apgar et al., Aug. 1988.

Primary Examiner—Kee M. Tung

Attorney, Agent, or Firm—Robert Groover; Betty Formby; Matthew S. Anderson

[57] **ABSTRACT**

The preferred embodiment discloses a pipelined graphics processor in which the sequence can be dynamically reconfigured (e.g. between primitives) in a rendering sequence. The pipeline sequence can be configured for compliance with specifications such as OpenGL, but may also be optimized by reconfiguring the pipeline sequence to eliminate unnecessary processing. In a preferred embodiment, pixel elimination sequences such as depth and stencil tests are performed before texturing calculations are performed, so that unneeded pixel data is discarded before said texturing calculations are performed.

26 Claims, 12 Drawing Sheets

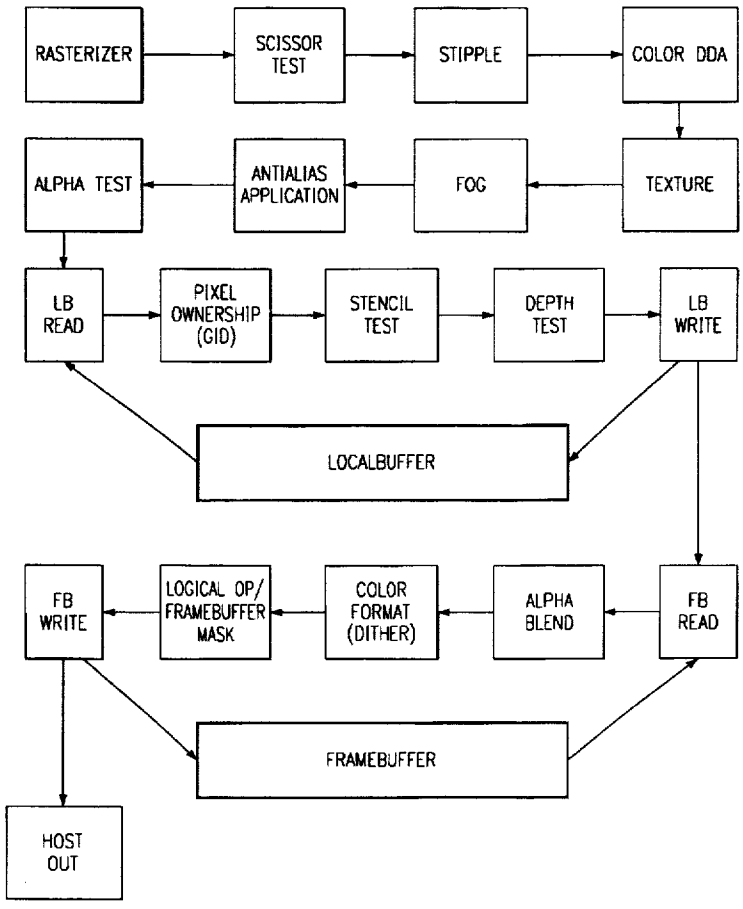


FIG. 1A

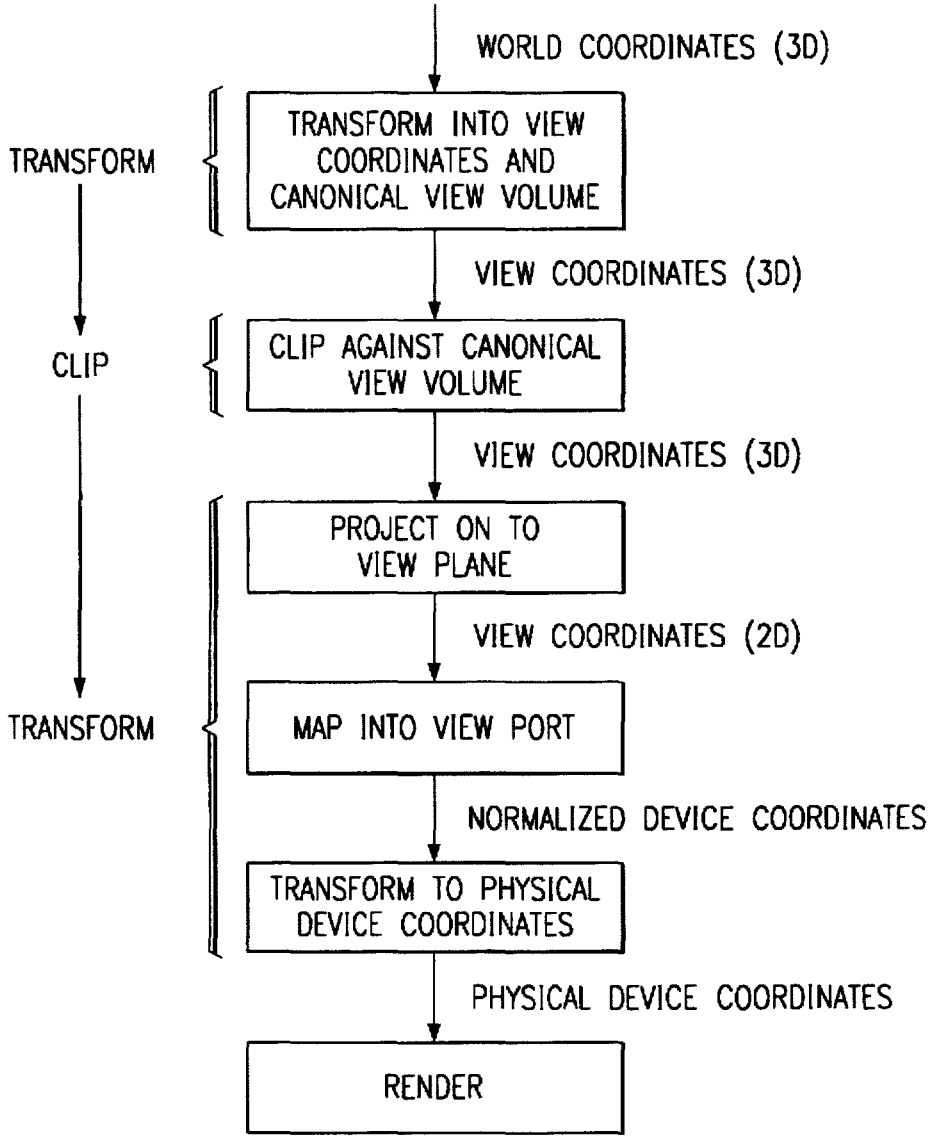
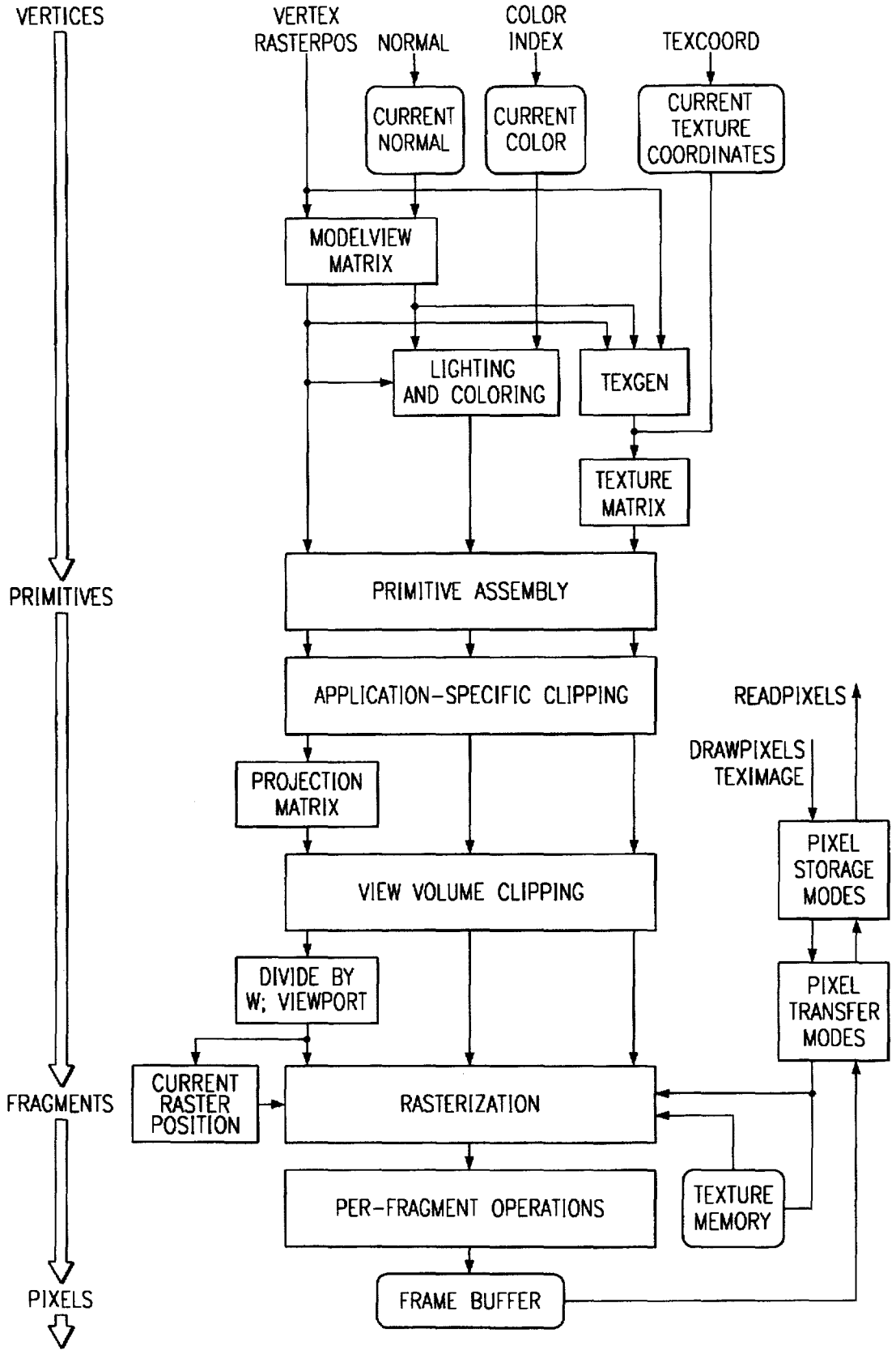

WORLD COORDINATES (3D)
TRANSFORM
TRANSFORM INTO VIEW COORDINATES AND CANONICAL VIEW VOLUME
VIEW COORDINATES (3D)
CLIP
CLIP AGAINST CANONICAL VIEW VOLUME
VIEW COORDINATES (3D)
PROJECT ON TO VIEW PLANE
VIEW COORDINATES (2D)
TRANSFORM
MAP INTO VIEW PORT
NORMALIZED DEVICE COORDINATES
TRANSFORM TO PHYSICAL DEVICE COORDINATES
PHYSICAL DEVICE COORDINATES
RENDER

FIG. 1B


VERTICES
VERTEX RASTERPOS
NORMAL
COLOR INDEX
TEXCOORD
CURRENT NORMAL
CURRENT COLOR
CURRENT TEXTURE COORDINATES
MODELVIEW MATRIX
LIGHTING AND COLORING
TEXGEN
TEXTURE MATRIX
PRIMITIVES
PRIMITIVE ASSEMBLY
APPLICATION-SPECIFIC CLIPPING
READPIXELS
DRAWPIXELS TEXIMAGE
PROJECTION MATRIX
PIXEL STORAGE MODES
VIEW VOLUME CLIPPING
DIVIDE BY W; VIEWPORT
PIXEL TRANSFER MODES
FRAGMENTS
CURRENT RASTER POSITION
RASTERIZATION
PER-FRAGMENT OPERATIONS
TEXTURE MEMORY
PIXELS
FRAME BUFFER

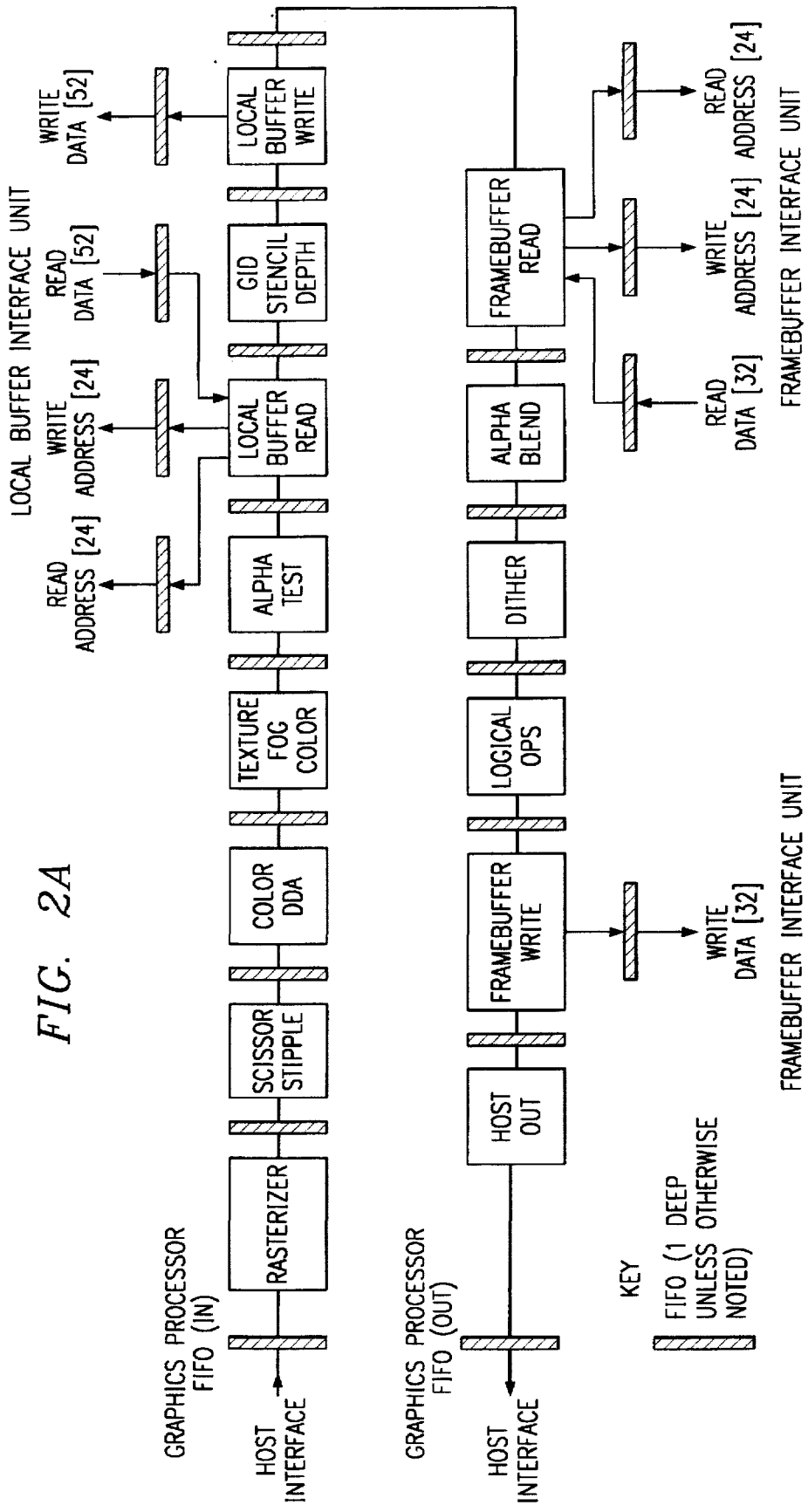

GRAPHICS PROCESSOR FIFO (IN)
HOST INTERFACE
RASTERIZER
SCISSOR STIPPLE
COLOR DDA
TEXTURE FOG COLOR
ALPHA TEST
LOCAL BUFFER READ
GID STENCIL DEPTH
LOCAL BUFFER WRITE
LOCAL BUFFER INTERFACE UNIT
READ ADDRESS [24]
WRITE ADDRESS [24]
READ DATA [52]
WRITE DATA [52]
GRAPHICS PROCESSOR FIFO (OUT)
HOST INTERFACE
HOST OUT
FRAMEBUFFER WRITE
LOGICAL OPS
DITHER
ALPHA BLEND
FRAMEBUFFER READ
FRAMEBUFFER INTERFACE UNIT
WRITE DATA [32]
READ DATA [32]
WRITE ADDRESS [24]
READ ADDRESS [24]
FRAMEBUFFER INTERFACE UNIT
KEY
FIFO (1 DEEP UNLESS OTHERWISE NOTED)

FIG. 2A

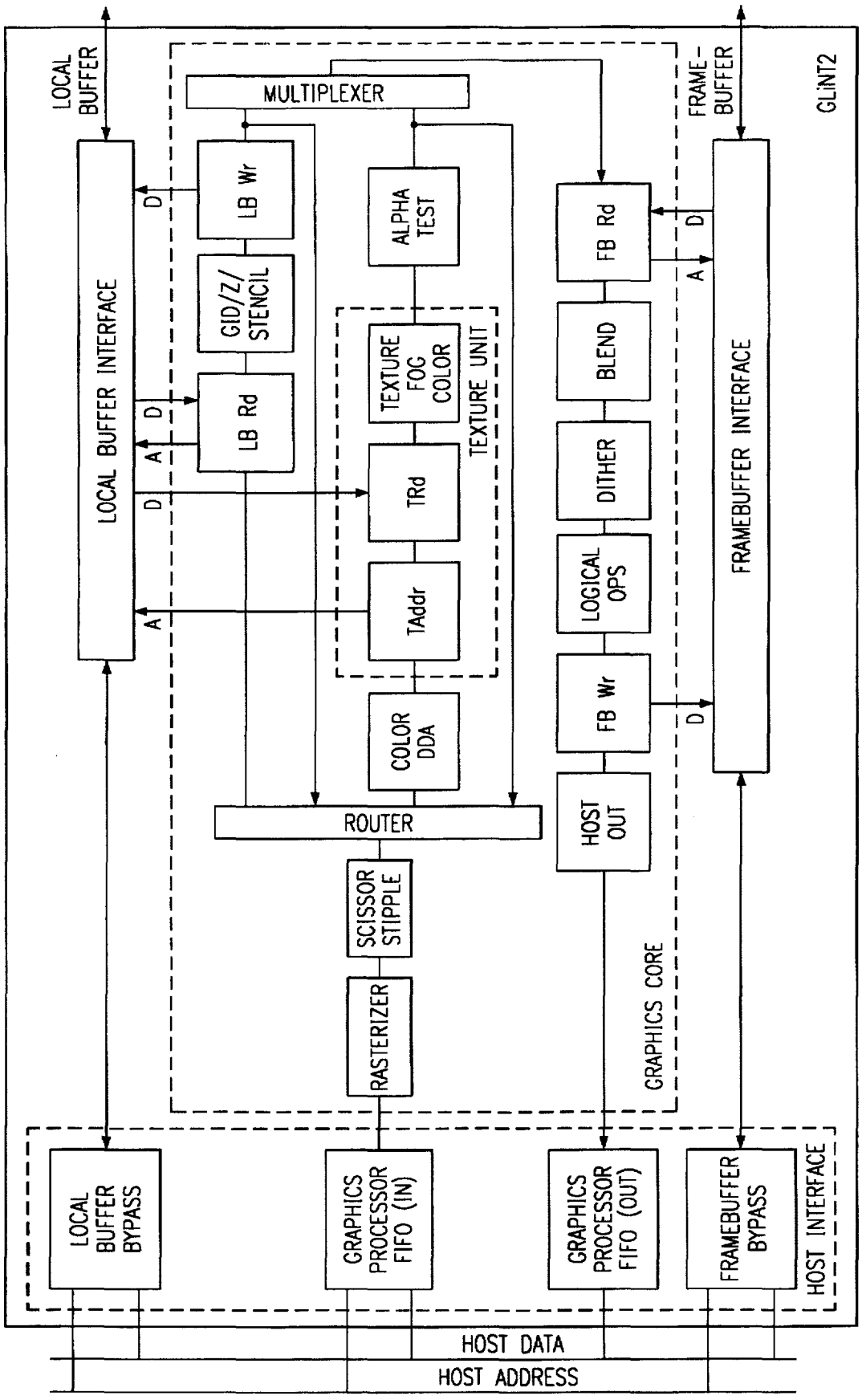
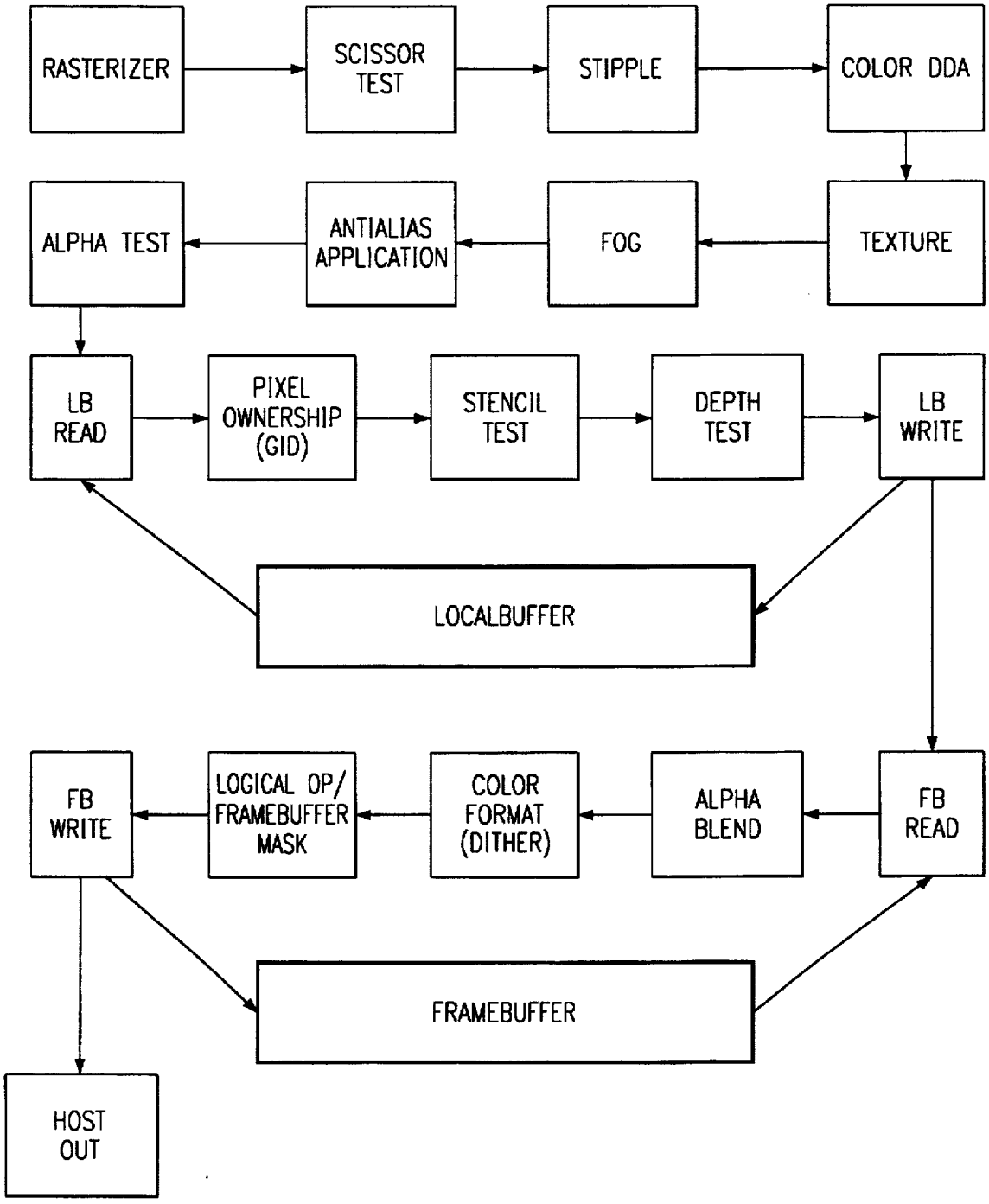

LOCAL BUFFER
MULTIPLEXER
FRAME-BUFFER
GLiNT2
LOCAL BUFFER INTERFACE
LB Wr
GID/Z/ STENCIL
LB Rd
ALPHA TEST
TEXTURE FOG COLOR
TRd
TAddr
TEXTURE UNIT
COLOR DDA
ROUTER
SCISSOR STIPPLE
RASTERIZER
FB Rd
BLEND
DITHER
LOGICAL OPS
FB Wr
HOST OUT
FRAMEBUFFER INTERFACE
GRAPHICS CORE
LOCAL BUFFER BYPASS
GRAPHICS PROCESSOR FIFO (IN)
GRAPHICS PROCESSOR FIFO (OUT)
FRAMEBUFFER BYPASS
HOST INTERFACE
HOST DATA
HOST ADDRESS
D
A

FIG. 2B

FIG. 2C


RASTERIZER
SCISSOR TEST
STIPPLE
COLOR DDA
TEXTURE
FOG
ANTIALIAS APPLICATION
ALPHA TEST
LB READ
PIXEL OWNERSHIP (GID)
STENCIL TEST
DEPTH TEST
LB WRITE
LOCALBUFFER
FB READ
ALPHA BLEND
COLOR FORMAT (DITHER)
LOGICAL OP/ FRAMEBUFFER MASK
FB WRITE
FRAMEBUFFER
HOST OUT

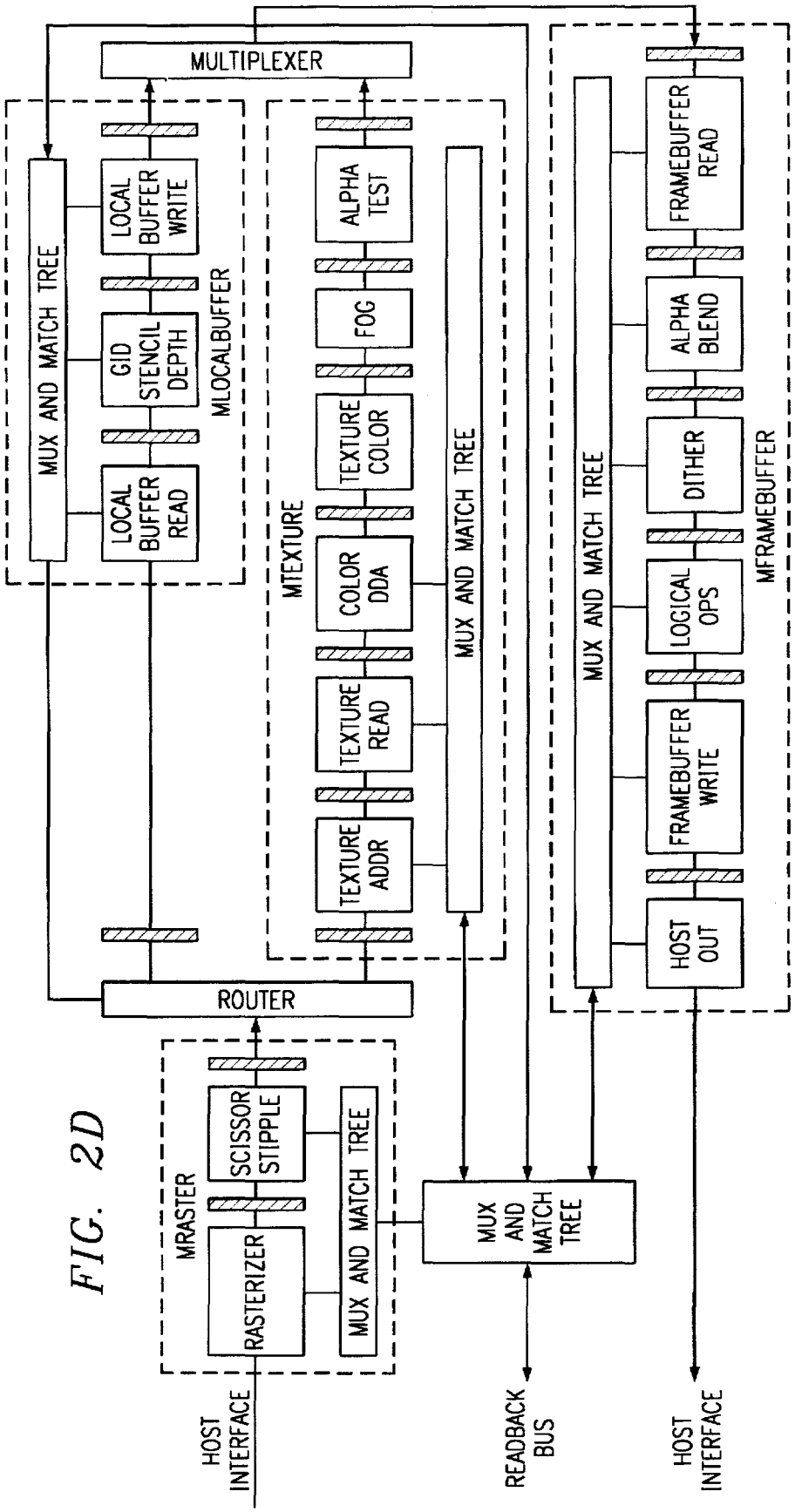

MULTIPLEXER
MUX AND MATCH TREE
LOCAL BUFFER WRITE
GID STENCIL DEPTH
LOCAL BUFFER READ
MLOCALBUFFER
MTEXTURE
ALPHA TEST
FOG
TEXTURE COLOR
COLOR DDA
TEXTURE READ
TEXTURE ADDR
MUX AND MATCH TREE
MUX AND MATCH TREE
FRAMEBUFFER READ
ALPHA BLEND
DITHER
LOGICAL OPS
FRAMEBUFFER WRITE
HOST OUT
MFRAMEBUFFER
ROUTER
MRASTER
SCISSOR STIPPLE
RASTERIZER
MUX AND MATCH TREE
MUX AND MATCH TREE
HOST INTERFACE
READBACK BUS
HOST INTERFACE

FIG. 2D

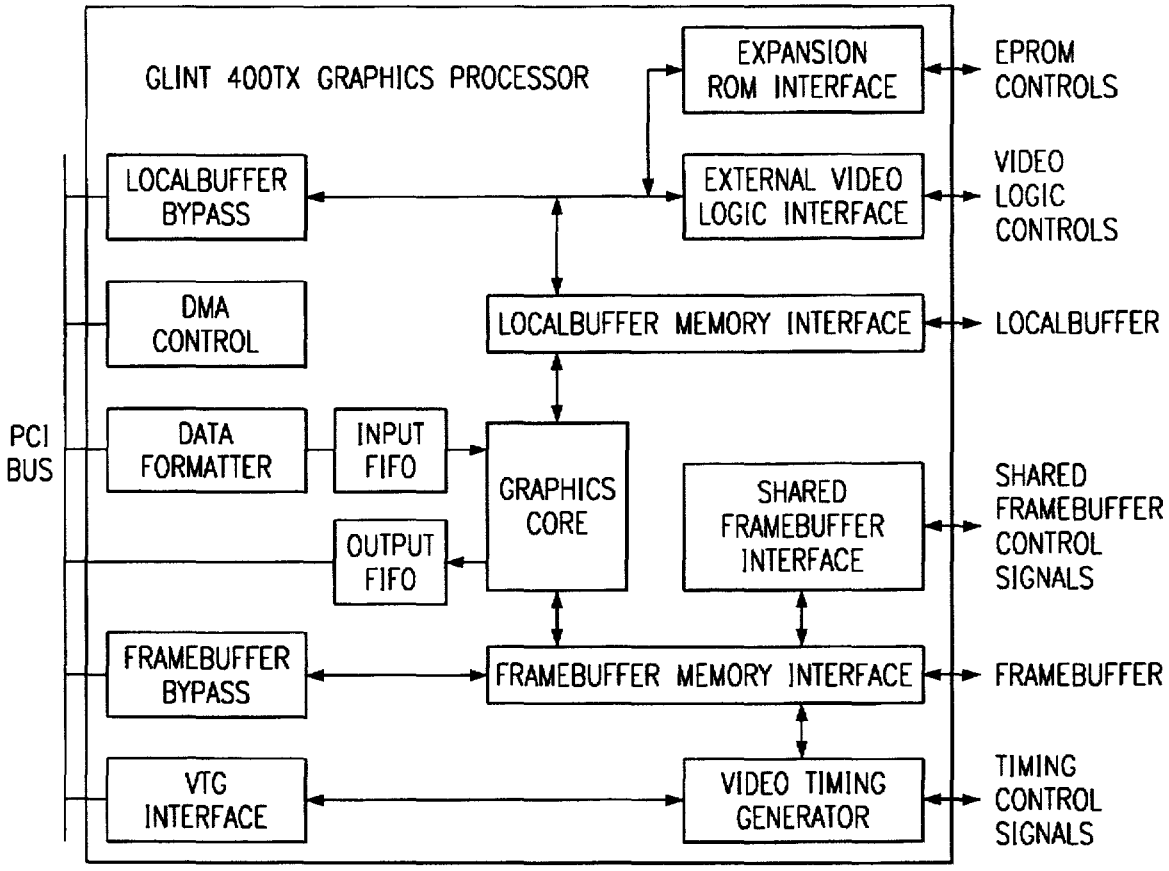

GLINT 400TX GRAPHICS PROCESSOR
EXPANSION ROM INTERFACE
EPROM CONTROLS
LOCALBUFFER BYPASS
EXTERNAL VIDEO LOGIC INTERFACE
VIDEO LOGIC CONTROLS
DMA CONTROL
LOCALBUFFER MEMORY INTERFACE
LOCALBUFFER
PCI BUS
DATA FORMATTER
INPUT FIFO
GRAPHICS CORE
SHARED FRAMEBUFFER INTERFACE
SHARED FRAMEBUFFER CONTROL SIGNALS
OUTPUT FIFO
FRAMEBUFFER BYPASS
FRAMEBUFFER MEMORY INTERFACE
FRAMEBUFFER
VTG INTERFACE
VIDEO TIMING GENERATOR
TIMING CONTROL SIGNALS

FIG. 2E

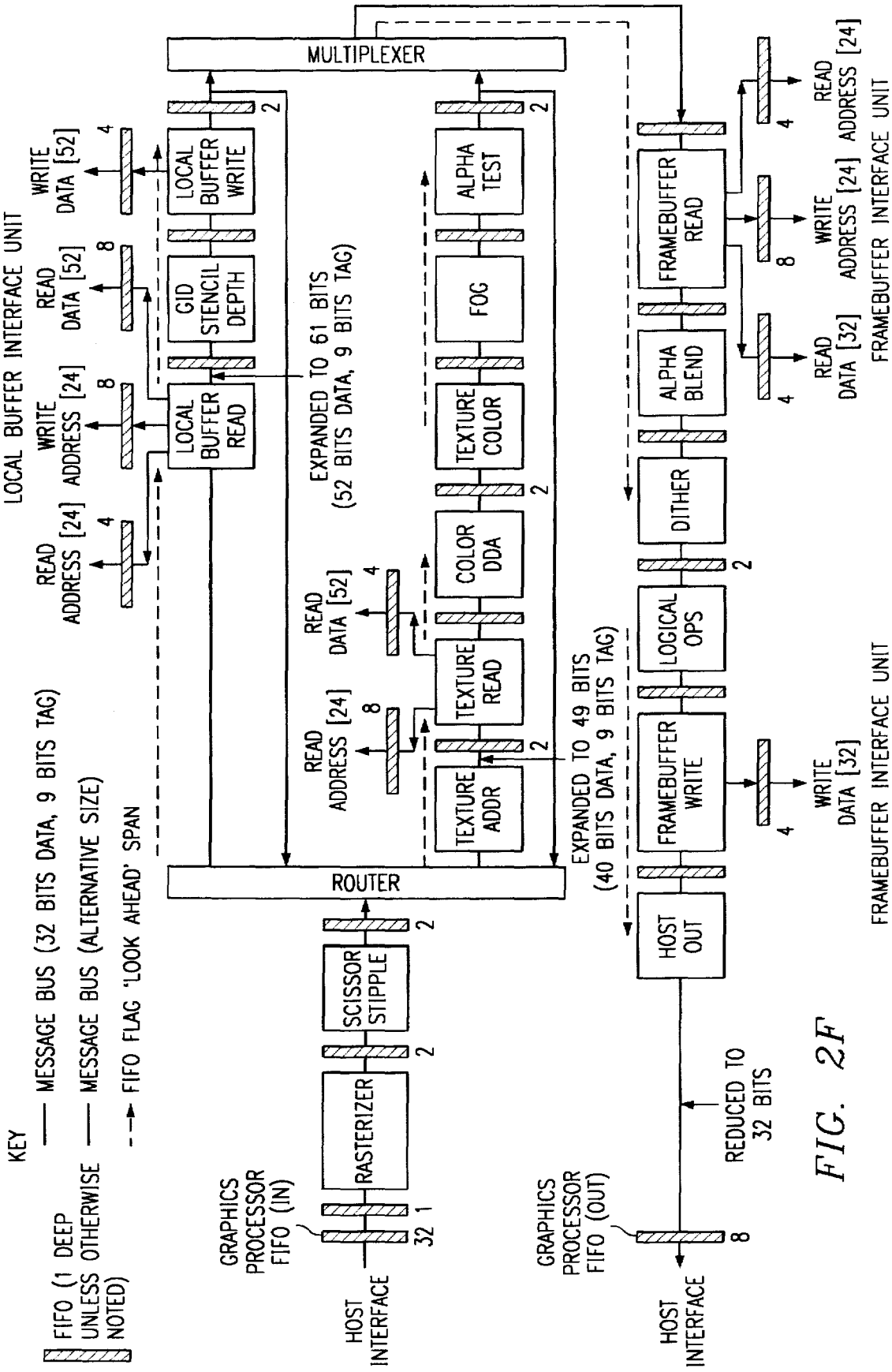

KEY
MESSAGE BUS (32 BITS DATA, 9 BITS TAG)
MESSAGE BUS (ALTERNATIVE SIZE)
FIFO FLAG 'LOOK AHEAD' SPAN
FIFO (1 DEEP UNLESS OTHERWISE NOTED)
LOCAL BUFFER INTERFACE UNIT
MULTIPLEXER
ROUTER
LOCAL BUFFER READ
LOCAL BUFFER WRITE
GID STENCIL DEPTH
TEXTURE ADDR
TEXTURE READ
COLOR DDA
TEXTURE COLOR
FOG
ALPHA TEST
SCISSOR STIPPLE
RASTERIZER
READ ADDRESS [24]
WRITE ADDRESS [24]
READ DATA [52]
WRITE DATA [52]
EXPANDED TO 61 BITS (52 BITS DATA, 9 BITS TAG)
EXPANDED TO 49 BITS (40 BITS DATA, 9 BITS TAG)
GRAPHICS PROCESSOR FIFO (IN)
HOST INTERFACE
HOST OUT
LOGICAL OPS
DITHER
ALPHA BLEND
FRAMEBUFFER READ
FRAMEBUFFER WRITE
READ DATA [32]
WRITE DATA [32]
READ ADDRESS [24]
WRITE ADDRESS [24]
FRAMEBUFFER INTERFACE UNIT
REDUCED TO 32 BITS
GRAPHICS PROCESSOR FIFO (OUT)
HOST INTERFACE

FIG. 2F

FIG. 3A

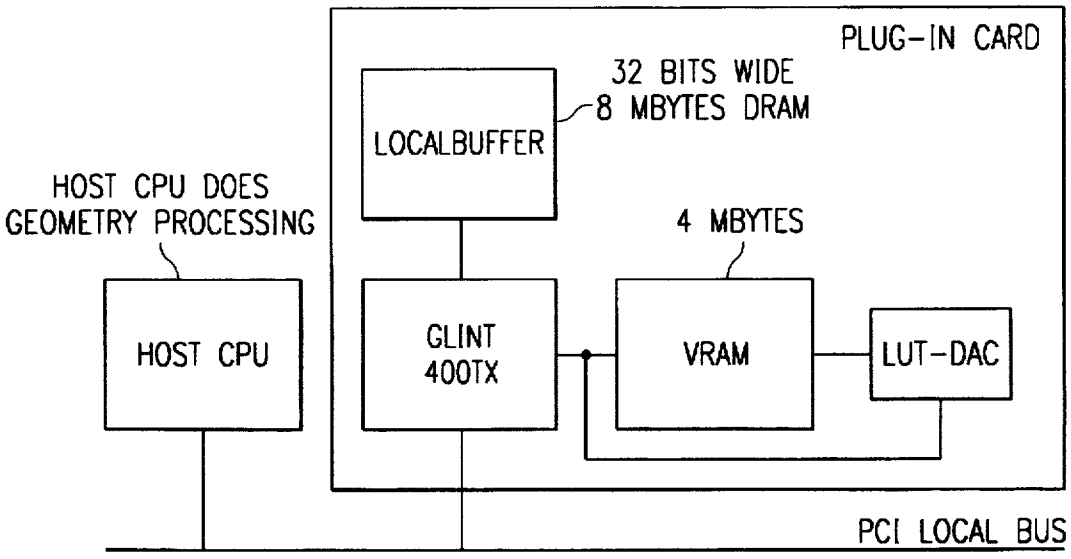

PLUG-IN CARD
32 BITS WIDE
8 MBYTES DRAM
LOCALBUFFER
HOST CPU DOES
GEOMETRY PROCESSING
4 MBYTES
HOST CPU
GLINT
400TX
VRAM
LUT-DAC
PCI LOCAL BUS

FIG. 3B

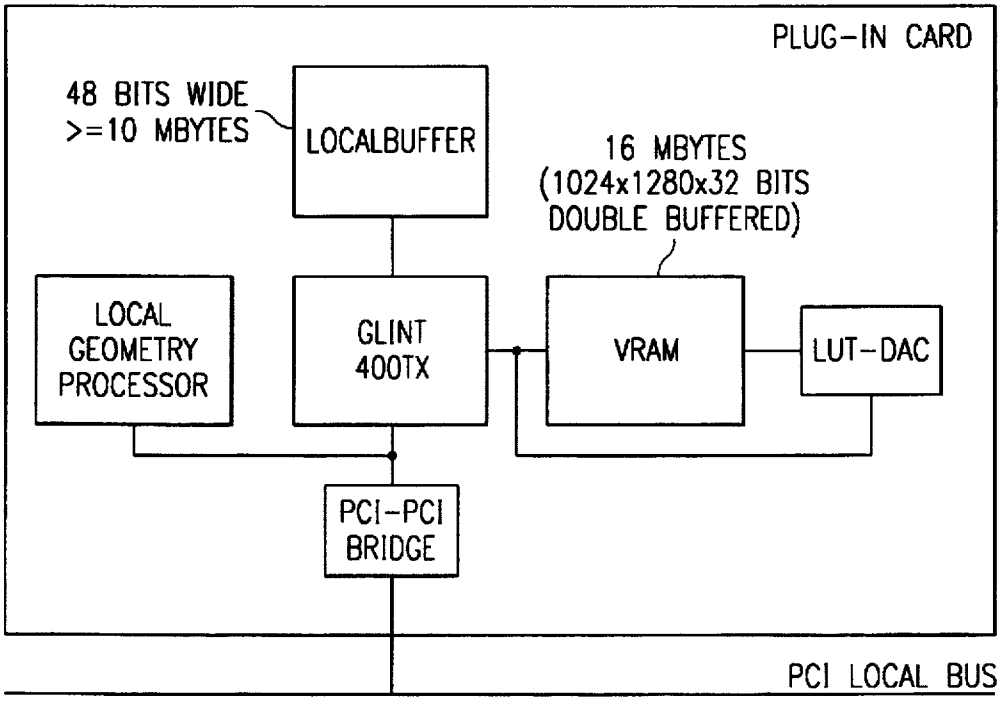

PLUG-IN CARD
48 BITS WIDE
>=10 MBYTES
LOCALBUFFER
16 MBYTES
(1024x1280x32 BITS
DOUBLE BUFFERED)
LOCAL
GEOMETRY
PROCESSOR
GLINT
400TX
VRAM
LUT-DAC
PCI-PCI
BRIDGE
PCI LOCAL BUS

FIG. 3C

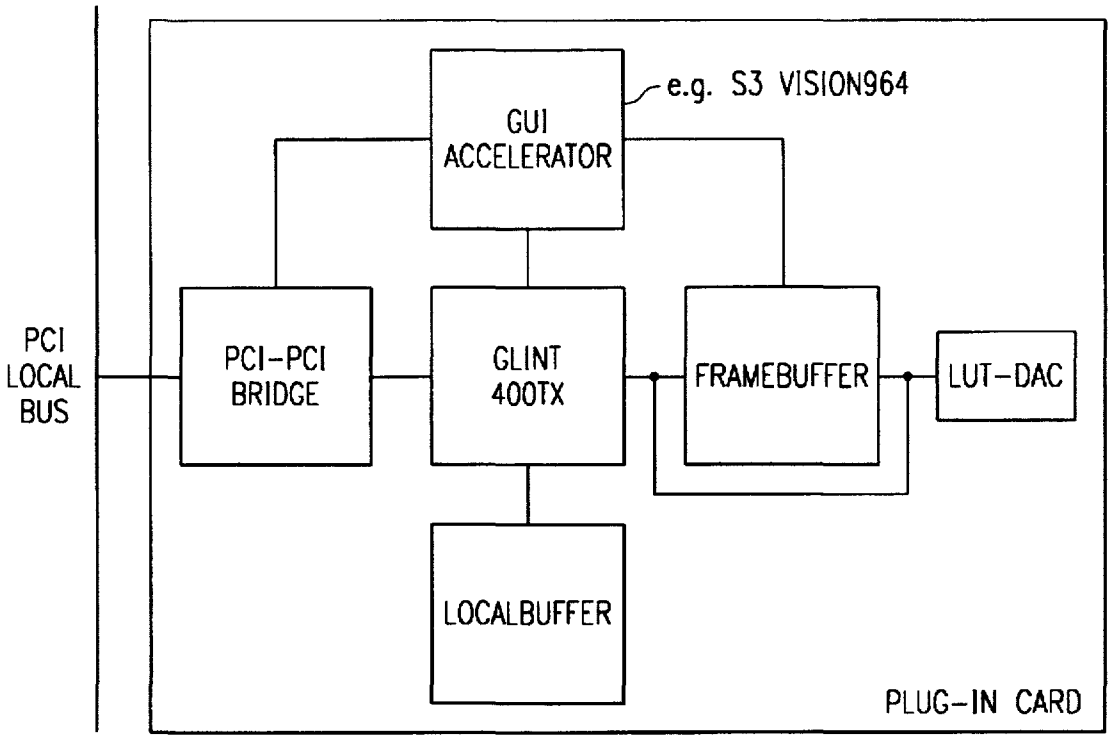

GUI
ACCELERATOR
e.g. S3 VISION964
PCI
LOCAL
BUS
PCI-PCI
BRIDGE
GLINT
400TX
FRAMEBUFFER
LUT-DAC
LOCALBUFFER
PLUG-IN CARD

FIG. 3D

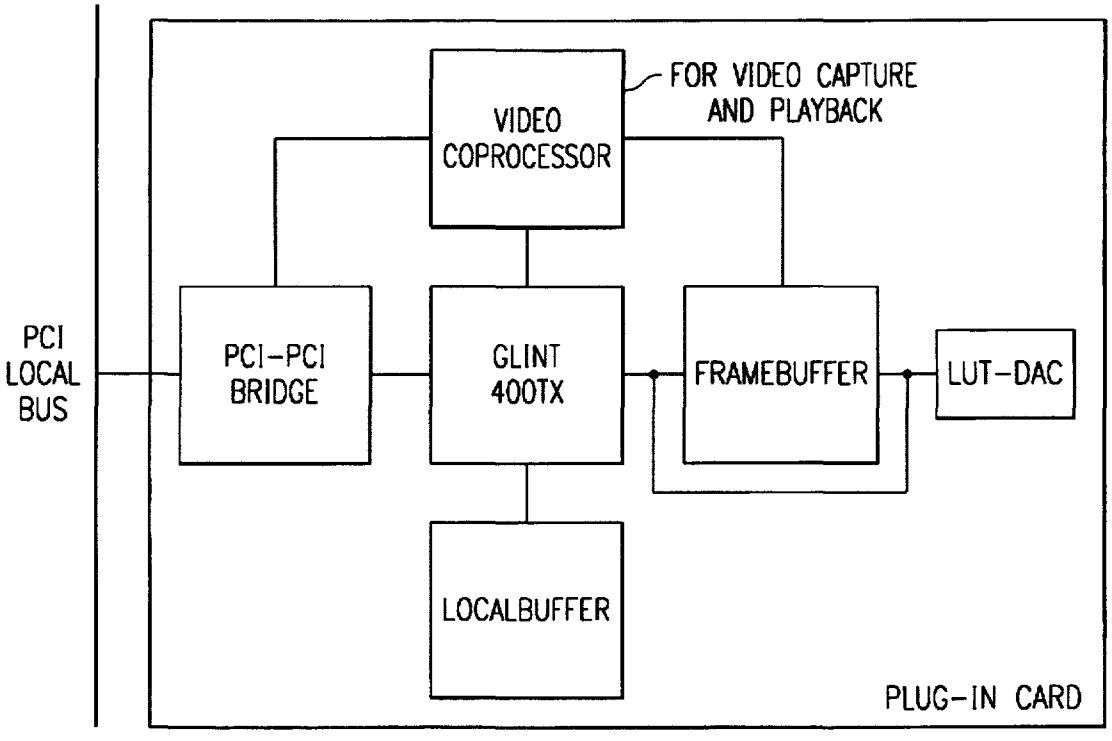

VIDEO
COPROCESSOR
FOR VIDEO CAPTURE
AND PLAYBACK
PCI
LOCAL
BUS
PCI-PCI
BRIDGE
GLINT
400TX
FRAMEBUFFER
LUT-DAC
LOCALBUFFER
PLUG-IN CARD

FIG. 4A

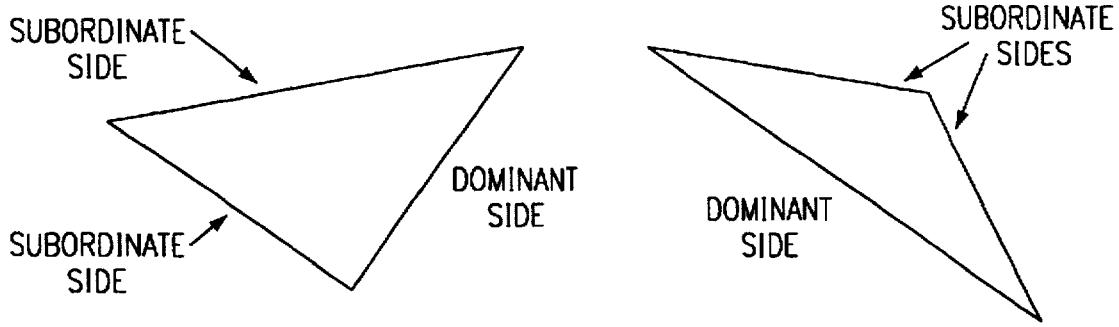
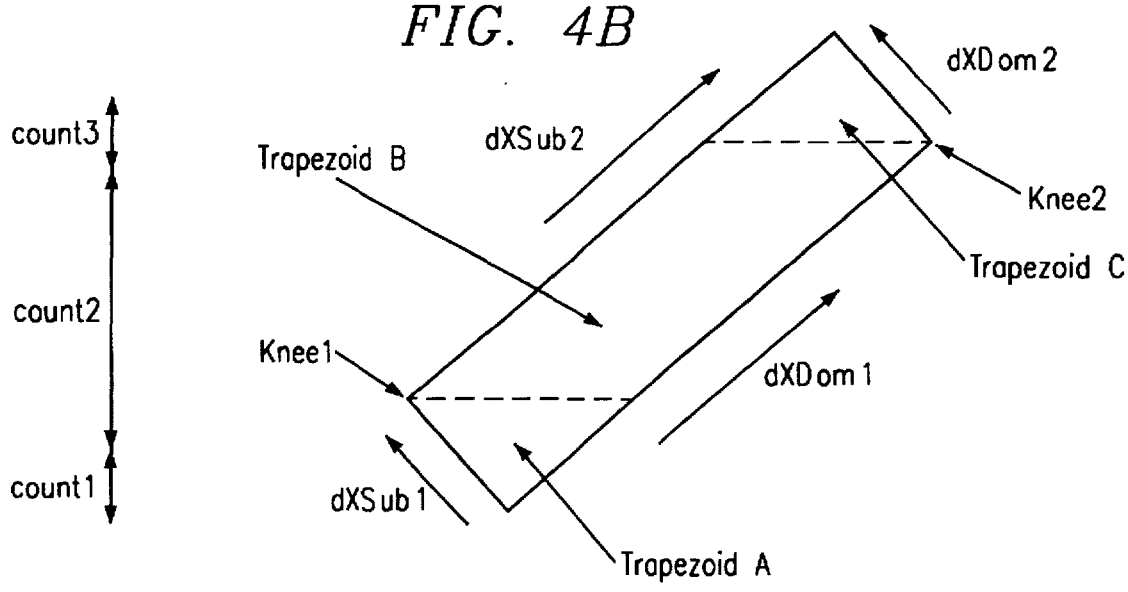

SUBORDINATE SIDE
SUBORDINATE SIDE
DOMINANT SIDE
SUBORDINATE SIDES
DOMINANT SIDE

FIG. 4B


count3
count2
count1
Trapezoid B
dXSub2
dXDom2
Knee2
Trapezoid C
Knee1
dXDom1
dXSub1
Trapezoid A

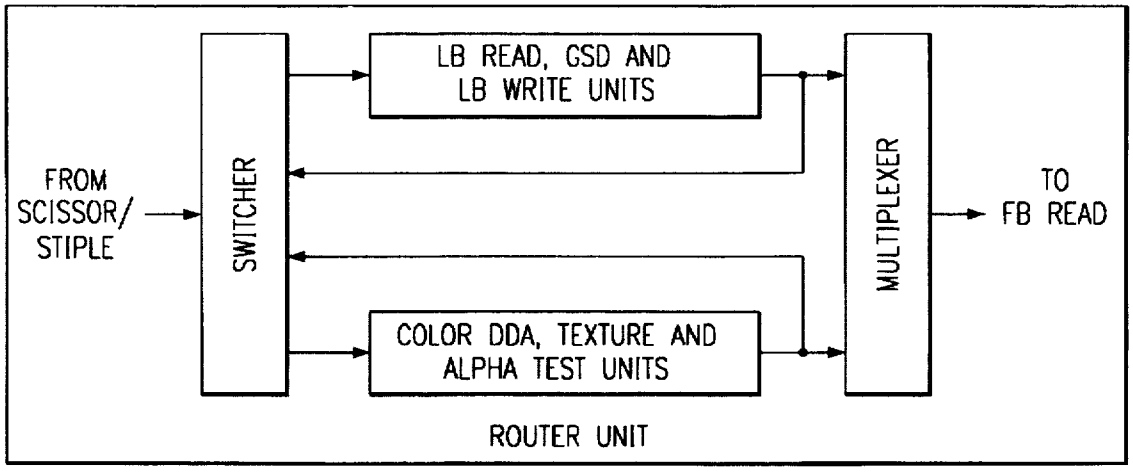


FIG. 5A

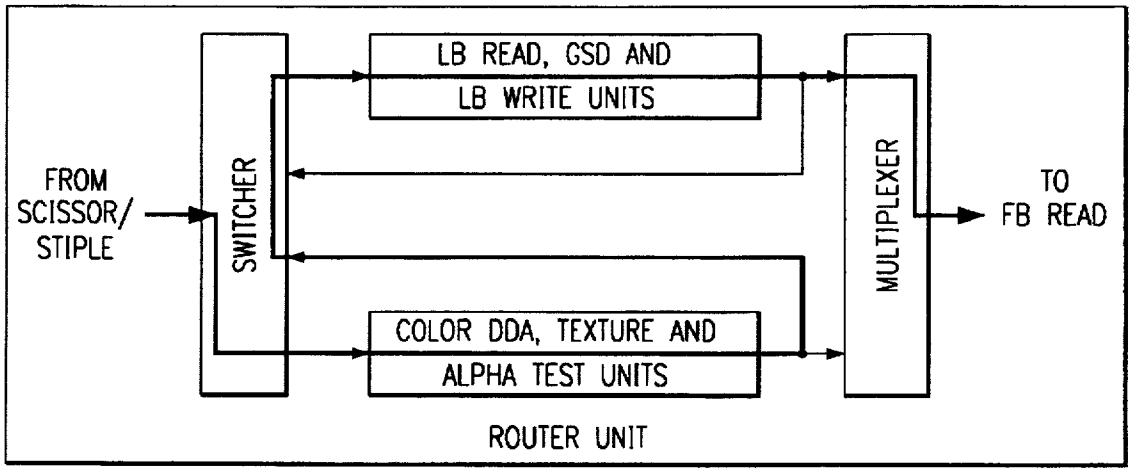


FIG. 5B

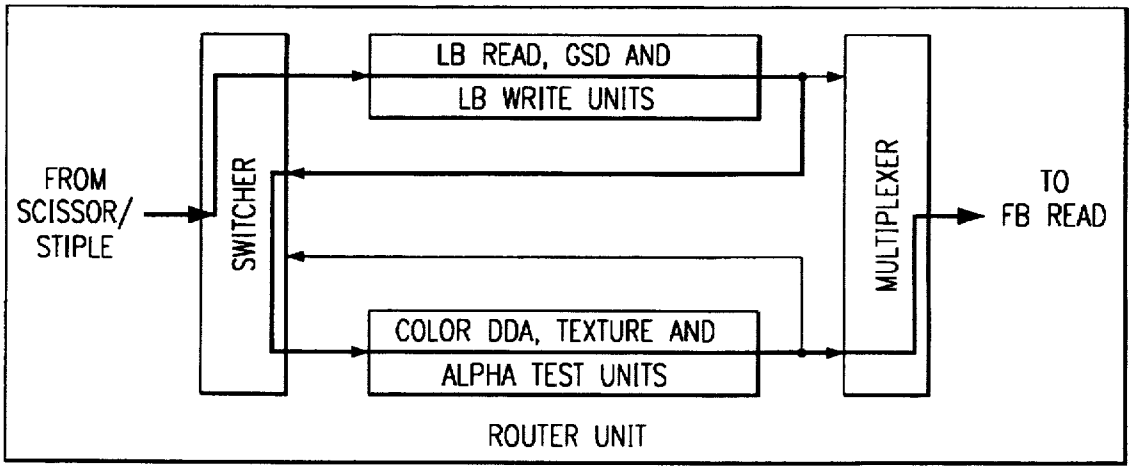


FIG. 5C

GRAPHICS RENDERING SYSTEM WITH RECONFIGURABLE PIPELINE SEQUENCE

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation-in-part of 08/410,345 filed Mar. 24, 1995, and claims priority from provisional 60/008,803 filed Dec. 18, 1995, which is hereby incorporated by reference.

BACKGROUND AND SUMMARY OF THE INVENTION

The present application relates to computer graphics and animation systems, and particularly to 3D graphics rendering hardware. Background of the art and the prior embodiment, according to the parent application, is described below. Some of the distinctions of the presently preferred embodiment are particularly noted beginning on page 8.

COMPUTER GRAPHICS AND RENDERING

Modern computer systems normally manipulate graphical objects as high-level entities. For example, a solid body may be described as a collection of triangles with specified vertices, or a straight line segment may be described by listing its two endpoints with three-dimensional or two-dimensional coordinates. Such high-level descriptions are a necessary basis for high-level geometric manipulations, and also have the advantage of providing a compact format which does not consume memory space unnecessarily.

Such higher-level representations are very convenient for performing the many required computations. For example, ray-tracing or other lighting calculations may be performed, and a projective transformation can be used to reduce a three-dimensional scene to its two-dimensional appearance from a given viewpoint. However, when an image containing graphical objects is to be displayed, a very low-level description is needed. For example, in a conventional CRT display, a "flying spot" is moved across the screen (one line at a time), and the beam from each of three electron guns is switched to a desired level of intensity as the flying spot passes each pixel location. Thus at some point the image model must be translated into a data set which can be used by a conventional display. This operation is known as "rendering."

The graphics-processing system typically interfaces to the display controller through a "frame store" or "frame buffer" of special two-port memory, which can be written to randomly by the graphics processing system, but also provides the synchronous data output needed by the video output driver. (Digital-to-analog conversion is also provided after the frame buffer.) Such a frame buffer is usually implemented using VRAM memory chips (or sometimes with DRAM and special DRAM controllers). This interface relieves the graphics processing system of most of the burden of synchronization for video output. Nevertheless, the amounts of data which must be moved around are very sizable, and the computational and data-transfer burden of placing the correct data into the frame buffer can still be very large.

Even if the computational operations required are quite simple, they must be performed repeatedly on a large number of data points. For example, in a typical 1995 high-end configuration, a display of 1280×1024 elements may need to be refreshed at 72 Hz, with a color resolution

of 24 bits per pixel. If blending is desired, additional bits (e.g. another 8 bits per pixel) will be required to store an "alpha" or transparency value for each pixel. This implies manipulation of more than 3 billion bits per second, without allowing for any of the actual computations being performed. Thus it may be seen that this is an environment with unique data manipulation requirements.

If the display is unchanging, no demand is placed on the rendering operations. However, some common operations (such as zooming or rotation) will require every object in the image space to be re-rendered. Slow rendering will make the rotation or zoom appear jerky. This is highly undesirable. Thus efficient rendering is an essential step in translating an image representation into the correct pixel values. This is particularly true in animation applications, where newly rendered updates to a computer graphics display must be generated at regular intervals.

The rendering requirements of three-dimensional graphics are particularly heavy. One reason for this is that, even after the three-dimensional model has been translated to a two-dimensional model, some computational tasks may be bequeathed to the rendering process. (For example, color values will need to be interpolated across a triangle or other primitive.) These computational tasks tend to burden the rendering process. Another reason is that since three-dimensional graphics are much more lifelike, users are more likely to demand a fully rendered image. (By contrast, in the two-dimensional images created e.g. by a GUI or simple game, users will learn not to expect all areas of the scene to be active or filled with information.)

FIG. 1A is a very high-level view of other processes performed in a 3D graphics computer system. A three dimensional image which is defined in some fixed 3D coordinate system (a "world" coordinate system) is transformed into a viewing volume (determined by a view position and direction), and the parts of the image which fall outside the viewing volume are discarded. The visible portion of the image volume is then projected onto a viewing plane, in accordance with the familiar rules of perspective. This produces a two-dimensional image, which is now mapped into device coordinates. It is important to understand that all of these operations occur prior to the operations performed by the rendering subsystem of the present invention. FIG. 1B is an expanded version of FIG. 1A, and shows the flow of operations defined by the OpenGL standard.

A vast amount of engineering effort has been invested in computer graphics systems, and this area is one of increasing activity and demands. Numerous books have discussed the requirements of this area; see, e.g., ADVANCES IN COMPUTER GRAPHICS (ed. Enderle 1990-); Chellappa and Sawchuk, DIGITAL IMAGE PROCESSING AND ANALYSIS (1985); COMPUTER GRAPHICS HARDWARE (ed. Reghbati and Lee 1988); COMPUTER GRAPHICS: IMAGE SYNTHESIS (ed. Joy et al.); Foley et al., FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS (2.ed. 1984); Foley, COMPUTER GRAPHICS PRINCIPLES & PRACTICE (2.ed. 1990); Foley, INTRODUCTION TO COMPUTER GRAPHICS (1994); Giloi, Interactive Computer Graphics (1978); Hearn and Baker, COMPUTER GRAPHICS (2.ed. 1994); Hill, COMPUTER GRAPHICS (1990); Latham, DICTIONARY OF COMPUTER GRAPHICS (1991); Magnenat-Thalma, IMAGE SYNTHESIS THEORY & PRACTICE (1988); Newman and Sproull, PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS (2.ed. 1979); PICTURE ENGINEERING (ed. Fu and Kunii 1982); PICTURE PROCESSING & DIGITAL FILTERING (2.ed. Huang 1979); Prosise, HOW COMPUTER GRAPHICS WORK (1994); Rimmer, BIT MAPPED GRAPHICS (2.ed. 1993); Salmon, COMPUTER GRAPHICS SYSTEMS & CONCEPTS

(1987); Schachter, COMPUTER IMAGE GENERATION (1990); Watt, THREE-DIMENSIONAL COMPUTER GRAPHICS (2.ed. 1994); Scott Whitman, MULTIPROCESSOR METHODS FOR COMPUTER GRAPHICS RENDERING; the SIGGRAPH PROCEEDINGS for the years 1980–1994; and the *IEEE Computer Graphics and Applications* magazine for the years 1990–1994.

Background: Graphics Animation

In many areas of computer graphics a succession of slowly changing pictures are displayed rapidly one after the other, to give the impression of smooth movement, in much the same way as for cartoon animation. In general the higher the speed of the animation, the smoother (and better) the result.

When an application is generating animation images, it is normally necessary not only to draw each picture into the frame buffer, but also to first clear down the frame buffer, and to clear down auxiliary buffers such as depth (Z) buffers, stencil buffers, alpha buffers and others. A good treatment of the general principles may be found in *Computer Graphics: Principles and Practice*, James D. Foley et al., Reading Mass.: Addison-Wesley. A specific description of the various auxiliary buffers may be found in *The OpenGL Graphics System: A Specification* (*Version* 1.0), Mark Segal and Kurt Akeley, SGI.

In most applications the value written, when clearing any given buffer, is the same at every pixel location, though different values may be used in different auxiliary buffers. Thus the frame buffer is often cleared to the value which corresponds to black, while the depth (Z) buffer is typically cleared to a value corresponding to infinity.

The time taken to clear down the buffers is often a significant portion of the total time taken to draw a frame, so it is important to minimize it.

Background: Parallelism in Graphics Processing

Due to the large number of at least partially independent operations which are performed in rendering, many proposals have been made to use some form of parallel architecture for graphics (and particularly for rendering). See, for example, the special issue of *Computer Graphics* on parallel rendering (September 1994). Other approaches may be found in earlier patent filings by the assignee of the present application and its predecessors, e.g. U.S. Pat. No. 5,195,186, and published PCT applications PCT/GB90/00987, PCT/GB90/01209, PCT/GB90/01210, PCT/GB90/01212, PCT/GB90/01213, PCT/GB90/01214, PCT/GB90/01215, and PCT/GB90/01216.

Background: Pipelined Processing Generally

There are several general approaches to parallel processing. One of the basic approaches to achieving parallelism in computer processing is a technique known as pipelining. In this technique the individual processors are, in effect, connected in series in an assembly-line configuration: one processor performs a first set of operations on one chunk of data, and then passes that chunk along to another processor which performs a second set of operations, while at the same time the first processor performs the first set operations again on another chunk of data. Such architectures are generally discussed in Kogge, THE ARCHITECTURE OF PIPELINED COMPUTERS (1981).

Background: The OpenGL™ Standard

The "OpenGL" standard is a very important software standard for graphics applications. In any computer system which supports this standard, the operating system(s) and application software programs can make calls according to the OpenGL standards, without knowing exactly what the hardware configuration of the system is.

The OpenGL standard provides a complete library of low-level graphics manipulation commands, which can be used to implement three-dimensional graphics operations. This standard was originally based on the proprietary standards of Silicon Graphics, Inc., but was later transformed into an open standard. It is now becoming extremely important, not only in high-end graphics-intensive workstations, but also in high-end PCs. OpenGL is supported by Windows NT™, which makes it accessible to many PC applications.

The OpenGL specification provides some constraints on the sequence of operations. For instance, the color DDA operations must be performed before the texturing operations, which must be performed before the alpha operations. (A "DDA" or digital differential analyzer, is a conventional piece of hardware used to produce linear gradation of color (or other) values over an image area.)

Other graphics interfaces (or "APIs"), such as PHIGS or XGL, are also current as of 1995; but at the lowest level, OpenGL is a superset of most of these.

The OpenGL standard is described in the *OPENGL PROGRAMMING GUIDE* (1993), the OPENGL REFERENCE MANUAL (1993), and a book by Segal and Akeley (of SGI) entitled THE OPENGL GRAPHICS SYSTEM: A SPECIFICATION (Version 1.0).

FIG. 1B is an expanded version of FIG. 1A, and shows the flow of operations defined by the OpenGL standard. Note that the most basic model is carried in terms of vertices, and these vertices are then assembled into primitives (such as triangles, lines, etc.). After all manipulation of the primitives has been completed, the rendering operations will translate each primitive into a set of "fragments." (A fragment is the portion of a primitive which affects a single pixel.) Again, it should be noted that all operations above the block marked "Rasterization" would be performed by a host processor, or possibly by a "geometry engine" (i.e. a dedicated processor which performs rapid matrix multiplies and related data manipulations), but would normally not be performed by a dedicated rendering processor such as that of the presently preferred embodiment.

One disadvantage of standards such as OpenGL is that they require that texturing or other processor-intensive operations be performed on data before pixel elimination tests, e.g. depth testing, is performed, which wastes processor time by performing costly texturing calculations on pixels which will be eliminated later in the pipeline. When the OpenGL specification is not required or when the current OpenGI state vector cannot eliminate pixels as a result of the alpha test, however, it would be much more efficient to eliminate as many pixels as possible before doing these calculations. The present application discloses a method and device for reordering the processing steps in the rendering pipeline to either accommodate order-specific specifications such as OpenGL, or to provide for an optimized throughput by only performing processor-intensive operations on pixels which will actually be displayed.

Background: Texturing

Texture patterns are commonly used as a way to apply realistic visual detail at the sub-polygon level. See Foley et al., COMPUTER GRAPHICS: PRINCIPLES AND PRACTICE (2.ed. 1990, corr. 1995), especially at pages 741–744; Paul S. Heckbert, "Fundamentals of Texture Mapping and Image Warping," Thesis submitted to Dept. of EE and Computer Science, University of California, Berkeley, Jun. 17, 1994; Heckbert, "Survey of Computer Graphics," IEEE Computer Graphics, November 1986, pp.56ff. Since the surfaces are transformed (by the host or geometry engine) to produce a

2D view, the textures will need to be similarly transformed by a linear transform (normally projective or "affine"). (In conventional terminology, the coordinates of the object surface, i.e. the primitive being rendered, are referred to as an (s,t) coordinate space, and the map of the stored texture is referred to a (u,v) coordinate space.) The transformation in the resulting mapping means that a horizontal line in the (x,y) display space is very likely to correspond to a slanted line in the (u,v) space of the texture map, and hence many page breaks will occur, due to the texturing operation, as rendering walks along a horizontal line of pixels.

Innovative System and Methods

The preferred embodiment discloses a pipelined graphics processor in which the sequence can be dynamically reconfigured (e.g. between primitives) in a rendering sequence. The pipeline sequence can be configured for compliance with specifications such as OpenGL, but may also be optimized by reconfiguring the pipeline sequence to eliminate unnecessary processing. In a preferred embodiment, pixel elimination sequences such as depth and stencil tests are performed before texturing calculations are performed, so that unneeded pixel data is discarded before said texturing calculations are performed.

It is noted that the texturing operations become more computation-intense, early elimination of unneeded pixels becomes even more valuable. For example, Phong shading and bump mapping both require many more operations than more common shading and texture mapping techniques, thus making the system of the present application even more valuable in real-time rendering systems.

An overhead cost is that the reconfigurable portion of the pipeline must be flushed at each reconfiguration—but since reconfiguration is normally done only on a per-primitive basis, or even less frequently, this is a relatively small cost.

BRIEF DESCRIPTION OF THE DRAWING

The disclosed inventions will be described with reference to the accompanying drawings, which show important sample embodiments of the invention and which are incorporated in the specification hereof by reference, wherein:

FIG. **1A**, described above, is an overview of key elements and processes in a 3D graphics computer system.

FIG. **1B** is an expanded version of FIG. **1A**, and shows the flow of operations defined by the OpenGL standard.

FIG. **2A** is an overview of the graphics rendering chip of the preferred embodiment of the parent case.

FIG. **2B** is an overview of the graphics rendering chip of the presently preferred embodiment.

FIG. **2C** is a more schematic view of the sequence of operations performed in the graphics rendering chip of FIG. **2B**, when operating in a first mode.

FIG. **2D** is a different view of the graphics rendering chip of FIG. **2B**, showing the connections of a readback bus which provides a diagnostic pathway.

FIG. **2E** is yet another view of the graphics rendering chip of FIG. **2B**, showing how the functions of the core pipeline of FIG. **2C** are combined with various external interface functions.

FIG. **2F** is yet another view of the graphics rendering chip of FIG. **2B**, showing how the details of FIFO depth and lookahead are implemented, in the presently preferred embodiment.

FIG. **3A** shows a sample graphics board which incorporates the chip of FIG. **2B**.

FIG. **3B** shows another sample graphics board implementation, which differs from the board of FIG. **3A** in that more memory and an additional component is used to achieve higher performance.

FIG. **3C** shows another graphics board, in which the chip of FIG. **2B** shares access to a common frame store with GUI accelerator chip.

FIG. **3D** shows another graphics board, in which the chip of FIG. **2B** shares access to a common frame store with a video coprocessor (which may be used for video capture and playback functions.

FIG. **4A** illustrates the definition of the dominant side and the subordinate sides of a triangle.

FIG. **4B** illustrates the sequence of rendering an Anti-aliased Line primitive.

FIG. **5A** is a detailed view of the router unit of the presently preferred embodiment.

FIG. **5B** is a detailed view of the data path through the router unit of the presently preferred embodiment when operating in a first mode.

FIG. **5C** is a detailed view of the data path through the router unit of the presently preferred embodiment when operating in a second mode.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The numerous innovative teachings of the present application will be described with particular reference to the presently preferred embodiment (by way of example, and not of limitation). The presently preferred embodiment is a GLINT™ 400TX™ 3D rendering chip. The Hardware Reference Manual and Programmer's Reference Manual for this chip describe further details of this sample embodiment. Both are available, as of the effective filing date of this application, from 3Dlabs Inc. Ltd., 181 Metro Drive, Suite 520, San Jose Calif. 95110.

Definitions

The following definitions may help in understanding the exact meaning of terms used in the text of this application:

application: a computer program which uses graphics animation.

depth (Z) buffer: A memory buffer containing the depth component of a pixel. Used to, for example, eliminate hidden surfaces.

blt double-buffering: A technique for achieving smooth animation, by rendering only to an undisplayed back buffer, and then copying the back buffer to the front once drawing is complete.

FrameCount Planes: Used to allow higher animation rates by enabling DRAM local buffer pixel data, such as depth (Z), to be cleared down quickly.

frame buffer: An area of memory containing the displayable color buffers (front, back, left, right, overlay, underlay). This memory is typically separate from the local buffer.

local buffer: An area of memory which may be used to store non-displayable pixel information: depth(Z), stencil, FrameCount and GID planes. This memory is typically separate from the framebuffer.

pixel: Picture element. A pixel comprises the bits in all the buffers (whether stored in the local buffer or framebuffer), corresponding to a particular location in the framebuffer.

stencil buffer: A buffer used to store information about a pixel which controls how subsequent stencilled pixels at the same location may be combined with the current value

in the framebuffer. Typically used to mask complex two-dimensional shapes.

Preferred Chip Embodiment—Overview

The GLINT™ high performance graphics processors combine workstation class 3D graphics acceleration, and state-of-the-art 2D performance in a single chip. All 3D rendering operations are accelerated by GLINT, including Gouraud shading, texture mapping, depth buffering, anti-aliasing, and alpha blending.

The scalable memory architecture of GLINT makes it ideal for a wide range of graphics products, from PC boards to high-end workstation accelerators.

There will be several of the GLINT family of graphics processors: the GLINT 300SX™ is the embodiment of the parent case, and the GLINT 400TX™ is a presently preferred embodiment which is which is described herein in great detail. The two devices are generally compatible, with the 400TX adding local texture storage and texel address generation for all texture modes.

FIG. 2B is an overview of the graphics rendering chip of the presently preferred embodiment (i.e. the GLINT 400TX™).

General Concept

The overall architecture of the GLINT chip is best viewed using the software paradigm of a message passing system. In this system all the processing blocks are connected in a long pipeline with communication with the adjacent blocks being done through message passing. Between each block there is a small amount of buffering, the size being specific to the local communications requirements and speed of the two blocks.

The message rate is variable and depends on the rendering mode. The messages do not propagate through the system at a fixed rate typical of a more traditional pipeline system. If the receiving block can not accept a message, because its input buffer is full, then the sending block stalls until space is available.

The message structure is fundamental to the whole system as the messages are used to control, synchronize and inform each block about the processing it is to undertake. Each message has two fields—a 32 bit data field and a 9 bit tag field. (This is the minimum width guaranteed, but some local block to block connections may be wider to accommodate more data.) The data field will hold color information, coordinate information, local state information, etc. The tag field is used by each block to identify the message type so it knows how to act on it.

Each block, on receiving a message, can do one of several things:

- Not recognize the message so it just passes it on to the next block.
- Recognize it as updating some local state (to the block) so the local state is updated and the message terminated, i.e. not passed on to the next block.
- Recognize it as a processing action, and if appropriate to the unit, the processing work specific to the unit is done. This may entail sending out new messages such as Color and/or modifying the initial message before sending it on. Any new messages are injected into the message stream before the initial message is forwarded on. Some examples will clarify this.

When the Depth Block receives a message 'new fragment', it will calculate the corresponding depth and do the depth test. If the test passes then the 'new fragment' message is passed to the next unit. If the test fails then the message is modified and passed on. The temptation is not to pass the message on when the test fails (because the pixel is not going to be updated), but other units downstream need to keep their local DDA units in step.

(In the present application, the messages are being described in general terms so as not to be bogged down in detail at this stage. The details of what a 'new fragment' message actually specifies (i.e. coordinate, color information) is left till later. In general, the term "pixel" is used to describe the picture element on the screen or in memory. The term "fragment" is used to describe the part of a polygon or other primitive which projects onto a pixel. Note that a fragment may only cover a part of a pixel.) When the Texture Read Unit (if enabled) gets a 'new fragment' message, it will calculate the texture map addresses, and will accordingly provide 1, 2, 4 or 8 texels to the next unit together with the appropriate number of interpolation coefficients.

Each unit and the message passing are conceptually running asynchronous to all the others. However, in the presently preferred embodiment there is considerable synchrony because of the common clock.

How does the host process send messages? The message data field is the 32 bit data written by the host, and the message tag is the bottom 9 bits of the address (excluding the byte resolution address lines). Writing to a specific address causes the message type associated with that address to be inserted into the message queue. Alternatively, the on-chip DMA controller may fetch the messages from the host's memory.

The message throughput, in the presently preferred embodiment, is 50M messages per second and this gives a fragment throughput of up to 50M per second, depending on what is being rendered. Of course, this rate will predictably be further increased over time, with advances in process technology and clock rates.

Linkage

The block diagram of FIG. 2A shows how the units are connected together in the GLINT 300SX embodiment, and the block diagram of FIG. 2B shows how the units are connected together in the presently preferred embodiment. Some general points are:

- The following functionality is present in the 400TX, but missing from the 300SX: The Texture Address (TAddr) and Texture Read (TRd) Units are missing. Also, the router and multiplexer are missing from this section, so the unit ordering is Scissor/Stipple, Color DDA, Texture Fog Color, Alpha Test, LB Rd, etc.
- In the embodiment of FIG. 2B, the order of the units can be configured in two ways. The most general order (Router, Color DDA, Texture Unit, Alpha Test, LB Rd, GID/Z/Stencil, LB Wr, Multiplexer) and will work in all modes of OpenGL. However, when the alpha test is disabled it is much better to do the Graphics ID, depth and stencil tests before the texture operations rather than after. This is because the texture operations have a high processing cost and this should not be spent on fragments which are later rejected because of window, depth or stencil tests.
- The loop back to the host at the bottom provides a simple synchronization mechanism. The host can insert a Sync command and when all the preceding rendering has finished the sync command will reach the bottom host interface which will notify the host the sync event has occurred.

Benefits

The very modular nature of this architecture gives great benefits. Each unit lives in isolation from all the others and

has a very well defined set of input and output messages. This allows the internal structure of a unit (or group of units) to be changed to make algorithmic/speed/gate count tradeoffs.

The isolation and well defined logical and behavioral interface to each unit allows much better testing and verification of the correctness of a unit.

The message passing paradigm is easy to simulate with software, and the hardware design is nicely partitioned. The architecture is self synchronizing for mode or primitive changes.

The host can mimic any block in the chain by inserting messages which that block would normally generate. These message would pass through the earlier blocks to the mimicked block unchanged and from then onwards to the rest of the blocks which cannot tell the message did not originate from the expected block. This allows for an easy work around mechanism to correct any flaws in the chip. It also allows other rasterization paradigms to be implemented outside of the chip, while still using the chip for the low level pixel operations.

"A Day in the Life of a Triangle"

Before we get too detailed in what each unit does it is worth while looking in general terms at how a primitive (e.g. triangle) passes through the pipeline, what messages are generated, and what happens in each unit. Some simplifications have been made in the description to avoid detail which would otherwise complicate what is really a very simple process. The primitive we are going to look at is the familiar Gouraud shaded Z buffered triangle, with dithering. It is assumed any other state (i.e. depth compare mode) has been set up, but (for simplicity) such other states will be mentioned as they become relevant.

The application generates the triangle vertex information and makes the necessary OpenGL calls to draw it.

The OpenGL server/library gets the vertex information, transforms, clips and lights it. It calculates the initial values and derivatives for the values to interpolate (X_{left}, X_{right}, red, green, blue and depth) for unit change in dx and $dxdy_{left}$. All these values are in fixed point integer and have unique message tags. Some of the values (the depth derivatives) have more than 32 bits to cope with the dynamic range and resolution so are sent in two halves Finally, once the derivatives, start and end values have been sent to GLINT the 'render triangle' message is sent.

On GLINT: The derivative, start and end parameter messages are received and filter down the message stream to the appropriate blocks. The depth parameters and derivatives to the Depth Unit; the RGB parameters and derivative to the Color DDA Unit; the edge values and derivatives to the Rasterizer Unit.

The 'render triangle' message is received by the rasterizer unit and all subsequent messages (from the host) are blocked until the triangle has been rasterized (but not necessarily written to the frame store). A 'prepare to render' message is passed on so any other blocks can prepare themselves.

The Rasterizer Unit walks the left and right edges of the triangle and fills in the spans between. As the walk progresses messages are send to indicate the direction of the next step: StepX or StepYDomEdge. The data field holds the current (x, y) coordinate. One message is sent per pixel within the triangle boundary. The step messages are duplicated into two groups: an active group and a passive group. The messages always start off in the active group but may be changed to the passive group if this pixel fails one of the tests (e.g. depth) on its path down the message stream. The two groups are distinguished by a single bit in the message tag. The step messages (in either form) are always passed throughout the length of the message stream, and are used by all the DDA units to keep their interpolation values in step. The step message effectively identifies the fragment and any other messages pertaining to this fragment will always precede the step message in the message stream.

The Scissor and Stipple Unit. This unit does 4 tests on the fragment (as embodied by the active step message). The screen scissor test takes the coordinates associated with the step message, converts them to be screen relative (if necessary) and compares them against the screen boundaries. The other three tests (user scissor, line stipple and area stipple) are disabled for this example. If the enabled tests pass then the active step is forwarded onto the next unit, otherwise it is changed into a passive step and then forwarded.

The Color DDA unit responds to an active step message by generating a Color message and sending this onto the next unit. The active step message is then forwarded to the next unit. The Color message holds, in the data field, the current RGBA value from the DDA. If the step message is passive then no Color message is generated. After the Color message is sent (or would have been sent) the step message is acted on to increment the DDA in the correct direction, ready for the next pixel.

Texturing, Fog and Alpha Tests Units are disabled so the messages just pass through these blocks.

In general terms the Local Buffer Read Unit reads the Graphic ID, Stencil and Depth information from the Local Buffer and passes it onto the next unit. More specifically it does:

1. If the step message is passive then no further action occurs.
2. On an active step message it calculates the linear address in the local buffer of the required data. This is done using the (X, Y) position recorded in the step message and locally stored information on the 'screen width' and window base address. Separate read and write addresses are calculated.
3. The addresses are passed to the Local Buffer Interface Unit and the identified local buffer location read. The write address is held for use later.
4. Sometime later the local buffer data is returned and is formatted into a consistent internal format and inserted into a 'Local Buffer Data' message and passed on to the next unit.

   The message data field is made wider to accommodate the maximum Local Buffer width of 52 bits (32 depth, 8 stencil, 4 graphic ID, 8 frame count) and this extra width just extends to the Local Buffer Write block.

   The actual data read from the local buffer can be in several formats to allow narrower width memories to be used in cost sensitive systems. The narrower data is formatted into a consistent internal format in this block.

The Graphic ID, Stencil and Depth Unit just passes the Color message through and stores the LBData message until the step message arrives. A passive step message would just pass straight through. When the active step message is received the internal Graphic ID, stencil and depth values are compared with the ones in the LBData message as specified by this unit's mode information. If the enabled tests pass then the new local buffer data is sent in the LBWriteData message to the next unit and the

active step message forwarded. If any of the enabled tests fail then an LBCancelWrite message is sent followed by the equivalent passive step message. The depth DDA is stepped to update the local depth value.

The Local Buffer Write Unit performs any writes which are necessary. The LBWriteData message has its data formatted into the external local buffer format and this is posted to the Local Buffer Interface Unit to be written into the memory (the write address is already waiting in the Local Buffer Interface Unit). The LBWriteCancel message just informs the Local Buffer Interface Unit that the pending write address is no longer needed and can be discarded. The step message is just passed through.

In general terms the Framebuffer Read Unit reads the color information from the framebuffer and passes it onto the next unit. More specifically it does:

1. If the step message is passive then no further action occurs.
2. On an active step message it calculates the linear address in the framebuffer of the required data. This is done using the (X, Y) position recorded in the step message and locally stored information on the 'screen width' and window base address. Separate read and write addresses are calculated.
3. The addresses are passed to the Framebuffer Interface Unit and the identified framebuffer location read. The write address is held for use later.
4. Sometime later the color data is returned and inserted into a 'Frame Buffer Data' message and passed on to the next unit.

   The actual data read from the framestore can be in several formats to allow narrower width memories to be used in cost sensitive systems. The formatting of the data is deferred until the Alpha Blend Unit as it is the only unit which needs to match it up with the internal formats. In this example no alpha blending or logical operations are taking place, so reads are disabled and hence no read address is sent to the Framebuffer Interface Unit. The Color and step messages just pass through.

The Alpha Blend Unit is disabled so just passes the messages through.

The Dither Unit stores the Color message internally until an active step is received. On receiving this it uses the least significant bits of the (X, Y) coordinate information to dither the contents of the Color message. Part of the dithering process is to convert from the internal color format into the format of the framebuffer. The new color is inserted into the Color message and passed on, followed by the step message.

The Logical Operations are disabled so the Color message is just converted into the FBWriteData message (just the tag changes) and forwarded on to the next unit. The step message just passes through.

The Framebuffer Write Unit performs any writes which are necessary.

The FBWriteData message has its data posted to the Framebuffer Interface Unit to be written into the memory (the write address is already waiting in the Framebuffer Interface Unit).

The step message is just passed through.

The Host Out Unit is mainly concerned with synchronization with the host so for this example will just consume any messages which reach this point in the message stream.

This description has concentrated on what happens as one fragment flows down the message stream. It is important to remember that at any instant in time there are many fragments flowing down the message stream and the further down they reach the more processing has occurred.

Interfacing Between Blocks FIG. 2B shows the FIFO buffering and lookahead connections which are used in the presently preferred embodiment. The FIFOs are used to provide an asynchronous interface between blocks, but are expensive in terms of gate count. Note that most of these FIFOs are only one stage deep (except where indicated), which reduces their area. To maintain performance, lookahead connections are used to accelerate the "startup" of the pipeline. For example, when the Local-Buffer-Read block issues a data request, the Texture/Fog/Color blocks also receive this, and begin to transfer data accordingly. Normally a single-entry deep FIFO cannot be read and written in the same cycle, as the writing side doesn't know that the FIFO is going to be read in that cycle (and hence become eligible to be written). The look-ahead feature give the writing side this insight, so that single-cycle transfer can be achieved. This accelerates the throughput of the pipeline.

Programming Model

The following text describes the programming model for GLINT.

GLINT as a Register file

The simplest way to view the interface to GLINT is as a flat block of memory-mapped registers (i.e. a register file). This register file appears as part of Region 0 of the PCI address map for GLINT. See the GLINT Hardware Reference Manual for details of this address map.

When a GLINT host software driver is initialized it can map the register file into its address space. Each register has an associated address tag, giving its offset from the base of the register file (since all registers reside on a 64-bit boundary, the tag offset is measured in multiples of 8 bytes). The most straightforward way to load a value into a register is to write the data to its mapped address. In reality the chip interface comprises a 16 entry deep FIFO, and each write to a register causes the written value and the register's address tag to be written as a new entry in the FIFO.

Programming GLINT to draw a primitive consists of writing initial values to the appropriate registers followed by a write to a command register. The last write triggers the start of rendering.

GLINT has approximately 200 registers. All registers are 32 bits wide and should be 32-bit addressed. Many registers are split into bit fields, and it should be noted that bit 0 is the least significant bit.

Register Types

GLINT has three main types of register:

Control Registers

Command Registers

Internal Registers

Control Registers are updated only by the host—the chip effectively uses them as read-only registers. Examples of control registers are the Scissor Clip unit min and max registers. Once initialized by the host, the chip only reads these registers to determine the scissor clip extents.

Command Registers are those which, when written to, typically cause the chip to start rendering (some command registers such as ResetPickResult or Sync do not initiate rendering). Normally, the host will initialize the appropriate control registers and then write to a command register to initiate drawing. There are two types of command registers: begin-draw and continue-draw. Begin-draw commands cause rendering to start with those values specified by the

control registers. Continue-draw commands cause drawing to continue with internal register values as they were when the previous drawing operation completed. Making use of continue-draw commands can significantly reduce the amount of data that has to be loaded into GLINT when drawing multiple connected objects such as polylines. Examples of command registers include the Render and ContinueNewLine registers.

For convenience this application will usually refer to "sending a Render command to GLINT" rather than saying (more precisely) "the Render Command register is written to, which initiates drawing".

Internal Registers are not accessible to host software. They are used internally by the chip to keep track of changing values. Some control registers have corresponding internal registers. When a begindraw command is sent and before rendering starts, the internal registers are updated with the values in the corresponding control registers. If a continue-draw command is sent then this update does not happen and drawing continues with the current values in the internal registers. For example, if a line is being drawn then the StartXDom and StartY control registers specify the (x, y) coordinates of the first point in the line. When a begin-draw command is sent these values are copied into internal registers. As the line drawing progresses these internal registers are updated to contain the (x, y) coordinates of the pixel being drawn. When drawing has completed the internal registers contain the (x, y) coordinates of the next point that would have been drawn. If a continue-draw command is now given these final (x, y) internal values are not modified and further drawing uses these values. If a begin-draw command had been used the internal registers would have been reloaded from the StartXDom and StartY registers.

For the most part internal registers can be ignored. It is helpful to appreciate that they exist in order to understand the continue-draw commands.

GLINT I/O Interface

There are a number of ways of loading GLINT registers for a given context:

- The host writes a value to the mapped address of the register
- The host writes address-tag/data pairs into a host memory buffer and uses the on-chip DMA to transfer this data to the FIFO.
- The host can perform a Block Command Transfer by writing address and data values to the FIFO interface registers.

In all cases where the host writes data values directly to the chip (via the register file) it has to worry about FIFO overflow. The InFIFOSpace register indicates how many free entries remain in the FIFO. Before writing to any register the host must ensure that there is enough space left in the FIFO. The values in this register can be read at any time. When using DMA, the DMA controller will automatically ensure that there is room in the FIFO before it performs further transfers. Thus a buffer of any size can be passed to the DMA controller.

FIFO Control

The description above considered the GLINT interface to be a register file. More precisely, when a data value is written to a register this value and the address tag for that register are combined and put into the FIFO as a new entry. The actual register is not updated until GLINT processes this entry. In the case where GLINT is busy performing a time consuming operation (e.g. drawing a large texture mapped polygon), and not draining the FIFO very quickly, it is possible for the FIFO to become full. If a write to a register is performed when the FIFO is full no entry is put into the FIFO and that write is effectively lost.

The input FIFO is **16** entries deep and each entry consists of a tag/data pair. The InFIFOSpace register can be read to determine how many entries are free. The value returned by this register will never be greater than **16**.

To check the status of the FIFO before every write is very inefficient, so it is preferably checked before loading the data for each rectangle. Since the FIFO is **16** entries deep, a further optimization is to wait for all **16** entries to be free after every second rectangle. Further optimizations can be made by moving dXDom, dXSub and dY outside the loop (as they are constant for each rectangle) and doing the FIFO wait after every third rectangle.

The InFIFOSpace FIFO control register contains a count of the number of entries currently free in the FIFO. The chip increments this register for each entry it removes from the FIFO and decrements it every time the host puts an entry in the FIFO.

The DMA Interface

Loading registers directly via the FIFO is often an inefficient way to download data to GLINT. Given that the FIFO can accommodate only a small number of entries, GLINT has to be frequently interrogated to determine how much space is left. Also, consider the situation where a given API function requires a large amount of data to be sent to GLINT. If the FIFO is written directly then a return from this function is not possible until almost all the data has been consumed by GLINT. This may take some time depending on the types of primitives being drawn.

To avoid these problems GLINT provides an on-chip DMA controller which can be used to load data from arbitrary sized (<64K 32-bit words) host buffers into the FIFO. In its simplest form the host software has to prepare a host buffer containing register address tag descriptions and data values. It then writes the base address of this buffer to the DMAAddress register and the count of the number of words to transfer to the DMACount register. Writing to the DMACount register starts the DMA transfer and the host can now perform other work. In general, if the complete set of rendering commands required by a given call to a driver function can be loaded into a single DMA buffer then the driver function can return. Meanwhile, in parallel, GLINT is reading data from the host buffer and loading it into its FIFO. FIFO overflow never occurs since the DMA controller automatically waits until there is room in the FIFO before doing any transfers.

The only restriction on the use of DMA control registers is that before attempting to reload the DMACount register the host software must wait until previous DMA has completed. It is valid to load the DMAAddress register while the previous DMA is in progress since the address is latched internally at the start of the DMA transfer.

Using DMA leaves the host free to return to the application, while in parallel, GLINT is performing the DMA and drawing. This can increase performance significantly over loading a FIFO directly. In addition, some algorithms require that data be loaded multiple times (e.g. drawing the same object across multiple clipping rectangles). Since the GLINT DMA only reads the buffer data, it can be downloaded many times simply by restarting the DMA. This can be very beneficial if composing the buffer data is a time consuming task.

The host can use this hardware capability in various ways. For example, a further optional optimization is to use a double buffered mechanism with two DMA buffers. This allows the second buffer to be filled before waiting for the

previous DMA to complete, thus further improving the parallelism between host and GLINT processing. Thus, this optimization is dependent on the allocation of the host memory. If there is only one DMA host buffer then either it is being filled or it is being emptied—it cannot be filled and emptied at the same time, since there is no way for the host and DMA to interact once the DMA transfer has started. The host is at liberty to allocate as many DMA buffers as it wants; two is the minimum to do double buffering, but allocating many small buffers is generally better, as it gives the benefits of double buffering together with low latency time, so GLINT is not idle while large buffer is being filled up. However, use of many small buffers is of course more complicated.

In general the DMA buffer format consists of a 32-bit address tag description word followed by one or more data words. The DMA buffer consists of one or more sets of these formats. The following paragraphs describe the different types of tag description words that can be used.

DMA Tag Description Format

There are 3 different tag addressing modes for DMA: hold, increment and indexed. The different DMA modes are provided to reduce the amount of data which needs to be transferred, hence making better use of the available DMA bandwidth. Each of these is described in the following sections.

Hold Format

In this format the 32-bit tag description contains a tag value and a count specifying the number of data words following in the buffer. The DMA controller writes each of the data words to the same address tag. For example, this is useful for image download where pixel data is continuously written to the Color register. The bottom 9 bits specify the register to which the data should be written; the high-order 16 bits specify the number of data words (minus 1) which follow in the buffer and which should be written to the address tag (note that the 2 -bit mode field for this format is zero so a given tag value can simply be loaded into the low order 16 bits).

A special case of this format is where the top 16 bits are zero indicating that a single data value follows the tag (i.e. the 32-bit tag description is simply the address tag value itself). This allows simple DMA buffers to be constructed which consist of tag/data pairs.

Increment Format

This format is similar to the hold format except that as each data value is loaded the address tag is incremented (the value in the DMA buffer is not changed; GLINT updates an internal copy). Thus, this mode allows contiguous GLINT registers to be loaded by specifying a single 32-bit tag value followed by a data word for each register. The low-order 9 bits specify the address tag of the first register to be loaded. The 2 bit mode field is set to 1 and the high-order 16 bits are set to the count (minus 1) of the number of registers to update. To enable use of this format, the GLINT register file has been organized so that registers which are frequently loaded together have adjacent address tags. For example, the 32 AreaStipplePattern registers can be loaded as follows:

```
AreaStipplePattern0, Count=31, Mode=1
row 0 bits
row 1 bits
...
row 31 bits
```

Indexed Format

GLINT address tags are 9 bit values. For the purposes of the Indexed DMA Format they are organized into major groups and within each group there are up to 16 tags. The low-order 4 bits of a tag give its offset within the group. The high-order 5 bits give the major group number.

The following Register Table lists the individual registers with their Major Group and Offset in the presently preferred embodiment:

Register Table

The following table lists registers by group, giving their tag values and indicating their type. The register groups may be used to improve data transfer rates to GLINT when using DMA.

The following types of register are distinguished:

Unit	Register	Major Group (hex)	Off-set (hex)	Type
Rasterizer	StartXDom	00	0	Control
	dXDom	00	1	Control
	StartXSub	00	2	Control
	dXSub	00	3	Control
	StartY	00	4	Control
	dY	00	5	Control
	Count	00	6	Control
	Render	00	7	Command
	ContinueNewLine	00	8	Command
	ContinueNewDom	00	9	Command
	ContinueNewSub	00	A	Command
	Continue	00	B	Command
	FlushSpan	00	C	Command
	BitMaskPattern	00	D	Mixed
Rasterizer	PointTable[0–3]	01	0–3	Control
	RasterizerMode	01	4	Control
Scissor Stipple	ScissorMode	03	0	Control
	ScissorMinXY	03	1	Control
	ScissorMaxXY	03	2	Control
	ScreenSize	03	3	Control
	AreaStippleMode	03	4	Control
	LineStippleMode	03	5	Control
	LoadLineStipple Counters	03	6	Control
	UpdateLineStipple Counters	03	7	Command
	SaveLineStipple State	03	8	Command
	WindowOrigin	03	9	Control
Scissor Stipple	AreaStipplePat-tern[0–31]	04 05	0-F 0-F	Control
Texture Color/Fog	Texe10	0C	0	Control
	Texe11	0C	1	Control
	Texe12	0C	2	Control
	Texe13	0C	3	Control
	Texe14	0C	4	Control
	Texe15	0C	5	Control
	Texe16	0G	6	Control
	Texe17	0C	7	Control
	Interp0	0C	8	Control
	Interp1	0C	9	Control
	Interp2	0C	A	Control
	Interp3	0C	B	Control
	Interp4	0C	C	Control
	TextureFilter	0C	D	Control
Texture/Fog Color	TextureColor Mode	0D	0	Control
	TextureEnvColor	0D	1	Control
	FogMode	0D	2	Control
	FogColor	0D	3	Control
	FStart	0D	4	Control
	dFdx	0D	5	Control
	dFdyDom	0D	6	Control
Color DDA	RStart	0F	0	Control
	dRdx	0F	1	Control
	dRdyDom	0F	2	Control
	GStart	0F	3	Control
	dGdx	0F	4	Control
	dGdyDom	0F	5	Control

-continued

Unit	Register	Major Group (hex)	Off-set (hex)	Type
	BStart	0F	6	Control
	dBdx	0F	7	Control
	dBdyDom	0F	8	Control
	AStart	0F	9	Control
	dAdx	0F	A	Control
	dAdyDom	0F	B	Control
	ColorDDAMode	0F	C	Control
	ConstantColor	0F	D	Control
	Color	0F	E	Mixed
Alpha Test	AlphaTestMode	10	0	Control
	AntialiasMode	10	1	Control
Alpha Blend	AlphaBlendMode	10	2	Control
Dither	DitherMode		3	Control
Logical Ops	FBSoftwareWrite Mask	10	4	Control
	LogicalOpMode	10	5	Control
	FBWriteData	10	6	Control
LB Read	LBReadMode	11	0	Control
	LBReadFormat	11	1	Control
	LBSourceOffset	11	2	Control
	LBStencil	11	5	Output
	LBDepth	11	6	Output
	LBWindowBase	11	7	Control
LB Write	LBWriteMode	11	8	Control
	LBWriteFormat	11	9	Control
GID/Stencil/ Depth	Window	13	0	Control
	StencilMode	13	1	Control
	StencilData	13	2	Control
	Stencil	13	3	Mixed
	DepthMode	13	4	Control
	Depth	13	5	Mixed
	ZStartU	13	6	Control
	ZStartL	13	7	Control
	dZdxU	13	8	Control
	dZdxL	13	9	Control
	dZdyDomU	13	A	Control
	dZdyDomL	13	B	Control
	FastClearDepth	13	C	Control
FB Read	FBReadMode	15	0	Control
	FBSourceOffset	15	1	Control
	FBPixelOffset	15	2	Control
	FBColor	15	3	Output
	FBWindowBase	15	6	Control
FB Write	FBWriteMode	15	7	Control
	FBHardwareWrite Mask	15	8	Control
	FBBlockColor	15	9	Control
Host Out	FilterMode	18	0	Control
	StatisticMode	18	1	Control
	MinRegion	18	2	Control
	MaxRegion	18	3	Control
	ResetPickResult	18	4	Command
	MinHitRegion	18	5	Command
	MaxHitRegion	18	6	Command
	PickResult	18	7	Command
	Sync	18	8	Command

This format allows up to 16 registers within a group to be loaded while still only specifying a single address tag description word.

If the Mode of the address tag description word is set to indexed mode, then the high-order 16 bits are used as a mask to indicate which registers within the group are to be used. The bottom 4 bits of the address tag description word are unused. The group is specified by bits 4 to 8. Each bit in the mask is used to represent a unique tag within the group. If a bit is set then the corresponding register will be loaded. The number of bits set in the mask determines the number of data words that should be following the tag description word in the DMA buffer. The data is stored in order of increasing corresponding address tag.

DMA Buffer Addresses

Host software must generate the correct DMA buffer address for the GLINT DMA controller. Normally, this means that the address passed to GLINT must be the physical address of the DMA buffer in host memory. The buffer must also reside at contiguous physical addresses as accessed by GLINT. On a system which uses virtual memory for the address space of a task, some method of allocating contiguous physical memory, and mapping this into the address space of a task, must be used.

If the virtual memory buffer maps to non-contiguous physical memory, then the buffer must be divided into sets of contiguous physical memory pages and each of these sets transferred separately. In such a situation the whole DMA buffer cannot be transferred in one go; the host software must wait for each set to be transferred. Often the best way to handle these fragmented transfers is via an interrupt handler.

DMA Interrupts

GLINT provides interrupt support, as an alternative means of determining when a DMA transfer is complete. If enabled, the interrupt is generated whenever the DMACount register changes from having a non-zero to having a zero value. Since the DMACount register is decremented every time a data item is transferred from the DMA buffer this happens when the last data item is transferred from the DMA buffer.

To enable the DMA interrupt, the DMAInterruptEnable bit must be set in the IntEnable register. The interrupt handler should check the DMAFlag bit in the IntFlags register to determine that a DMA interrupt has actually occurred. To clear the interrupt a word should be written to the IntFlags register with the DMAFlag bit set to one.

This scheme frees the processor for other work while DMA is being completed. Since the overhead of handling an interrupt is often quite high for the host processor, the scheme should be tuned to allow a period of polling before sleeping on the interrupt.

Output FIFO and Graphics Processor FIFO Interface

To read data back from GLINT an output FIFO is provided. Each entry in this FIFO is 32-bits wide and it can hold tag or data values. Thus its format is unlike the input FIFO whose entries are always tag/data pairs (we can think of each entry in the input FIFO as being 41 bits wide: 9 bits for the tag and 32 bits for the data). The type of data written by GLINT to the output FIFO is controlled by the FilterMode register. This register allows filtering of output data in various categories including the following:

- Depth: output in this category results from an image upload of the Depth buffer.
- Stencil: output in this category results from an image upload of the Stencil buffer.
- Color: output in this category results from an image upload of the framebuffer.
- Synchronization: synchronization data is sent in response to a Sync command.

The data for the FilterMode register consists of 2 bits per category. If the least significant of these two bits is set (0×1) then output of the register tag for that category is enabled; if the most significant bit is set (0×2) then output of the data for that category is enabled. Both tag and data output can be

enabled at the same time. In this case the tag is written first to the FIFO followed by the data.

For example, to perform an image upload from the framebuffer, the FilterMode register should have data output enabled for the Color category. Then, the rectangular area to be uploaded should be described to the rasterizer. Each pixel that is read from the framebuffer will then be placed into the output FIFO. If the output FIFO becomes full, then GLINT will block internally until space becomes available. It is the programmer's responsibility to read all data from the output FIFO. For example, it is important to know how many pixels should result from an image upload and to read exactly this many from the FIFO.

To read data from the output FIFO the OutputFIFOWords register should first be read to determine the number of entries in the FIFO (reading from the FIFO when it is empty returns undefined data). Then this many 32-bit data items are read from the FIFO. This procedure is repeated until all the expected data or tag items have been read. The address of the output FIFO is described below.

Note that all expected data must be read back. GLINT will block if the FIFO becomes full. Programmers must be careful to avoid the deadlock condition that will result if the host is waiting for space to become free in the input FIFO while GLINT is waiting for the host to read data from the output FIFO.

Graphics Processor FIFO Interface

GLINT has a sequence of 1K×32 bit addresses in the PCI Region 0 address map called the Graphics Processor FIFO Interface. To read from the output FIFO any address in this range can be read (normally a program will choose the first address and use this as the address for the output FIFO). All 32-bit addresses in this region perform the same function: the range of addresses is provided for data transfer schemes which force the use of incrementing addresses.

Writing to a location in this address range provides raw access to the input FIFO. Again, the first address is normally chosen. Thus the same address can be used for both input and output FIFOs. Reading gives access to the output FIFO; writing gives access to the input FIFO.

Writing to the input FIFO by this method is different from writing to the memory mapped register file. Since the register file has a unique address for each register, writing to this unique address allows GLINT to determine the register for which the write is intended. This allows a tag/data pair to be constructed and inserted into the input FIFO. When writing to the raw FIFO address an address tag description must first be written followed by the associated data. In fact, the format of the tag descriptions and the data that follows is identical to that described above for DMA buffers. Instead of using the GLINT DMA it is possible to transfer data to GLINT by constructing a DMA-style buffer of data and then copying each item in this buffer to the raw input FIFO address. Based on the tag descriptions and data written GLINT constructs tag/data pairs to enter as real FIFO entries. The DMA mechanism can be thought of as an automatic way of writing to the raw input FIFO address.

Note, that when writing to the raw FIFO address the FIFO full condition must still be checked by reading the InFIFOSpace register. However, writing tag descriptions does not cause any entries to be entered into the FIFO: such a write simply establishes a set of tags to be paired with the subsequent data. Thus, free space need be ensured only for actual data items that are written (not the tag values). For example, in the simplest case where each tag is followed by a single data item, assuming that the FIFO is empty, then 32 writes are possible before checking again for free space.

Other Interrupts

GLINT also provides interrupt facilities for the following:

Sync: If a Sync command is sent and the Sync interrupt has been enabled then once all rendering has been completed, a data value is entered into the Host Out FIFO, and a Sync interrupt is generated when this value reaches the output end of the FIFO. Synchronization is described further in the next section.

External: this provides the capability for external hardware on a GLINT board (such as an external video timing generator) to generate interrupts to the host processor.

Error: if enabled the error interrupt will occur when GLINT detects certain error conditions , such as an attempt to write to a full FIFO.

Vertical Retrace: if enabled a vertical retrace interrupt is generated at the start of the video blank period.

Each of these are enabled and cleared in a similar way to the DMA interrupt.

Synchronization

There are three main cases where the host must synchronize with GLINT:

before reading back from registers

before directly accessing the framebuffer or the local-buffer via the bypass mechanism

framebuffer management tasks such as double buffering

Synchronizing with GLINT implies waiting for any pending DMA to complete and waiting for the chip to complete any processing currently being performed. The following pseudo-code shows the general scheme:

```
GLINTData data;
// wait for DMA to complete
while (*DMACount != 0) {
  poll or wait for interrupt
}
while (*InFIFOSpace < 2) {
  ; // wait for free space in the FIFO
}
// enable sync output and send the Sync command
data.Word = 0;
data.FilterMode.Synchronization = 0x1;
FilterMode(data.Word);
Sync(0x0);
/* wait for the sync output data */
do {
  while (*OutFIFOWords == 0)
    ; // poll waiting for data in output
FIFO
} while (*OutputFIFO != Sync_tag);
```

Initially, we wait for DMA to complete as normal. We then have to wait for space to become free in the FIFO (since the DMA controller actually loads the FIFO). We need space for 2 registers: one to enable generation of an output sync value, and the Sync command itself. The enable flag can be set at initialization time. The output value will be generated only when a Sync command has actually been sent, and GLINT has then completed all processing.

Rather than polling it is possible to use a Sync interrupt as mentioned in the previous section. As well as enabling the interrupt and setting the filter mode, the data sent in the Sync command must have the most significant bit set in order to generate the interrupt. The interrupt is generated when the tag or data reaches the output end of the Host Out FIFO. Use of the Sync interrupt has to be considered carefully as GLINT will generally empty the FIFO more quickly than it takes to set up and handle the interrupt.

Host Framebuffer Bypass

Normally, the host will access the framebuffer indirectly via commands sent to the GLINT FIFO interface. However,

GLINT does provide the whole framebuffer as part of its address space so that it can be memory mapped by an application. Access to the framebuffer via this memory mapped route is independent of the GLINT FIFO.

Drivers may choose to use direct access to the framebuffer for algorithms which are not supported by GLINT. The framebuffer bypass supports big-endian, little-endian and GIB-endian formats.

A driver making use of the framebuffer bypass mechanism should synchronize framebuffer accesses made through the FIFO with those made directly through the memory map. If data is written to the FIFO and then an access is made to the framebuffer, it is possible that the framebuffer access will occur before the commands in the FIFO have been fully processed. This lack of temporal ordering is generally not desirable.

Framebuffer Dimensions and Depth

At reset time the hardware stores the size of the framebuffer in the FBMemoryControl register. This register can be read by software to determine the amount of VRAM on the display adapter. For a given amount of VRAM, software can configure different screen resolutions and off-screen memory regions.

The framebuffer width must be set up in the FBReadMode register. The first 9 bits of this register define 3 partial products which determine the offset in pixels from one scanline to the next. Typically, these values will be worked out at initialization time and a copy kept in software. When this register needs to be modified the software copy is retrieved and any other bits modified before writing to the register.

Once the offset from one scanline to the next has been established, determining the visible screen width and height becomes a clipping issue. The visible screen width and height are set up in the ScreenSize register and enabled by setting the ScreenScissorEnable bit in the ScissorMode register.

The framebuffer depth (8, 16 or 32-bit) is controlled by the FBModeSel register. This register provides a 2 bit field to control which of the three pixel depths is being used. The pixel depth can be changed at any time but this should not be attempted without first synchronizing with GLINT. The FBModeSel register is not a FIFO register and is updated immediately it is written. If GLINT is busy performing rendering operations, changing the pixel depth will corrupt that rendering.

Normally, the pixel depth is set at initialization time. To optimize certain 2D rendering operations it may be desirable to change it at other times. For example, if the pixel depth is normally 8 (or 16) bits, changing the pixel depth to 32 bits for the duration of a bitblt can quadruple (or double) the blt speed, when the bit source and destination edges are aligned on 32 bit boundaries. Once such a blt sequence has been set up the host software must wait and synchronize with GLINT and then reset the pixel depth before continuing with further rendering. It is not possible to change the pixel depth via the FIFO, thus explicit synchronization must always be used.

Host Localbuffer Bypass

As with the framebuffer, the localbuffer can be mapped in and accessed directly. The host should synchronize with GLINT before making any direct access to the localbuffer.

At reset time the hardware saves the size of the localbuffer in the LBMemoryControl register (localbuffer visible region size). In bypass mode the number of bits per pixel is either 32 or 64. This information is also set in the LBMemoryControl register (localbuffer bypass packing). This pixel packing defines the memory offset between one pixel and the next. A further set of 3 bits (localbuffer width) in the LBMemoryControl register defines the number of valid bits per pixel. A typical localbuffer configuration might be 48 bits per pixel but in bypass mode the data for each pixel starts on a 64-bit boundary. In this case valid pixel data will be contained in bits 0 to 47. Software must set the LBReadFormat register to tell GLINT how to interpret these valid bits.

Host software must set the width in pixels of each scanline of the localbuffer in the LBReadMode FIFO register. The first 9 bits of this register define 3 partial products which determine the offset in pixels from one scanline to the next. As with the framebuffer partial products, these values will usually be worked out at initialization time and a copy kept in software. When this register needs to be modified the software copy is retrieved and any other bits modified before writing to the register. If the system is set up so that each pixel in the framebuffer has a corresponding pixel in the localbuffer then this width will be the same as that set for the framebuffer.

The localbuffer is accessible via Regions 1 and 3 of the PCI address map for GLINT. The localbuffer bypass supports big-endian and little-endian formats. These are described in a later section.

Register Read Back

Under some operating environments, multiple tasks will want access to the GLINT chip. Sometimes a server task or driver will want to arbitrate access to GLINT on behalf of multiple applications. In these circumstances, the state of the GLINT chip may need to be saved and restored on each context switch. To facilitate this, the GLINT control registers can be read back. (However, internal and command registers cannot be read back.)

To perform a context switch the host must first synchronize with GLINT. This means waiting for outstanding DMA to complete, sending a Sync command and waiting for the sync output data to appear in the output FIFO. After this the registers can be read back.

To read a GLINT register the host reads the same address which would be used for a write, i.e. the base address of the register file plus the offset value for the register.

Note that since internal registers cannot be read back care must be taken when context switching a task which is making use of continue-draw commands. Continue-draw commands rely on the internal registers maintaining previous state. This state will be destroyed by any rendering work done by a new task. To prevent this, continue-draw commands should be performed via DMA since the context switch code has to wait for outstanding DMA to complete. Alternatively, continue-draw commands can be performed in a non-preemptable code segment.

Normally, reading back individual registers should be avoided. The need to synchronize with the chip can adversely affect performance. It is usually more appropriate to keep a software copy of the register which is updated when the actual register is updated.

Byte Swapping

Internally GLINT operates in little-endian mode. However, GLINT is designed to work with both big- and little-endian host processors. Since the PCIBus specification defines that byte ordering is preserved regardless of the size of the transfer operation, GLINT provides facilities to handle byte swapping. Each of the Configuration Space, Control Space, Framebuffer Bypass and Localbuffer Bypass memory areas have both big and little endian mappings available. The mapping to use typically depends on the endian ordering of the host processor.

The Configuration Space may be set by a resistor in the board design to be either little endian or big endian.

The Control Space in PCI address region 0, is 128K bytes in size, and consists of two 64K sized spaces. The first 64K provides little endian access to the control space registers; the second 64K provides big endian access to the same registers.

The framebuffer bypass consists of two PCI address regions: Region 2 and Region 4. Each is independently configurable to by the Aperture0 and Aperture 1 control registers respectively, to one of three modes: no byte swap, 16-bit swap, full byte swap. Note that the 16 bit mode is needed for the following reason. If the framebuffer is configured for 16-bit pixels and the host is big-endian then simply byte swapping is not enough when a 32-bit access is made (to write two pixels). In this case, the required effect is that the bytes are swapped within each 16-bit word, but the two 16-bit halves of the 32-bit word are not swapped. This preserves the order of the pixels that are written as well as the byte ordering within each pixel. The 16 bit mode is referred to as GIB-endian in the PCI Multimedia Design Guide, version 1.0.

The localbuffer bypass consists of two PCI address regions: Region 1 and Region 3. Each is independently configurable to by the Aperture0 and Aperture 1 control registers respectively, to one of two modes: no byte swap, full byte swap.

To save on the size of the address space required for GLINT, board vendors may choose to turn off access to the big endian regions (3 and 4) by the use of resistors on the board.

There is a bit available in the DMAControl control register to enable byte swapping of DMA data. Thus for big-endian hosts, this control bit would normally be enabled.

Red and Blue Swapping

For a given graphics board the RAMDAC and/or API will usually force a given interpretation for true color pixel values. For example, 32-bit pixels will be interpreted as either ARGB (alpha at byte 3, red at byte 2, green at byte 1 and blue at byte 0) or ABGR (blue at byte 2 and red at byte 0). The byte position for red and blue may be important for software which has been written to expect one byte order or the other, in particular when handling image data stored in a file.

GLINT provides two registers to specify the byte positions of blue and red internally. In the Alpha Blend Unit the AlphaBlendMode register contains a 1-bit field called ColorOrder. If this bit is set to zero then the byte ordering is ABGR; if the bit is set to one then the ordering is ARGB. As well as setting this bit in the Alpha Blend unit, it must also be set in the Color Formatting unit. In this unit the DitherMode register contains a Color Order bit with the same interpretation. The order applies to all of the true color pixel formats, regardless of the pixel depth.

Hardware Data Structures

Some of the hardware data structure implementations used in the presently preferred embodiment will now be described in detail. Of course these examples are provided merely to illustrate the presently preferred embodiment in great detail, and do not necessarily delimit any of the claimed inventions.

Localbuffer

The localbuffer holds the per pixel information corresponding to each displayed pixel and any texture maps. The per pixel information held in the localbuffer are Graphic ID (GID), Depth, Stencil and Frame Count Planes (FCP). The possible formats for each of these fields, and their use are covered individually in the following sections.

The maximum width of the localbuffer is 48 bits, but this can be reduced by changing the external memory configuration, albeit at the expense of reducing the functionality or dynamic range of one or more of the fields.

The localbuffer memory can be from 16 bits (assuming a depth buffer is always needed) to 48 bits wide in steps of 4 bits. The four fields supported in the localbuffer, their allowed lengths and positions are shown in the following table:

Field	Lengths	Start bit positions
Depth	16, 24, 32	0
Stencil	0, 4, 8	16, 20, 24, 28, 32
FrameCount	0, 4, 8	16, 20, 24, 28, 32, 36, 40
GID	0, 4	16, 20, 24, 28, 32, 36, 40, 44, 48

The order of the fields is as shown with the depth field at the least significant end and GID field at the most significant end. The GID is at the most significant end so that various combinations of the Stencil and FrameCount field widths can be used on a per window basis without the position of the GID fields moving. If the GID field is in a different positions in different windows then the ownership tests become impossible to do.

The GID, FrameCount, Stencil and Depth fields in the localbuffer are converted into the internal format by right justification if they are less than their internal widths, i.e. the unused bits are the most significant bits and they are set to 0.

The format of the localbuffer is specified in two places: the LBReadFormat register and the LBWriteFormat register.

It is still possible to part populate the localbuffer so other combinations of the field widths are possible (i.e. depth field width of 0), but this may give problems if texture maps are to be stored in the localbuffer as well.

Any non-bypass read or write to the localbuffer always reads or writes all 48 bits simultaneously.

GID field

The 4 bit GID field is used for pixel ownership tests to allow per pixel window clipping. Each window using this facility is assigned one of the GID values, and the visible pixels in the window have their GID field set to this value. If the test is enabled the current GID (set to correspond with the current window) is compared with the GID in the localbuffer for each fragment. If they are equal this pixel belongs to the window so the localbuffer and framebuffer at this coordinate may be updated.

Using the GID field for pixel ownership tests is optional and other methods of achieving the same result are:

clip the primitive to the window's boundary (or rectangular tiles which make up the window's area) and render only the visible parts of the primitive

use the scissor test to define the rectangular tiles which make up the window's visible area and render the primitive once per tile (This may be limited to only those tiles which the primitive intersects).

Depth Field

The depth field holds the depth (Z) value associated with a pixel and can be 16, 24 or 32 bits wide.

Stencil Field

The stencil field holds the stencil value associated with a pixel and can be 0, 4 or 8 bits wide.

The width of the stencil buffer is also stored in the StencilMode register and is needed for clamping and masking during the update methods. The stencil compare mask should be set up to exclude any absent bits from the stencil compare operation.

FrameCount Field

The Frame Count Field holds the frame count value associated with a pixel and can be 0, 4 or 8 bits wide. It is used during animation to support a fast clear mechanism to aid the rapid clearing of the depth and/or stencil fields needed at the start of each frame.

In addition to the fast clear mechanism the extent of all updates to the localbuffer and framebuffer can be recorded (MinRegion and MaxRegion registers) and read back (MinHitRegion and MaxHitRegion commands) to give the bounding box of the smallest area to clear. For some applications this will be significantly smaller than the whole window or screen, and hence faster.

The fast clear mechanism provides a method where the cost of clearing the depth and stencil buffers can be amortized over a number of clear operations issued by the application. This works as follows:

The window is divided up into n regions, where n is the range of the frame counter (16 or 256). Every time the application issues a clear command the reference frame counter is incremented (and allowed to roll over if it exceeds its maximum value) and the n^{th} region is cleared only. The clear updates the depth and/or stencil buffers to the new values and the frame count buffer with the reference value. This region is much smaller than the full window and hence takes less time to clear.

When the localbuffer is subsequently read and the frame count is found to be the same as the reference frame count (held in the Window register) the localbuffer data is used directly. However, if the frame count is found to be different from the reference frame count (held in the Window register) the data which would have been written, if the localbuffer had been cleared properly, is substituted for the stale data returned from the read. Any new writes to the localbuffer will set the frame count to the reference value so the next read on this pixel works normally without the substitution. The depth data to substitute is held in the FastClearDepth register and the stencil data to substitute is held in the StencilData register (along with other stencil information).

The fast clear mechanism does not present a total solution as the user can elect to clear just the stencil planes or just the depth planes, or both. The situation where the stencil planes only are 'cleared' using the fast clear method, then some rendering is done and then the depth planes are 'cleared' using the fast clear will leave ambiguous pixels in the localbuffer. The driver software will need to catch this situation, and fall back to using a per pixel write to do the second clear. Which field(s) the frame count plane refers to is recorded in the Window register.

When clear data is substituted for real memory data (during normal rendering operations) the depth write mask and stencil write masks are ignored to mimic the OpenGL operation when a buffer is cleared.

Localbuffer Coordinates

The coordinates generated by the rasterizer are 16 bit 2's complement numbers, and so have the range +32767 to −32768. The rasterizer will produce values in this range, but any which have a negative coordinate, or exceed the screen width or height (as programmed into the ScreenSize register) are discarded.

Coordinates can be defined window relative or screen relative and this is only relevant when the coordinate gets converted to an actual physical address in the localbuffer. In general it is expected that the windowing system will use absolute coordinates and the graphics system will use relative coordinates (to be independent of where the window really is).

GUI systems (such as Windows, Windows NT and X) usually have the origin of the coordinate system at the top left corner of the screen but this is not true for all graphics systems. For instance OpenGL uses the bottom left corner as its origin. The WindowOrigin bit in the LBReadMode register selects the top left (0) or bottom left (1) as the origin.

The actual equations used to calculate the localbuffer address to read and write are:

```
Bottom left origin:
  Destination address = LBWindowBase − Y * W + X
  Source address =
    LBWindowBase − Y*W + X + LBSourceOffset
Top left origin:
  Destination address = LBWindowBase + Y * W + X
  Source address =
    LBWindowBase + Y*W + X + LBSourceOffset
```

where:

x is the pixel's X coordinate.

Y is the pixel's Y coordinate.

LBWindowBase holds the base address in the localbuffer of the current window.

LBSourceOffset is normally zero except during a copy operation where data is read from one address and written to another address. The offset between source and destination is held in the LBSourceOffset register.

W is the screen width. Only a subset of widths are supported and these are encoded into the PP0, PP1 and PP2 fields in the LBReadMode register.

These address calculations translate a 2D address into a linear address.

The Screen width is specified as the sum of selected partial products so a full multiply operation is not needed. The partial products are selected by the fields PP0, PP1 and PP2 in the LBReadMode register.

For arbitrary width screens, for instance bitmaps in 'off screen' memory, the next largest width from the table must be chosen. The difference between the table width and the bitmap width will be an unused strip of pixels down the right hand side of the bitmap.

Note that such bitmaps can be copied to the screen only as a series of scanlines rather than as a rectangular block. However, often windowing systems store offscreen bitmaps in rectangular regions which use the same stride as the screen. In this case normal bitblts can be used.

Texture Memory

The localbuffer is used to hold textures in the GLINT 400TX variant. In the GLINT 300SX variant the texture information is supplied by the host.

Framebuffer

The framebuffer is a region of memory where the information produced during rasterization is written prior to being displayed. This information is not restricted to color but can include window control data for LUT management and double buffering.

The framebuffer region can hold up to 32 MBytes and there are very few restrictions on the format and size of the individual buffers which make up the video stream. Typical buffers include:

True color or color index main planes,

Overlay planes,

Underlay planes,

Window ID planes for LUT and double buffer management,

Cursor planes.

Any combination of these planes can be supported up to a maximum of 32 MBytes, but usually it is the video level processing which is the limiting factor. The following text examines the options and choices available from GLINT for rendering, copying, etc. data to these buffers.

To access alternative buffers either the FBPixelOffset register can be loaded, or the base address of the window held in the FBWindow-Base register can be redefined. This is described in more detail below.

Buffer Organization

Each buffer resides at an address in the framebuffer memory map. For rendering and copying operations the actual buffer addresses can be on any pixel boundary. Display hardware will place some restrictions on this as it will need to access the multiple buffers in parallel to mix the buffers together depending on their relative priority, opacity and double buffer selection. For instance, visible buffers (rather than offscreen bitmaps) will typically need to be on a page boundary.

Consider the following highly configured example with a 1280×1024 double buffered system with 32 bit main planes (RGBA), 8 bit overlay and 4 bits of window control information (WID).

Combining the WID and overlay planes in the same 32 bit pixel has the advantage of reducing the amount of data to copy when a window moves, as only two copies are required—one for the main planes and one for the overlay and WID planes.

Note the position of the overlay and WID planes. This was not an arbitrary choice but one imposed by the (presumed) desire to use the color processing capabilities of GLINT (dither and interpolation) in the overlay planes. The conversion of the internal color format to the external one stored in the framebuffer depends on the size and position of the component. Note that GLINT does not support all possible configurations. For example; if the overlay and WID bits were swapped, then eight bit color index starting at bit 4 would be required to render to the overlay, but this is not supported.

Framebuffer Coordinates

Coordinate generation for the framebuffer is similar to that for the localbuffer, but there are some key differences.

As was mentioned before, the coordinates generated by the rasterizer are 16 bit 2's complement numbers. Coordinates can be defined as window relative or screen relative, though this is only relevant when the coordinate gets converted to an actual physical address in the framebuffer. The WindowOrigin bit in the FBReadMode register selects top left (0) or bottom left (1) as the origin for the framebuffer.

The actual equations used to calculate the framebuffer address to read and write are:

```
Bottom left origin:
  Destination address = FBWindowBase - Y*W + X +
    FBPixelOffset
  Source address = FBWindowBase - Y*W + X +
    FBPixelOffset + FBSourceOffset
Top left Origin:
  Destination address = FBWindowBase + Y*W + X +
    FBPixelOffset
  Source address = FBWindowBase + Y*W + X +
    FBPixelOffset + FBSourceOffset
```

These address calculations translate a 2D address into a linear address, so non power of two framebuffer widths (i.e. 1280) are economical in memory.

The width is specified as the sum of selected partial products so a full multiply operation is not needed. The partial products are selected by the fields PP0, PP1 and PP2 in the FBReadMode register. This is the same mechanism as is used to set the width of the localbuffer, but the widths may be set independently.

For arbitrary screen sizes, for instance when rendering to 'off screen' memory such as bitmaps the next largest width from the table must be chosen. The difference between the table width and the bitmap width will be an unused strip of pixels down the right hand side of the bitmap.

Note that such bitmaps can be copied to the screen only as a series of scanlines rather than as a rectangular block. However, often windowing systems store offscreen bitmaps in rectangular regions which use the same stride as the screen. In this case normal bitblts can be used.

Color Formats

The contents of the framebuffer can be regarded in two ways:

As a collection of fields of up to 32 bits with no meaning or assumed format as far as GLINT is concerned. Bit planes may be allocated to control cursor, LUT, multi-buffer visibility or priority functions. In this case GLINT will be used to set and clear bit planes quickly but not perform any color processing such as interpolation or dithering. All the color processing can be disabled so that raw reads and writes are done and the only operations are write masking and logical ops. This allows the control planes to be updated and modified as necessary. Obviously this technique can also be used for overlay buffers, etc. providing color processing is not required.

As a collection of one or more color components. All the processing of color components, except for the final write mask and logical ops are done using the internal color format of 8 bits per red, green, blue and alpha color channels. The final stage before write mask and logical ops processing converts the internal color format to that required by the physical configuration of the framebuffer and video logic. The nomenclature n@m means this component is n bits wide and starts at bit position m in the framebuffer. The least significant bit position is 0 and a dash in a column indicates that this component does not exist for this mode. The ColorOrder is specified by a bit in the DitherMode register.

Some important points to note:

The alpha channel is always associated with the RGB color channels rather than being a separate buffer. This allows it to be moved in parallel and to work correctly in multi-buffer updates and double buffering. If the framebuffer is not configured with an alpha channel (e.g. 24 bit framebuffer width with 8:8:8:8 RGB format) then some of the rendering modes which use the retained alpha buffer cannot be used. In these cases the NoAlphaBuffer bit in the AlphaBlendMode register should be set so that an alpha value of 255 is substituted. For the RGB modes where no alpha channel is present (e.g. 3:3:2) then this substitution is done automatically.

For the Front and Back modes the data value is replicated into both buffers.

All writes to the framebuffer try to update all 32 bits irrespective of the color format. This may not matter if the memory planes don't exist, but if they are being used (as overlay planes, for example) then the write masks (FBSoftwareWriteMask or FBHardwareWriteMask) must be set up to protect the alternative planes.

When reading the framebuffer RGBA components are scaled to their internal width of 8 bits, if needed for alpha blending.

CI values are left justified with the unused bits (if any) set to zero and are subsequently processed as the red compo-

nent. The result is replicated into each of the streams G,B and A giving four copies for CI8 and eight copies for CI4.

The 4:4:4:4 Front and Back formats are designed to support 12 bit double buffering with 4 bit Alpha, in a 32 bit system.

The 3:3:2 Front and Back formats are designed to support 8 bit double buffering in a 16 bit system.

The 1:2:1 Front and Back formats are designed to support 4 bit double buffering in an 8 bit system.

It is possible to have a color index buffer at other positions as long as reduced functionality is acceptable. For example a 4 bit CI buffer at bit position 16 can be achieved using write masking and 4:4:4:4 Front format with color interpolation, but dithering is lost.

The format information needs to be stored in two places: the DitherMode register and the AlphaBlendMode register.

			Internal Color Channel			
	Format	Name	R	G	B	A
Color	0	8:8:8:8	8@0	8@8	8@16	8@24
Order:	1	5:5:5:5	5@0	5@5	5@10	5@15
RGB	2	4:4:4:4	4@0	4@4	4@8	4@12
	3	4:4:4:4	4@0	4@8	4@16	4@24
		Front	4@4	4@12	4@20	4@28
	4	4:4:4:4	4@0	4@8	4@16	4@24
		Back	4@4	4@12	4@20	4@28
	5	3:3:2	3@0	3@3	2@6	—
		Front	3@8	3@11	2@14	
	6	3:3:2	3@0	3@3	2@6	—
		Back	3@8	3@11	2@14	
	7	1:2:1	1@0	2@1	1@3	—
		Front	1@4	2@5	1@7	
	8	1:2:1	1@0	2@1	1@3	—
		Back	1@4	2@5	1@7	
Color	0	8:8:8:8	8@16	8@8	8@0	8@24
Order:	1	5:5:5:5	5@10	5@5	5@0	5@15
BGR	2	4:4:4:4	4@8	4@4	4@0	4@12
	3	4:4:4:4	4@16	4@8	4@0	4@24
		Front	4@20	4@12	4@4	4@28
	4	4:4:4:4	4@16	4@8	4@0	4@24
		Back	4@20	4@12	4@4	4@28
	5	3:3:2	3@5	3@2	2@0	—
		Front	3@13	3@10	2@8	
	6	3:3:2	3@5	3@2	2@0	—
		Back	3@13	3@10	2@8	
	7	1:2:1	1@3	2@1	1@0	—
		Front	1@7	2@5	1@4	
	8	1:2:1	1@3	2@1	1@0	—
		Back	1@7	2@5	1@4	
CI	14	CI8	8@0	0	0	0
	15	CI4	4@0	0	0	0

Overlays and Underlays

In a GUI system there are two possible relationships between the overlay planes (or underlay) and the main planes.

The overlay planes are fixed to the main planes, so that if the window is moved then both the data in the main planes and overlay planes move together.

The overlay planes are not fixed to the main planes but floating, so that moving a window only moves the associated main or overlay planes.

In the fixed case both planes can share the same GID. The pixel offset is used to redirect the reads and writes between the main planes and the overlay (underlay) buffer. The pixel ownership tests using the GID field in the localbuffer work as expected.

In the floating case different GIDs are the best choice, because the same GID planes in the localbuffer can not be used for pixel ownership tests. The alternatives are not to use the GID based pixel ownership tests for one of the buffers but rely on the scissor clipping, or to install a second set of GID planes so each buffer has it's own set. GLINT allows either approach.

If rendering operations to the main and overlay planes both need the depth or stencil buffers, and the windows in each overlap then each buffer will need its own exclusive depth and/or stencil buffers. This is easily achieved with GLINT by assigning different regions in the localbuffer to each of the buffers. Typically this would double the local-buffer memory requirements.

One scenario where the above two considerations do not cause problems, is when the overlay planes are used exclusively by the GUI system, and the main planes are used for the 3D graphics.

VRAM Modes

High performance systems will typically use VRAM for the framebuffer and the extended functionality of VRAM over DRAM can be used to enhance performance for many rendering tasks.

Hardware Write Masks.

These allow write masking in the framebuffer without incurring a performance penalty. If hardware write masks are not available, GLINT must be programmed to read the memory, merge the value with the new value using the write mask, and write it back.

To use hardware write masking, the required write mask is written to the FBHardwareWriteMask register, the FBSoftwareWriteMask register should be set to all 1's, and the number of framebuffer reads is set to 0 (for normal rendering). This is achieved by clearing the ReadSource and ReadDestination enables in the FBReadMode register.

To use software write masking, the required write mask is written to the FBSoftwareWriteMask register and the number of framebuffer reads is set to 1 (for normal rendering). This is achieved by setting the ReadDestination enable in the FBReadMode register.

Block Writes Block writes cause consecutive pixels in the framebuffer to be written simultaneously. This is useful when filling large areas but does have some restrictions:

No pixel level clipping is available;

No depth or stencil testing can be done;

All the pixels must be written with the same value so no color interpolation, blending, dithering or logical ops can be done; and

The area is defined in screen relative coordinates.

Block writes are not restricted to rectangular areas and can be used for any trapezoid. Hardware write masking is available during block writes.

The following registers need to be set up before block fills can be used:

FBBlockColor register with the value to write to each pixel; and

FBWriteMode register with the block width field.

Sending a Render command with the PrimitiveType field set to "trapezoid" and the FastFillEnable and FastFillIncrement fields set up will then cause block filling of the area. Note that during a block fill of a trapezoid any inappropriate state is ignored so even if color interpolation, depth testing and logical ops, for example, are enabled they have no effect.

The block sizes supported are 8, 16 and 32 pixels. GLINT takes care of filling any partial blocks at the end of spans.

Graphics Programming

GLINT provides a rich variety of operations for 2D and 3D graphics supported by its Pipelined architecture.

The Graphics Pipeline

This section describes each of the units in the graphics Pipeline. FIG. 2C shows a schematic of the pipeline. In this diagram, the localbuffer contains the pixel ownership values (known as Graphic IDs), the FrameCount Planes (FCP), Depth (Z) and Stencil buffer. The framebuffer contains the Red, Green, Blue and Alpha bitplanes. The operations in the Pipeline include:

Rasterizer scan converts the given primitive into a series of fragments for processing by the rest of the pipeline.

Scissor Test clips out fragments that lie outside the bounds of a user defined scissor rectangle and also performs screen clipping to stop illegal access outside the screen memory.

Stipple Test masks out certain fragments according to a specified pattern. Line and area stipples are available.

Color DDA is responsible for generating the color information (True Color RGBA or Color Index(CI)) associated with a fragment.

Texture is concerned with mapping a portion of a specified image (texture) onto a fragment. The process involves filtering to calculate the texture color, and application which applies the texture color to the fragment color.

Fog blends a fog color with a fragment's color according to a given fog factor. Fogging is used for depth cuing images and to simulate atmospheric fogging.

Antialias Application combines the incoming fragment's alpha value with its coverage value when anti aliasing is enabled.

Alpha Test conditionally discards a fragment based on the outcome of a comparison between the fragments alpha value and a reference alpha value.

Pixel Ownership is concerned with ensuring that the location in the framebuffer for the current fragment is owned by the current visual. Comparison occurs between the given fragment and the Graphic ID value in the localbuffer, at the corresponding location, to determine whether the fragment should be discarded.

Stencil Test conditionally discards a fragment based on the outcome of a test between the given fragment and the value in the stencil buffer at the corresponding location. The stencil buffer is updated dependent on the result of the stencil test and the depth test.

Depth Test conditionally discards a fragment based on the outcome of a test between the depth value for the given fragment and the value in the depth buffer at the corresponding location. The result of the depth test can be used to control the updating of the stencil buffer.

Alpha Blending combines the incoming fragment's color with the color in the framebuffer at the corresponding location.

Color Formatting converts the fragment's color into the format in which the color information is stored in the framebuffer.

This may optionally involve dithering.

The Pipeline structure of GLINT is very efficient at processing fragments, for example, texture mapping calculations are not actually performed on fragments that get clipped out by scissor testing. This approach saves substantial computational effort. The pipelined nature does however mean that when programming GLINT one should be aware of what all the pipeline stages are doing at any time. For example, many operations require both a read and/or write to the localbuffer and framebuffer; in this case it is not sufficient to set a logical operation to XOR and enable logical operations, but it is also necessary to enable the reading/writing of data from/to the framebuffer.

A Gouraud Shaded Triangle

We may now revisit the "day in the life of a triangle" example given above, and review the actions taken in greater detail. Again, the primitive being rendered will be a Gouraud shaded, depth buffered triangle. For this example assume that the triangle is to be drawn into a window which has its colormap set for RGB as opposed to color index operation. This means that all three color components; red, green and blue, must be handled. Also, assume the coordinate origin is bottom left of the window and drawing will be from top to bottom. GLINT can draw from top to bottom or bottom to top.

Consider a triangle with vertices, v_1, v_2 and v_3 where each vertex comprises X, Y and Z coordinates. Each vertex has a different color made up of red, green and blue (R, G and B) components. The alpha component will be omitted for this example.

Initialization

GLINT requires many of its registers to be initialized in a particular way, regardless of what is to be drawn, for instance, the screen size and appropriate clipping must be set up. Normally this only needs to be done once and for clarity this example assumes that all initialization has already been done.

Other state will change occasionally, though not usually on a per primitive basis, for instance enabling Gouraud shading and depth buffering.

Dominant and Subordinate Sides of a Triangle

As shown in FIG. 4A, the dominant side of a triangle is that with the greatest range of Y values. The choice of dominant side is optional when the triangle is either flat bottomed or flat topped.

GLINT always draws triangles starting from the dominant edge towards the subordinate edges. This simplifies the calculation of set up parameters as will be seen below.

These values allow the color of each fragment in the triangle to be determined by linear interpolation. For example, the red component color value of a fragment at XN, Ym could be calculated by:

adding $dRdy_{13}$, for each scanline between Y_1 and Y_n, to R_1.

then adding dRdx for each fragment along scanline Y_n from the left edge to X_n.

The example chosen has the 'knee,' i.e. vertex 2, on the right hand side, and drawing is from left to right. If the knee were on the left side (or drawing was from right to left), then the Y deltas for both the subordinate sides would be needed to interpolate the start values for each color component (and the depth value) on each scanline. For this reason GLINT always draws triangles starting from the dominant edge and towards the subordinate edges. For the example triangle, this means left to right.

Register Set Up for Color Interpolation

For the example triangle, the GLINT registers must be set as follows, for color interpolation. Note that the format for color values is 24 bit, fixed point 2's complement.

```
// Load the color start and delta values to draw
// a triangle
RStart (R1)
GStart (G1)
BStart (B1)
dRdyDom (dRdy13)        // To walk up the dominant edge
dGdyDom (dGdy13)
dBdyDom (dBdy13)
dRdx (dRdx)             // To walk along the scanline
```

-continued

```
dGdx (dGdx)
dBdx (dBdx)
```

Calculating Depth Gradient Values

To draw from left to right and top to bottom, the depth gradients or deltas) required for interpolation are:

$$dZdy_{13} = \frac{Z_3 - Z_1}{Y_3 - Y_1}$$

And from the plane equation:

$$dZdx = \left\{ (Z_1 - Z_3)\frac{(Y_2 - Y_3)}{c} \right\} - \left\{ (Z_2 - Z_3)\frac{(Y_3 - Y_1)}{c} \right\}$$

where

$$c = |(X_1 - X_3)(Y_2 - Y_3) - (X_2 - X_3)(Y_1 - Y_1)|$$

The divisor, shown here as c, is the same as for color gradient values. The two deltas $dZdyl_{13}$ and dZdx allow the Z value of each fragment in the triangle to be determined by linear interpolation, just as for the color interpolation.

Register Set Up for Depth Testing

Internally GLINT uses fixed point arithmetic. Each depth value must be converted into a 2's complement 32.16 bit fixed point number and then loaded into the appropriate pair of 32 bit registers. The 'Upper' or 'U' registers store the integer portion, whilst the 'Lower' or 'L' registers store the 16 fractional bits, left justified and zero filled.

For the example triangle, GLINT would need its registers set up as follows:

```
// Load the depth start and delta values
// to draw a triangle
ZStartU (Z1_MS)
ZStartL (Z1_LS)
dZdyDomU (dZdy13_MS)
dZdyDomL (dZdy13_LS)
dZdxU (dZdx_MS)
dZdxL (dZdx_LS)
```

Calculating the Slopes for each Side

GLINT draws filled shapes such as triangles as a series of spans with one span per scanline. Therefore it needs to know the start and end X coordinate of each span. These are determined by 'edge walking'. This process involves adding one delta value to the previous span's start X coordinate and another delta value to the previous span's end x coordinate to determine the X coordinates of the new span. These delta values are in effect the slopes of the triangle sides. To draw from left to right and top to bottom, the slopes of the three sides are calculated as:

$$dX_{13} = \frac{X_3 - X_1}{Y_3 - Y_1}$$

$$dX_{12} = \frac{X_2 - X_1}{Y_2 - Y_1}$$

$$dX_{23} = \frac{X_3 - X_2}{Y_3 - Y_2}$$

This triangle will be drawn in two parts, top down to the 'knee' (i.e. vertex 2), and then from there to the bottom. The dominant side is the left side so for the top half:

dXDom=dX_{13}

dXSub=dX_{12}

The start X,Y, the number of scanlines, and the above deltas give GLINT enough information to edge walk the top half of the triangle. However, to indicate that this is not a flat topped triangle (GLINT is designed to rasterize screen aligned trapezoids and flat topped triangles), the same start position in terms of X must be given twice as StartXDom and StartXSub.

To edge walk the lower half of the triangle, selected additional information is required. The slope of the dominant edge remains unchanged, but the subordinate edge slope needs to be set to:

dXSub=dX_{23}

Also the number of scanlines to be covered from Y_2 to Y_3 needs to be given. Finally to avoid any rounding errors accumulated in edge walking to X_2 (which can lead to pixel errors), StartXSub must be set to X_2.

Rasterizer Mode

The GLINT rasterizer has a number of modes which have effect from the time they are set until they are modified and can thus affect many primitives. In the case of the Gouraud shaded triangle the default value for these modes are suitable.

Subpixel Correction

GLINT can perform subpixel correction of all interpolated values when rendering aliased trapezoids. This correction ensures that any parameter (color/depth/texture/fog) is correctly sampled at the center of a fragment. Subpixel correction will generally always be enabled when rendering any trapezoid which is smooth shaded, textured, fogged or depth buffered. Control of subpixel correction is in the Render command register described in the next section, and is selectable on a per primitive basis.

Rasterization

GLINT is almost ready to draw the triangle. Setting up the registers as described here and sending the Render command will cause the top half of the example triangle to be drawn.

For drawing the example triangle, all the bit fields within the Render command should be set to 0 except the PrimitiveType which should be set to trapezoid and the SubPixelCorrectionEnable bit which should be set to TRUE.

```
// Draw triangle with knee
// Set deltas
StartXDom (X1<<16)  //  Converted to 16.16 fixed
point
dXDom (((X3 - X1)<<16)/(Y3 - Y1))
StartXSub (X1<<16)
dXSub (((X2 - X1)<<16)/(Y2 - Y1))
StartY (Y1<<16)
dY (-1<<16)
Count (Y1 - Y2)
// Set the render command mode
render.PrimitiveType = GLINT_TRAPEZOID_PRIMITIVE
render.SubPixelCorrectionEnable = TRUE
// Draw the top half of the triangle
Render(render)
```

After the Render command has been issued, the registers in GLINT can immediately be altered to draw the lower half of the triangle. Note that only two registers need be loaded and the command ContinueNewSub sent. Once GLINT has received ContinueNewSub, drawing of this sub-triangle will begin.

```
// Setup the delta and start for the new edge
StartXSub (X2<<16)
dXSub (((X3 - X2)<<16)/(Y3 - Y2))
// Draw sub-triangle
ContinueNewSub (Y2 - Y3)  // Draw lower half
```

Rasterizer Unit

The rasterizer decomposes a given primitive into a series of fragments for processing by the rest of the Pipeline.

GLINT can directly rasterize:

aliased screen aligned trapezoids

aliased single pixel wide lines

aliased single pixel points

antialiased screen aligned trapezoids

antialiased circular points

All other primitives are treated as one or more of the above, for example an antialiased line is drawn as a series of antialiased trapezoids.

Trapezoids GLINT's basic area primitives are screen aligned trapezoids. These are characterized by having top and bottom edges parallel to the X axis. The side edges may be vertical (a rectangle), but in general will be diagonal. The top or bottom edges can degenerate into points in which case we are left with either flat topped or flat bottomed triangles. Any polygon can be decomposed into screen aligned trapezoids or triangles. Usually, polygons are decomposed into triangles because the interpolation of values over non-triangular polygons is ill defined. The rasterizer does handle flat topped and flat bottomed 'bow tie' polygons which are a special case of screen aligned trapezoids.

To render a triangle, the approach adopted to determine which fragments are to be drawn is known as 'edge walking'. Suppose the aliased triangle shown in FIG. **4A** was to be rendered from top to bottom and the origin was bottom left of the window. Starting at (X1, Y1) then decrementing Y and using the slope equations for edges **1–2** and **1–3**, the intersection of each edge on each scanline can be calculated. This results in a span of fragments per scanline for the top trapezoid. The same method can be used for the bottom trapezoid using slopes **2–3** and **1–3**.

It is usually required that adjacent triangles or polygons which share an edge or vertex are drawn such that pixels which make up the edge or vertex get drawn exactly once. This may be achieved by omitting the pixels down the left or the right sides and the pixels along the top or lower sides. GLINT has adopted the convention of omitting the pixels down the right hand edge. Control of whether the pixels along the top or lower sides are omitted depends on the start Y value and the number of scanlines to be covered. With the example, if StartY =Y1 and the number of scanlines is set to Y1–Y2, the lower edge of the top half of the triangle will be excluded. This excluded edge will get drawn as part of the lower half of the triangle.

To minimize delta calculations, triangles may be scan converted from left to right or from right to left. The direction depends on the dominant edge, that is the edge which has the maximum range of Y values. Rendering always proceeds from the dominant edge towards the relevant subordinate edge. In the example above, the dominant edge is **1–3** so rendering will be from right to left.

The sequence of actions required to render a triangle (with a 'knee') is:

Load the edge parameters and derivatives for the dominant edge and the first subordinate edges in the first triangle.

Send the Render command. This starts the scan conversion of the first triangle, working from the dominant edge. This means that for triangles where the knee is on the left we are scanning right to left, and vice versa for triangles where the knee is on the right.

Load the edge parameters and derivatives for the remaining subordinate edge in the second triangle.

Send the ContinueNewSub command. This starts the scan conversion of the second triangle.

Pseudocode for the above example is:

```
// Set the rasterizer mode to the default
RasterizerMode (0)
// Setup the start values and the deltas.
// Note that the X and Y coordinates are converted
// to 16.16 format
StartXDom (X1<<16)
dXDom (((X3- X1)<<16)/(Y3 - Y1))
StartXSub (X1<<16)
dXSub (((X2- X1)<<16)/(Y2 - Y1))
StartY (Y1<<16)
dY (-1<16) // Down the screen
Count (Y1 - Y2)
// Set the render mode to aliased primitive with
// subpixel correction.
render.PrimitiveType = GLINT_TRAPEZOID_PRIMITIVE
render.SubpixelCorrectionEnable = GLINT_TRUE
render.AntialiasEnable = GLINT_DISABLE
// Draw top half of the triangle
Render(render)
// Set the start and delta for the second half of
// the triangle.
StartXSub (X2<<16)
dXSub (((X3- X2)<<16)/(Y3 - Y2))
// Draw lower half of triangle
ContinueNewSub (abs(Y2 - Y3))
```

After the Render command has been sent, the registers in GLINT can immediately be altered to draw the second half of the triangle. For this, note that only two registers need be loaded and the command ContinueNewSub be sent. Once drawing of the first triangle is complete and GLINT has received the ContinueNewSub command, drawing of this sub-triangle will start. The ContinueNewSub command register is loaded with the remaining number of scanlines to be rendered.

Lines

Single pixel wide aliased lines are drawn using a DDA algorithm, so all GLINT needs by way of input data is StartX, StartY, dX, dY and length.

For polylines, a ContinueNewLine command (analogous to the Continue command used at the knee of a triangle) is used at vertices.

When a Continue command is issued some error will be propagated along the line. To minimize this, a choice of actions are available as to how the DDA units are restarted on the receipt of a Continue command. It is recommended that for OpenGL rendering the ContinueNewLine command is not used and individual segments are rendered.

Antialiased lines, of any width, are rendered as antialiased screen-aligned trapezoids.

Points

GLINT supports a single pixel aliased point primitive. For points larger than one pixel trapezoids should be used. In this case the PrimitiveType field in the Render command should be set to equal GLINT_POINT_PRIMITIVE.

Anti aliasing

GLINT uses a subpixel point sampling algorithm to antialias primitives. GLINT can directly rasterize antialiased trapezoids and points. Other primitives are composed from these base primitives.

The rasterizer associates a coverage value with each fragment produced when antialiasing. This value represents the percentage coverage of the pixel by the fragment. GLINT supports two levels of antialiasing quality:

normal, which represents 4×4 pixel subsampling

high, which represents 8×8 pixel subsampling.

Selection between these two is made by the AntialiasingQuality bit within the Render command register.

When rendering antialiased primitives with GLINT the FlushSpan command is used to terminate rendering of a primitive. This is due to the nature of GLINT antialiasing. When a primitive is rendered which does not happen to complete on a scanline boundary, GLINT retains antialiasing information about the last sub-scanline(s) it has processed, but does not generate fragments for them unless a FlushSpan command is received. The commands ContinueNewSub, ContinueNewDom or Continue can then be used, as appropriate, to maintain continuity between adjacent trapezoids. This allows complex antialiased primitives to be built up from simple trapezoids or points.

To illustrate this consider using screen aligned trapezoids to render an antialiased line. The line will in general consist of three screen aligned trapezoids as shown in FIG. 4B. This FIG. illustrates the sequence of rendering an Antialiased Line primitive. Note that the line has finite width.

The procedure to render the line is as follows:

```
// Setup the blend and coverage application units
// as appropriate - not shown
// In this example only the edge deltas are shown
// loaded into registers for clarity. In reality
// start X and Y values are required
// Render Trapezoid A
dY(1<<16)
dXDom(dXDom1<<16)
dXSub(dXSub1<<16)
Count(count1)
render.PrimitiveType = GLINT_TRAPEZOID
remder.AntialiasEnable = GLINT_TRUE
render.AntialiasQuality = GLINT_MIN_ANTIALIAS
render.CoverageEnable = GLINT_TRUE
Render(render)
// Render Trapezoid B
dXSub(dXSub2<<16)
ContinueNewSub(count2)
// Render Trapezoid C
dXDom(dXDom2<<16)
ContinueNewDom(count3)
// Now we have finished the primitive flush out
// the last scanline
FlushSpan( )
```

Note that when rendering antialiased primitives, any count values should be given in subscanlines, for example if the quality is 4×4 then any scanline count must be multiplied by 4 to convert it into a subscanline count. Similarly, any delta value must be divided by 4.

When rendering, AntialiasEnable must be set in the Antialias-Mode register to scale the fragments color by the coverage value. An appropriate blending function should also be enabled.

Note, when rendering antialiased bow-ties, the coverage value on the cross-over scanline may be incorrect.

GLINT can render small antialiased points. Antialiased points are treated as circles, with the coverage of the boundary fragments ranging from 0% to 100%. GLINT supports:

point radii of 0.5 to 16.0 in steps of 0.25 for 4×4 antialiasing

point radii of 0.25 to 8.0 in steps of 0.125 for 8×8 antialiasing

To scan convert an antialiased point as a circle, GLINT traverses the boundary in sub scanline steps to calculate the coverage value. For this, the sub-scanline intersections are calculated incrementally using a small table. The table holds the change in X for a step in Y. Symmetry is used so the table only holds the delta values for one quadrant.

StartXDom, StartXSub and StartY are set to the top or bottom of the circle and dY set to the subscanline step. In the case of an even diameter, the last of the required entries in the table is set to zero.

Since the table is configurable, point shapes other than circles can be rendered. Also if the StartXDom and StartXSub values are not coincident then horizontal thick lines with rounded ends, can be rendered.

Block Write Operation

GLINT supports VRAM block writes with block sizes of 8, 16 and 32 pixels. The block write method does have some restrictions: None of the per pixel clipping, stipple, or fragment operations are available with the exception of write masks. One subtle restriction is that the block coordinates will be interpreted as screen relative and not window relative when the pixel mask is calculated in the Framebuffer Units.

Any screen aligned trapezoid can be filled using block writes, not just rectangles.

The use of block writes is enabled by setting the FastFillEnable and FastFillIncrement fields in the Render command register. The framebuffer write unit must also be configured.

Note only the Rasterizer, Framebuffer Read and Framebuffer Write units are involved in block filling. The other units will ignore block write fragments, so it is not necessary to disable them.

Sub Pixel Precision and Correction

As the rasterizer has 16 bits of fraction precision, and the screen width used is typically less than 2^{16} wide a number of bits called subpixel precision bits, are available. Consider a screen width of 4096 pixels. This figure gives a subpixel precision of 4 bits ($4096=2^{12}$). The extra bits are required for a number of reasons:

antialiasing (where vertex start positions can be supplied to subpixel precision)

when using an accumulation buffer (where scans are rendered multiple times with jittered input vertices)

for correct interpolation of parameters to give high quality shading as described below

GLINT supports subpixel correction of interpolated values when rendering aliased trapezoids. Subpixel correction ensures that all interpolated parameters associated with a fragment (color, depth, fog, texture) are correctly sampled at the fragment's center. This correction is required to ensure consistent shading of objects made from many primitives. It should generally be enabled for all aliased rendering which uses interpolated parameters.

Subpixel correction is not applied to antialiased primitives.

Bitmaps

A Bitmap primitive is a trapezoid or line of ones and zeros which control which fragments are generated by the rasterizer. Only fragments where the corresponding Bitmap bit is set are submitted for drawing. The normal use for this is in drawing characters, although the mechanism is available for all primitives. The Bitmap data is packed contiguously into 32 bit words so that rows are packed adjacent to each other. Bits in the mask word are by default used from the least significant end towards the most significant end and are applied to pixels in the order they are generated in.

The rasterizer scans through the bits in each word of the Bitmap data and increments the X,Y coordinates to trace out the rectangle of the given width and height. By default, any set bits (1) in the Bitmap cause a fragment to be generated, any reset bits (0) cause the fragment to be rejected.

The selection of bits from the BitMaskPattern register can be mirrored, that is, the pattern is traversed from MSB to LSB rather than LSB to MSB. Also, the sense of the test can be reversed such that a set bit causes a fragment to be rejected and vice versa. This control is found in the RasterizerMode register.

When one Bitmap word has been exhausted and pixels in the rectangle still remain then rasterization is suspended until the next write to the BitMaskPattern register. Any unused bits in the last Bitmap word are discarded.

Image Copy/Upload/Download

GLINT supports three "pixel rectangle" operations: copy, upload and download. These can apply to the Depth or Stencil Buffers (held within the localbuffer) or the framebuffer.

It should be emphasized that the GLINT copy operation moves RAW blocks of data around buffers. To zoom or re-format data, in the presently preferred embodiment, external software must upload the data, process it and then download it again.

To copy a rectangular area, the rasterizer would be configured to render the destination rectangle, thus generating fragments for the area to be copied. GLINT copy works by adding a linear offset to the destination fragment's address to find the source fragment's address.

Note that the offset is independent of the origin of the buffer or window, as it is added to the destination address. Care must be taken when the source and destination overlap to choose the source scanning direction so that the overlapping area is not overwritten before it has been moved. This may be done by swapping the values written to the StartXDom and StartXSub, or by changing the sign of dY and setting StartY to be the opposite side of the rectangle.

Localbuffer copy operations are correctly tested for pixel ownership. Note that this implies two reads of the localbuffer, one to collect the source data, and one to get the destination GID for the pixel ownership test.

GLINT buffer upload/downloads are very similar to copies in that the region of interest is generated in the rasterizer. However, the localbuffer and framebuffer are generally configured to read or to write only, rather than both read and write. The exception is that an image load may use pixel ownership tests, in which case the localbuffer destination read must be enabled.

Units which can generate fragment values, the color DDA unit for example, should generally be disabled for any copy/upload/download operations.

Warning: During image upload, all the returned fragments must be read from the Host Out FIFO, otherwise the GLINT pipeline will stall. In addition it is strongly recommended that any units which can discard fragments (for instance the following tests: bitmask, alpha, user scissor, screen scissor, stipple, pixel ownership, depth, stencil), are disabled otherwise a shortfall in pixels returned may occur, also leading to deadlock.

Note that because the area of interest in copy/upload/download operations is defined by the rasterizer, it is not limited to rectangular regions.

Color formatting can be used when performing image copies, uploads and downloads. This allows data to be formatted from, or to, any of the supported GLINT color formats.

Rasterizer Mode

A number of long-term modes can be set using the Rasterizer-Mode register, these are:

Mirror BitMask: This is a single bit flag which specifies the direction bits are checked in the BitMask register. If the bit is reset, the direction is from least significant to most significant (bit **0** to bit **31**), if the bit is set, it is from most significant to least significant (from bit **31** to bit **0**).

Invert BitMask: This is a single bit which controls the sense of the accept/reject test when using a Bitmask. If the bit is reset then when the BitMask bit is set the fragment is accepted and when it is reset the fragment is rejected. When the bit is set the sense of the test is reversed.

Fraction Adjust: These 2 bits control the action taken by the rasterizer on receiving a ContinueNewLine command. As GLINT uses a DDA algorithm to render lines, an error accumulates in the DDA value. GLINT provides for greater control of the error by doing one of the following:

leaving the DDA running, which means errors will be propagated along a line.

or setting the fraction bits to either zero, a half or almost a half (0×7FFF).

Bias Coordinates: Only the integer portion of the values in the DDAs are used to generate fragment addresses. Often the actual action required is a rounding of values, this can be achieved by setting the bias coordinate bit to true which will automatically add almost a half (0×7FFF) to all input coordinates.

Rasterizer Unit Registers

Real coordinates with fractional parts are provided to the rasterizer in 2's complement 16 bit integer, 16 bit fraction format. The following Table lists the command registers which control the rasterizer unit:

Register Name	Description
Render	Starts the rasterization process
ContinueNewDom	Allows the rasterization to continue with a new dominant edge. The dominant edge DDA is reloaded with the new parameters. The subordinate edge is carried on from the previous trapezoid. This allows any convex polygon to be broken down into a collection of trapezoids, with continuity maintained across boundaries. The data field holds the number of scanlines (or sub scanlines) to fill. Note this count does not get loaded into the Count register.
ContinueNewSub	Allows the rasterization to continue with a new subordinate edge. The subordinate DDA is reloaded with the new

-continued

Register Name	Description
	parameters. The dominant edge is carried on from the previous trapezoid. This is useful when scan converting triangles with a 'knee' (i.e. two subordinate edges). The data field holds the number of scanlines (or sub scanlines) to fill. Note this count does not get loaded into the Count register.
Continue	Allows the rasterization to continue after new delta value(s) have been loaded, but does not cause either of the trapezoid's edge DDAs to be reloaded. The data field holds the number of scanlines (or sub scanlines) to fill. Note this count does not get loaded into the Count register.
ContinueNewLine	Allows the rasterization to continue for the next segment in a polyline. The XY position is carried on from the previous line, but the fraction bits in the DDAs can be: kept, set to zero, half, or nearly one half, under control of the RasterizerMode. The data field holds the number of scanlines to fill. Note this count does not get loaded into the Count register. The use of ContinueNewLine is not recommended for OpenGL because the DDA units will start with a slight error as compared with the value they would have been loaded with for the second and subsequent segments.
FlushSpan	Used when antialiasing to force the last span out when not all sub spans may be defined.

The following Table shows the control registers of the rasterizer, in the presently preferred embodiment:

RasterizerMode	Defines the long term mode of operation of the rasterizer.
StartXDom	Initial X value for the dominant edge in trapezoid filling, or initial X value in line drawing.
dXDom	Value added when moving from one scanline (or sub scanline) to the next for the dominant edge in trapezoid filling. Also holds the change in X when plotting lines so for Y major lines this will be some fraction (dx/dy), otherwise it is normally ± 1.0, depending on the required scanning direction.
StartXSub	Initial X value for the subordinate edge.
dXSub	Value added when moving from one scanline (or sub scanline) to the next for the subordinate edge in trapezoid filling.
StartY	Initial scanline (or sub scanline) in trapezoid filling, or initial Y position for line drawing.
dY	Value added to Y to move from one scanline to the next. For X major lines this will be some fraction (dy/dx), otherwise it is normally ± 1.0, depending on the required scanning direction.
Count	Number of pixels in a line. Number of scanlines in a trapezoid. Number of sub scanlines in an antialiased trapezoid. Diameter of a point in sub scanlines.
BitMaskPattern	Value used to control the BitMask stipple operation (if enabled).
PointTable0 PointTable1 PointTable2 PointTable3	Antialias point data table. There are 4 words in the table and the register tag is decoded to select a word.

For efficiency, the Render command register has a number of bit fields that can be set or cleared per render operation, and which qualify other state information within GLINT. These bits are AreaStippleEnable, LineStippleEnable, ResetLineStipple, TextureEnable FogEnable, CoverageEnable and SubpixelCorrection.

One use of this feature can occur when a window is cleared to a background color. For normal 3D primitives, stippling and fog operations may have been enabled, but these are to be ignored for window clears. Initially the FogMode, AreaStippleMode and LineStippleMode registers are enabled through the UnitEnable bits. Now bits need only be set or cleared within the Render command to achieve the required result, removing the need for the FogMode, AreaStippleMode and LineStippleMode registers to be loaded for every render operation.

The bitfields of the Render command register, in the presently preferred embodiment, are detailed below:

Bit	Name	Description
0	Area-Stipple-Enable	This bit, when set, enables area stippling of the fragments produced during rasterization. Note that area stipple in the Stipple Unit must be enabled as well for stippling to occur. When this bit is reset no area stippling occurs irrespective of the setting of the area stipple enable bit in the Stipple Unit. This bit is useful to temporarily force no area stippling for this primitive.
1	Line-Stipple-Enable	This bit, when set, enables line stippling of the fragments produced during rasterization in the Stipple Unit. Note that line stipple in the Stipple Unit must be enabled as well for stippling to occur. When this bit is reset no line stippling occurs irrespective of the setting if the line stipple enable bit in the Stipple Unit. This bit is useful to temporarily force no line stippling for this primitive.
2	Reset-Line-Stipple	This bit, when set, causes the line stipple counters in the Stipple Unit to be reset to zero, and would typically be used for the first segment in a polyline. This action is also qualified by the LineStippleEnable bit and also the stipple enable bits in the Stipple Unit. When this bit is reset the stipple counters carry on from where they left off (if line stippling is enabled)
3	FastFillEnable	This bit, when set, causes fast block filling of primitives. When this bit is reset the normal rasterization process occurs.
4, 5	Fast-Fill-Increment	This two bit field selects the block size the framebuffer supports. The sizes supported and the corresponding codes are: 0 = 8 pixels 1 = 16 pixels 2 = 32 pixels
6, 7	Primitive-Type	This two bit field selects the primitive type to rasterize. The primitives are: 0 = Line 1 = Trapezoid 2 = Point
8	Antialias-Enable	This bit, when set, causes the generation of sub scanline data and the coverage value to be calculated for each fragment. The number of sub pixel samples to use is controlled by the AntialiasingQuality bit. When this bit is reset normal rasterization occurs.
9	An-tialiasing-Quality	This bit, when set, sets the sub pixel resolution to be 8 × 8 When this bit is reset the sub pixel resolution is 4 × 4.
10	UsePoint-Table	When this bit and the AntialiasingEnable are set, the dx values used to remove from one scanline to the next are derived from the Point Table.
11	SyncOn-BitMask	This bit, when set, causes a number of actions: - The least significant bit or most significant bit (depending on the MirrorBitMask bit) in the Bit Mask register is extracted and optionally inverted (controlled by the InvertMask bit). If this bit is 0 then the corresponding fragment is culled from being drawn. After every fragrant the Bit Mask register is rotated by one bit. If all the bits in the Bit Mask register have been used then rasterization is suspended until a new BitMaskPattern is received. If any other register is written while the rasterization is suspended then the rasterization is aborted. The register write which caused the abort is then processed as normal. Note the behavior is slightly different when the SyncOnHostData bit is set to prevent a deadlock from occurring. In this case the rasterization doesn't suspend when all the bits have been used and if new BitMaskPattern data words are not received in a timely manner then the subsequent fragments will just reuse the bitmask.
12	SyncOn HostData	When this bit is set a fragment is produced only when one of the following registers has been written by the host: Depth, FBColor, Stencil or Color. If SyncOnBitMask is reset, then if any register other than one of these four is written to, the rasterization is aborted. If SyncOnBitMask is set, then if any register other than one of these four, or BitMaskPattern, is written to, the rasterization is aborted. The register write which caused the abort is then processed as normal. Writing to the BitMaskPattern register doesn't cause any fragments to be generated, but just updates the BitMask register.
13	TextureEnable	This bit, when set, enables texturing of the fragments produced during rasterization. Note that the Texture Units must be suitably enabled as well for any texturing to occur.

-continued

Bit	Name	Description
		When this bit is reset no texturing occurs irrespective of the setting of the Texture Unit controls. This bit is useful to temporarily force no texturing for this primitive.
14	Fog-Enable	This bit, When set, enables fogging of the fragments produced during rasterization. Note that the Fog Unit must be suitably enabled as well for any fogging to occur. When this bit is reset no fogging occurs irrespective of the setting of the Fog Unit controls. This bit is useful to temporarily force no fogging for this primitive.
15	Coverage-Enable	This bit, when set, enables the coverage value produced as part of the antialiasing to weight the alpha value in the alpha test unit. Note that this unit must be suitably enabled as well. When this bit is reset no coverage application. occurs irrespective of the setting of the AntialiasMode in the Alpha. Test unit.
16	SubPixel-Correction Enable	This bit, when set enables the sub pixel correction of the color, depth, fog and texture values at the start of a scanline. When this bit is reset no correction is done at the start of a scanline. Sub pixel corrections are only applied to aliased trapezoids.

A number of long-term rasterizer modes are stored in the RasterizerMode register as shown below:

Bit	Name	Description
0	Mirror-BitMask	When this bit is set the bitmask bits are consumed from the most significant end towards the least significant end. When this bit is reset the bitmask bits are consumed from the least significant end towards the most significant end.
1	InvertBit-Mask	When this bit is set the bitmask is inverted first before being tested.
2,3	Fraction-Adjust	These bits control the action of a ContinueNewLine command and specify how the fraction bits in the Y and XDom DDAs are adjusted 0: No adjustment is done 1: Set the fraction bits to zero 2: Set the fraction bits to half 3: Set the fraction to nearly half, i.e. 0x7fff
4,5	BiasCoor-dinates	These bits control how much is added onto the StartXDom, StartXSub and StartY values, when they are loaded into the DDA units. The original registers are not affected: 0: Zero is added 1: Half is added 2: Nearly half, i.e. 0x7fff is added

Scissor Unit

Two scissor tests are provided in GLINT, the User Scissor test and the Screen Scissor test. The user scissor checks each fragment against a user supplied scissor region; the screen scissor checks that the fragment lies within the screen.

This test may reject fragments if some part of a window has been moved off the screen. It will not reject fragments if part of a window is simply overlapped by another window (GID testing can be used to detect this).

Stipple Unit

Stippling is a process whereby each fragment is checked against a bit in a defined pattern, and is rejected or accepted depending on the result of the stipple test. If it is rejected it undergoes no further processing; otherwise it proceeds down the pipeline. GLINT supports two types of stippling, line and area.

Area Stippling

A 32×32 bit area stipple pattern can be applied to fragments. The least significant n bits of the fragment's (X,Y) coordinates, index into a 2D stipple pattern. If the selected bit in the pattern is set, then the fragment passes the test, otherwise it is rejected. The number of address bits used, allow regions of 1,2,4,8,16 and 32 pixels to be stippled. The address selection can be controlled independently in the X and Y directions. In addition the bit pattern can be inverted or mirrored. Inverting the bit pattern has the effect of changing the sense of the accept/reject test. If the mirror bit is set the most significant bit of the pattern is towards the left of the window, the default is the converse.

In some situations window relative stippling is required but coordinates are only available screen relative. To allow window relative stippling, an offset is available which is added to the coordinates before indexing the stipple table. X and Y offsets can be controlled independently.

Line Stippling

In this test, fragments are conditionally rejected on the outcome of testing a linear stipple mask. If the bit is zero then the test fails, otherwise it passes. The line stipple pattern is 16 bits in length and is scaled by a repeat factor r (in the range 1 to 512). The stipple mask bit b which controls the acceptance or rejection of a fragment is determined using:

$$b=(\text{floor}(s/r)) \bmod 16$$

where s is the stipple counter which is incremented for every fragment (normally along the line). This counter may be reset at the start of a polyline, but between segments it continues as if there were no break.

The stipple pattern can be optionally mirrored, that is the bit pattern is traversed from most significant to least significant bits, rather than the default, from least significant to most significant.

Color DDA Unit

The color DDA unit is used to associate a color with a fragment produced by the rasterizer. This unit should be enabled for rendering operations and disabled for pixel rectangle operations (i.e. copies, uploads and downloads). Two color modes are supported by GLINT, true color RGBA and color index (CI).

Gouraud Shading

When in Gouraud shading mode, the color DDA unit performs linear interpolation given a set of start and increment values. Clamping is used to ensure that the interpolated value does not underflow or overflow the permitted color range.

For a Gouraud shaded trapezoid, GLINT interpolates from the dominant edge of a trapezoid to the subordinate edges. This means that two increment values are required per color component, one to move along the dominant edge and one to move across the span to the subordinate edge.

Note that if one is rendering to multiple buffers and has initialized the start and increment values in the color DDA unit, then any subsequent Render command will cause the start values to be reloaded.

If subpixel correction has been enabled for a primitive, then any correction required will be applied to the color components.

Flat Shading

In flat shading mode, a constant color is associated with each fragment. This color is loaded into the ConstantColor register.

Texture Unit

The texture unit combines the incoming fragment's color (generated in the color DDA unit) with a value derived from interpolating texture map values (texels).

Texture application consists of two stages; derivation of the texture color from the texels (a filtering process) and then application of the texture color to the fragment's color, which is dependent on the application mode (Decal, Blend or Modulate).

GLINT 300SX compared with the GLINT 400TX

Both the GLINT 300SX and GLINT 300TX support all the filtering and application modes described in this section. However, when using the GLINT 300SX, texel values, interpolants and texture filter selections are supplied by the host. This implies that texture coordinate interpolation and texel extraction are performed by the host using texture maps resident on the host. The recommended technique for performing texture mapping using the GLINT 300SX is to scan convert primitives on the host and render fragments as GLINT point primitives.

The GLINT 400TX automatically generates all data required for texture application as textures are stored in the localbuffer and texture parameter interpolation with full perspective correction takes place within the processor. Thus the GLINT 400TX is the processor of choice when full texture mapping acceleration is desired, the GLINT 300SX is more suitable in applications where the performance of texture mapping is not critical.

Texture Color Generation.

Texture color generation supports all the filter modes of OpenGL, that is:

Minification:
- Nearest
- Linear
- NearestMipMapNearest
- NearestMipMapLinear
- LinearMipMapNearest
- LinearMipMapLinear

Magnification:
- Nearest
- Linear

Minification is the name given to the filtering process used whereby multiple texels map to a fragment, while magnification is the name given to the filtering process whereby only a portion of a single texel maps to a single fragment.

Nearest is the simplest form of texture mapping where the nearest texel to the sample location is selected with no filtering applied.

Linear is a more sophisticated algorithm which is dependent on the type of primitive. For lines (which are 1D), it involves linear interpolation between the two nearest texels, for polygons and points which are considered to have finite area, linear is in fact bi-linear interpolation which interpolates between the nearest 4 texels.

Mip Mapping is a technique to allow the efficient filtering of texture maps when the projected area of the fragment covers more than one texel (ie. minification). A hierarchy of texture maps is held with each one being half the size (or one quarter the area) of the preceding one. A pair of maps are selected, based on the projected area of the texture. In terms of filtering this means that three filter operations are performed: one on the first map, one on the second map and one between the maps. The first filter name (Nearest or Linear) in the MipMap name specifies the filtering to do on the two maps, and the second filter name specifies the filtering to do between maps. So for instance, linear mapping between two maps, with linear interpolation between the results is supported (LinearMipMapLinear), but linear interpolation on one map, nearest on the other map, and linear interpolation between the two is not supported.

The filtering process takes a number of texels and interpolants, and with the current texture filter mode produces a texture color.

Fog Unit

The fog unit is used to blend the incoming fragment's color (generated by the color DDA unit, and potentially modified by the texture unit) with a predefined fog color. Fogging can be used to simulate atmospheric fogging, and also to depth cue images.

Fog application has two stages; derivation of the fog index for a fragment, and application of the fogging effect. The fog index is a value which is interpolated over the primitive using a DDA in the same way color and depth are interpolated. The fogging effect is applied to each fragment using one of the equations described below.

Note that although the fog values are linearly interpolated over a primitive the fog values can be calculated on the host using a linear fog function (typically for simple fog effects and depth cuing) or a more complex function to model atmospheric attenuation. This would typically be an exponential function.

Fog Index Calculation—The Fog DDA

The fog DDA is used to interpolate the fog index (f) across a primitive. The mechanics are similar to those of the other DDA units, and horizontal scanning proceeds from dominant to subordinate edge as discussed above.

The DDA has an internal range of approximately +511 to −512, so in some cases primitives may exceed these bounds. This problem typically occurs for very large polygons which span the whole depth of a scene. The correct solution is to tessellate the polygon until polygons lie within the acceptable range, but the visual effect is frequently negligible and can often be ignored.

The fog DDA calculates a fog index value which is clamped to lie in the range 0.0 to 1.0 before it is used in the appropriate fogging equation. (Fogging is applied differently depending on the color mode.)

Antialias Application Unit

Antialias application controls the combining of the coverage value generated by the rasterizer with the color generated in the color DDA units. The application depends on the color mode, either RGBA or Color Index (CI).

Antialias Application

When antialiasing is enabled this unit is used to combine the coverage value calculated for each fragment with the fragment's alpha value. In RGBA mode the alpha value is multiplied by the coverage value calculated in the rasterizer (its range is 0% to 100%). The RGB values remain unchanged and these are modified later in the Alpha Blend unit which must be set up appropriately. In CI mode the coverage value is placed in the lower 4 bits of the color field.

The Color Look Up Table is assumed to be set up such that each color has 16 intensities associated with it, one per coverage entry.

Polygon Antialiasing

When using GLINT to render antialiased polygons, depth buffering cannot be used. This is because the order the fragments are combined in is critical in producing the correct final color. Polygons should therefore be depth sorted, and rendered front to back, using the alpha blend modes: SourceAlphaSaturate for the source blend function and One for the destination blend function. In this way the alpha component of a fragment represents the percentage pixel coverage, and the blend function accumulates coverage until the value in the alpha buffer equals one, at which point no further contributions can made to a pixel.

For the antialiasing of general scenes, with no restrictions on rendering order, the accumulation buffer is the preferred choice. This is indirectly supported by GLINT via image uploading and downloading, with the accumulation buffer residing on the host.

When antialiasing, interpolated parameters which are sampled within a fragment (color, fog and texture), will sometimes be unrepresentative of a continuous sampling of a surface, and care should be taken when rendering smooth shaded antialiased primitives. This problem does not occur in aliased rendering, as the sample point is consistently at the center of a pixel.

Alpha Test Unit

The alpha test compares a fragment's alpha value with a reference value. Alpha testing is not available in color index (CI) mode. The alpha test conditionally rejects a fragment based on the comparison between a reference alpha value and one associated with the fragment.

Localbuffer Read/Write Unit

The localbuffer holds the Graphic ID, FrameCount, Stencil and Depth data associated with a fragment. The localbuffer read/write unit controls the operation of GID testing, depth testing and stencil testing.

Localbuffer Read

The LBReadMode register can be configured to make 0, 1 or 2 reads of the localbuffer. The following are the most common modes of access to the localbuffer:

- Normal rendering without depth, stencil or GID testing. This requires no localbuffer reads or writes.
- Normal rendering without depth or stencil testing and with GID testing. This requires a localbuffer read to get the GID from the localbuffer.
- Normal rendering with depth and/or stencil testing required which conditionally requires the localbuffer to be updated. This requires localbuffer reads and writes to be enabled.
- Copy operations. Operations which copy all or part of the localbuffer with or without GID testing. This requires reads and writes enabled.
- Image upload/download operations. Operations which download depth or stencil information to the local buffer or read depth, stencil fast clear or GID from the localbuffer.

Localbuffer Write

Writes to the localbuffer must be enabled to allow any update of the localbuffer to take place. The LBWriteMode register is a single bit flag which controls updating of the buffer.

Pixel Ownership (GID) Test Unit

Any fragment generated by the rasterizer may undergo a pixel ownership test. This test establishes the current fragment's write permission to the localbuffer and framebuffer.

Pixel Ownership Test

The ownership of a pixel is established by testing the GID of the current window against the GID of a fragment's destination in the GID buffer. If the test passes, then a write can take place, otherwise the write is discarded. The sense of the test can be set to one of: always pass, always fail, pass if equal, or pass if not equal. Pass if equal is the normal mode. In GLINT the GID planes, if present, are 4 bits deep allowing 16 possible Graphic ID's. The current GID is established by setting the Window register.

If the unit is disabled fragments pass through undisturbed.

Stencil Test Unit

The stencil test conditionally rejects fragments based on the outcome of a comparison between the value in the stencil buffer and a reference value. The stencil buffer is updated according to the current stencil update mode which depends on the result of the stencil test and the depth test.

Stencil Test

This test only occurs if all the preceding tests (bitmask, scissor, stipple, alpha, pixel ownership) have passed. The stencil test is controlled by the stencil function and the stencil operation. The stencil function controls the test between the reference stencil value and the value held in the stencil buffer. The stencil operation controls the updating of the stencil buffer, and is dependent on the result of the stencil and depth tests.

If the stencil test is enabled then the stencil buffer will be updated depending on the outcome of both the stencil and the depth tests (if the depth test is not enabled the depth result is set to pass).

In addition a comparison bit mask is supplied in the StencilData register. This is used to establish which bits of the source and reference value are used in the stencil function test. In addition it should normally be set to exclude the top four bits when the stencil width has been set to 4 bits in the StencilMode register.

The source stencil value can be from a number of places as controlled by a field in the StencilMode register:

LBWriteData Stencil	Use
Test logic	This is the normal mode.
Stencil register	This is used, for instance, in the OpenGL draw pixels function where the host supplies the stencil values in the Stencil register. This is used when a constant stencil values is needed, for example, when clearing the stencil buffer when fast clear planes are not available.
LBSourceData: (stencil value read from the localbuffer)	This is used, for instance, in the OpenGL copy pixels function when the stencil planes are to be copied to the destination. The source is offset from the destination by the value in LBSourceOffset register.
Source stencil value read from the localbuffer	This is used, for instance, in the OpenGL copy pixels function when the stencil planes in the destination are not to be updated. The stencil data will come either from the localbuffer date, or the FCStencil register, depending on whether fast clear operations are enabled.

Depth Test Unit

The depth (Z) test, if enabled, compares a fragment's depth against the corresponding depth in the depth buffer. The result of the depth test can effect the updating of the stencil buffer if stencil testing is enabled. This test is only performed if all the preceding tests (bitmask, scissor, stipple, alpha, pixel ownership, stencil) have passed. The source value can be obtained from a number of places as controlled by a field in the DepthMode register:

Source	Use
DDA (see below)	This is used for normal Depth buffered 3D rendering.
Depth register	This is used, for instance, in the OpenGL draw pixels function where the host supplies the depth values through the Depth register. Alternatively this is used when a constant depth value is needed, for example, when clearing the depth buffer (when fast clear planes are not available) or 2D rendering where the depth is held constant.
LBSourceData: Source depth value from the localbuffer	This is used, for instance, in the OpenGL copy pixels function when the depth planes are to be copied to the destination.
Source Depth	This is used, for instance, in the OpenGL copy pixels function when the depth planes in the destination are not updated. The depth data will come either from the localbuffer or the FCDepth register depending the state of the Fast Clear modes in operation.

When using the depth DDA for normal depth buffered rendering operations the depth values required are similar to those required for the color values in the color DDA unit:

ZStart=Start Z Value

dZdYDom=Increment along dominant edge.

dZdX=Increment along the scan line.

The dZdX value is not required for Z-buffered lines.

The depth unit must be enabled to update the depth buffer. If it is disabled then the depth buffer will only be updated if ForceL-BUpdate is set in the Window register.

Framebuffer Read/Write Unit

Before rendering can take place GLINT must be configured to perform the correct framebuffer read and write operations. Framebuffer read and write modes effect the operation of alpha blending, logic ops, write masks, image upload/download operations and the updating of pixels in the framebuffer.

Framebuffer Read

The FBReadMode register allows GLINT to be configured to make 0, 1 or 2 reads of the framebuffer. The following are the most common modes of access to the framebuffer: Note that avoiding unnecessary additional reads will enhance performance.

Rendering operations with no logical operations, software write-masking or alpha blending. In this case no read of the framebuffer is required and framebuffer writes should be enabled.

Rendering operations which use logical ops, software write masks or alpha blending. In these cases the destination pixel must be read from the framebuffer and framebuffer writes must be enabled.

Image copy operations. Here setup varies depending on whether hardware or software write masks are used. For software write masks, the framebuffer needs two reads, one for the source and one for the destination. When hardware write masks are used (or when the software write mask allows updating of all bits in a pixel) then only one read is required.

Image upload. This requires reading of the destination framebuffer reads to be enabled and framebuffer writes to be disabled.

Image download. In this case no framebuffer read is required (as long as software writemasking and logic ops are disabled) and the write must be enabled.

For both the read and the write operations, an offset is added to the calculated address. The source offset (FBSourceOffset) is used for copy operations. The pixel offset (FBPixelOffset) can be used to allow multi-buffer updates. The offsets should be set to zero for normal rendering.

The data read from the framebuffer may be tagged either FBDefault (data which may be written back into the framebuffer or used in some manner to modify the fragment color) or FBColor (data which will be uploaded to the host). The table below summarizes the framebuffer read/write control for common rendering operations:

Read-Source	ReadDestination	Writes	Read Data Type	Rendering Operation
Disabled	Disabled	Enabled	—	Rendering with no logical operations, software write masks or blending.
Disabled	Disabled	Enabled	—	Image download.
Disabled	Enabled	Disabled	FBColor	Image upload.
Enabled	Disabled	Enabled	FBDefault	Image copy with hardware write masks.
Disabled	Enabled	Enabled	FBDefault	Rendering using logical operations, software write masks or blending.
Enabled	Enabled	Enabled	FBDefault	Image copy with software writemasks.

Framebuffer Write

Framebuffer writes must be enabled to allow the framebuffer to be updated. A single 1 bit flag controls this operation.

The framebuffer write unit is also used to control the operation of fast block fills, if supported by the framebuffer. Fast fill rendering is enabled via the FastFillEnable bit in the Render command register, the framebuffer fast block size must be configured to the same value as the FastFillIncrement in the Render command register. The FBBlockColor register holds the data written to the framebuffer during a block fill operation and should be formatted to the 'raw' framebuffer format. When using the framebuffer in 8 bit packed mode the data should be replicated into each byte. When using the framebuffer in packed 16 bit mode the data should be replicated into the top 16 bits.

When uploading images the UpLoadData bit can be set to allow color formatting (which takes place in the Alpha Blend unit).

It should be noted that the block write capability provided by the chip of the presently preferred embodiment is itself believed to be novel. According to this new approach, a graphics system can do masked block writes of variable length (e.g. 8, 16, or 32 pixels, in the presently preferred embodiment). The rasterizer defines the limits of the block to be written, and hardware masking logic in the framebuffer interface permits the block to be filled in, with a specified primitive, only up to the limits of the object being rendered. Thus the rasterizer can step by the Block Fill increment. This permits the block-write capabilities of the VRAM chips to be used optimally, to minimize the length which must be written by separate writes per pixel.

Alpha Blend Unit

Alpha blending combines a fragment's color with those of the corresponding pixel in the framebuffer. Blending is supported in RGBA mode only.

Alpha Blending

The alpha blend unit combines the fragment's color value with that stored in the framebuffer, using the blend equation:

$$C_o = C_s S + C_d D$$

where: C_o is the output color; C_s is the source color (calculated internally); C_d is the destination color read from

the framebuffer; S is the source blending weight; and D is the destination blending weight. S and D are not limited to linear combinations; lookup functions can be used to implement other combining relations.

If the blend operations require any destination color components then the framebuffer read mode must be set appropriately.

Image Formatting

The alpha blend and color formatting units can be used to format image data into any of the supported GLINT framebuffer formats.

Consider the case where the framebuffer is in RGBA 4:4:4:4 mode, and an area of the screen is to be uploaded and stored in an 8 bit RGB 3:3:2 format. The sequence of operations is:

Set the rasterizer as appropriate

Enable framebuffer reads

Disable framebuffer writes and set the UpLoadData bit in the FBWriteMode register

Enable the alpha blend unit with a blend function which passes the destination value and ignores the source value (source blend Zero, destination blend One) and set the color mode to RGBA 4:4:4:4

Set the color formatting unit to format the color of incoming fragments to an 8 bit RGB 3:3:2 framebuffer format.

The upload now proceeds as normal. This technique can be used to upload data in any supported format.

The same technique can be used to download data which is in any supported framebuffer format, in this case the rasterizer is set to sync with FBColor, rather than Color. In this case framebuffer writes are enabled, and the UpLoadData bit cleared.

Color Formatting Unit

The color formatting unit converts from GLINT's internal color representation to a format suitable to be written into the framebuffer. This process may optionally include dithering of the color values for framebuffers with less than 8 bits width per color component. If the unit is disabled then the color is not modified in any way.

As noted above, the framebuffer may be configured to be RGBA or Color Index (CI).

Color Dithering

GLINT uses an ordered dither algorithm to implement color dithering. Several types of dithering can be selected.

If the color formatting unit is disabled, the color components RGBA are not modified and will be truncated when placed in the framebuffer. In CI mode the value is rounded to the nearest integer. In both cases the result is clamped to a maximum value to prevent overflow.

In some situations only screen coordinates are available, but window relative dithering is required. This can be implemented by adding an optional offset to the coordinates before indexing the dither tables. The offset is a two bit number which is supplied for each coordinate, X and Y. The XOffset, YOffset fields in the DitherMode register control this operation, if window relative coordinates are used they should be set to zero.

Logical Op Unit

The logical op unit performs two functions; logic operations between the fragment color (source color) and a value from the framebuffer (destination color); and, optionally, control of a special GLINT mode which allows high performance flat shaded rendering.

High Speed Flat Shaded Rendering

A special GLINT rendering mode is available which allows high speed rendering of unshaded images. To use the mode the following constraints must be satisfied:

Flat shaded aliased primitive

No dithering required

No logical ops

No stencil, depth or GID testing required

No alpha blending The following are available:

Bit masking in the rasterizer

Area and line stippling

User and Screen Scissor test

If all the conditions are met then high speed rendering can be achieved by setting the FBWriteData register to hold the framebuffer data (formatted appropriately for the framebuffer in use) and setting the UseConstantFBWriteData bit in the LogicalOpMode register. All unused units should be disabled.

This mode is most useful for 2D applications or for clearing the framebuffer when the memory does not support block writes. Note that FBWriteData register should be considered volatile when context switching.

Logical Operations

The logical operations supported by GLINT are:

Mode	Name	Operation	Mode	Name	Operation
0	Clear	0	8	Nor	~(S | D)
1	And	S & D	9	Equivalent	~(S ^ D)
2	And Reverse	S & ~D	10	Invert	~D
3	Copy	S	11	Or Reverse	S | ~D
4	And Inverted	~S & D	12	Copy Invert	~S
5	Noop	D	13	Or Invert	~S | D
6	Xor	S ^ D	14	Nand	~(S & D)
7	Or	S | D	15	Set	1

Where:

S=Source (fragment) Color, D=Destination (framebuffer) Color.

For correct operation of this unit in a mode which takes the destination color, GLINT must be configured to allow reads from the framebuffer using the FBReadMode register.

GLINT makes no distinction between RGBA and CI modes when performing logical operations. However, logical operations are generally only used in CI mode.

Framebuffer Write Masks

Two types of framebuffer write masking are supported by GLINT, software and hardware. Software write masking requires a read from the framebuffer to combine the fragment color with the framebuffer color, before checking the bits in the mask to see which planes are writeable. Hardware write masking is implemented using VRAM write masks and no framebuffer read is required.

Software Write Masks

Software write masking is controlled by the FBSoftwareWriteMask register. The data field has one bit per framebuffer bit which when set, allows the corresponding framebuffer bit to be updated. When reset it disables writing to that bit. Software write masking is applied to all fragments and is not controlled by an enable/disable bit. However it may effectively be disabled by setting the mask to all 1's. Note that the ReadDestination bit must be enabled in the FBReadMode register when using software write masks, in which some of the bits are zero.

Hardware Write Masks

Hardware write masks, if available, are controlled using the FBHardwareWriteMask register. If the framebuffer supports hardware write masks, and they are to be used, then software write masking should be disabled (by setting all the

bits in the FBSoftwareWriteMask register). This will result in fewer framebuffer reads when no logical operations or alpha blending is needed.

If the framebuffer is used in 8 bit packed mode, then an 8 bit hardware write mask must be replicated to all 4 bytes of the FBHardwareWriteMask register. If the framebuffer is in 16 bit packed mode then the 16 bit hardware write mask must be replicated to both halves of the FBHardwareWriteMask register.

Host Out Unit

Host Out Unit controls which registers are available at the output FIFO, gathering statistics about the rendering operations (picking and extent testing) and the synchronization of GLINT via the Sync register. These three functions are as follows:

Message filtering. This unit is the last unit in the core so any message not consumed by a preceding unit will end up here. These messages will fall in to three classifications: Rasterizer messages which are never consumed by the earlier units, messages associated with image uploads, and finally programmer mistakes where an invalid message was written to the input FIFO. Synchronization messages are a special category and are dealt with later. Any messages not filtered out are passed on the output FIFO.

Statistic Collection. Here the active step messages are used to record the extent of the rectangular region where rasterization has been occurring, or if rasterization has occurred inside a specific rectangular region. These facilities are useful for picking and debug activities.

Synchronization. It is often useful for the controlling software to find out when some rendering activity has finished, to allow the timely swapping or sharing of buffers, reading back of state, etc. To achieve this the software would send a Sync message and when this reached this unit any preceding messages or their actions are guaranteed to have finished. On receiving the Sync message it is entered into the FIFO and optionally generates an interrupt.

Sample Board-Level Embodiment

A sample board incorporating the GLINT chip may include simply:

the GLINT chip itself, which incorporates a PCI interface;

Video RAM (VRAM), to which the chip has read-write access through its frame buffer (FB) port;

DRAM, which provides a local buffer then made for such purposes as Z buffering; and

a RAMDAC, which provides analog color values in accordance with the color values read out from the VRAM.

Thus one of the advantages of the chip of the presently preferred embodiment is that a minimal board implementation is a trivial task.

FIG. 3A shows a sample graphics board which incorporates the chip of FIG. 2B.

FIG. 3B shows another sample graphics board implementation, which differs from the board of FIG. 3A in that more memory and an additional component is used to achieve higher performance.

FIG. 3C shows another graphics board, in which the chip of FIG. 2B shares access to a common frame store with GUI accelerator chip.

FIG. 3D shows another graphics board, in which the chip of FIG. 2B shares access to a common frame store with a video coprocessor (which may be used for video capture and playback functions (e.g. frame grabbing).

Alternative Board Embodiment with Additional Video Processor

In the presently preferred embodiment, the frame buffer interface of the GLINT chip contains additional simple interface logic, so that two chips can both access the same frame buffer memory. This permits the GLINT chip to be combined with an additional chip for management to the graphics produced by the graphical user interface. This provides a migration path for users and applications who need to take advantage of the existing software investment and device drivers for various other graphics chips.

FIG. 3C shows another graphics board, in which the chip of FIG. 2B shares access to a common frame store with a GUI accelerator chip (such as an S3 chip). This provides a path for software migration, and also provides a way to separate 3D rendering tasks from 2D rendering.

In this embodiment, a shared framebuffer is used to enable multiple devices to read or write data to the same physical framebuffer memory. Example applications using the GLINT 300SX:

- Using a video device as a coprocessor to GLINT, to grab live video into the framebuffer, for displaying video in a window or acquiring a video sequence;
- Using GLINT as a 3D coprocessor to a 2D GUI accelerator, preserving an existing investment in 2D driver software.

In a coprocessor system, the framebuffer is a shared resource, and so access to the resource needs to be arbitrated. There are also other aspects of sharing a framebuffer that need to be considered:

- Memory refreshing;
- Transfer of data from the memory cells into the shift registers of the VRAM;
- Control of writemasks and color registers.

GLINT uses the S3 Shared Frame Buffer Interface (SFBI) to share a framebuffer. This interface is able to handle all of the above aspects for two devices sharing a frame buffer, with the GLINT acting as an arbitration master or slave.

Timing Considerations in Shared Frame-Buffer Interface

The Control Signals used in the Shared Framebuffer interface, in the presently preferred embodiment, are as follows:

GLINT as Primary Controller

- FBReqN is internally re-synchronized to System Clock.
- FBSelOEN remains negated.
- FBGntN is asserted an unspecified amount of time after FBReqN is asserted.—Framebuffer Address, Data and Control lines are tri-stated by GLINT (the control lines should be held high by external pull-up resistors). The secondary controller is now free to drive the Framebuffer lines and access the memory.
- FBGntN remains asserted until GLINT requires a framebuffer access, or a refresh or transfer cycle.
- FBReqN must remain asserted while FBGntN is asserted.
- When FBGntN is removed, the secondary controller must relinquish the address, data and control bus in a graceful manner i.e. RAS, CAS, WE and OE must all be driven high before being tri-stated.
- The secondary controller must relinquish the bus and negate FBReqN within 500 ns of FBGntN being negated.
- Once FBReqN has been negated, it must remain inactive for at least 2 system clocks (40 ns at 50 MHz).

GLINT as a Secondary Controller

Framebuffer Refresh and VRAM transfer cycles by GLINT are turned off when GLINT is a secondary framebuffer controller.

GLINT asserts FBReqN whenever is requires a framebuffer access.

FBGntN is internally re-synchronized to system clock.

When FBGntN is asserted, GLINT drives FBselOEN to enable any external buffers used to drive the control signals, and then drives the framebuffer address, data and control lines to perform the memory access. FBReqN remains asserted while FBGntN is asserted.

When FBGntN is negated, GLINT finishes any outstanding memory cycles, drives the control lines inactive, negates FBselOEN and then tri-states the address, data and control lines, then releases FBReqN. GLINT guarantees to release FBReqN within 500 ns of FBGntN being negated.

GLINT will not reassert FBReqN within 4 system clock cycles (80 ns@ 50 MHz).

Considerations for Board-Level Implementations

The following are some points to be noted when implementing a shared framebuffer design with a GLINT 300SX:

Some 2D GUI Accelerators such as the S3 Vision964, and GLINT use configuration resistors on the framebuffer databus at reset. In this case care should be taken with the configuration setup where it effects read only registers inside either device. If conflicts exist that can not be resolved by the board initialization software, then the conflicts should be resolved by isolating the two devices from each other at reset so they can read the correct configuration information. This isolation need only be done for the framebuffer databus lines that cause problems;

GLINT should be configured as the secondary controller when used with an S3 GUI accelerator, as the S3 devices can only be primary controllers;

GLINT cannot be used on the daughter card interface as described in the S3 documentation, because this gives no access to the PCI bus. A suitable PCI bridge should be used in a design with a PCI 2D GUI accelerator and GLINT so they can both have access to the PCI bus;

The use of ribbon cable to carry the framebuffer signals between two PCI boards is not recommended, because of noise problems and the extra buffering required would impact performance;

The GLINT 300SX does not provide a way of sharing its localbuffer.

The 400TX also allows grabbing of live video into the localbuffer and real-time texture mapping of that video into the framebuffer for video manipulation effects.

Alternative Board Embodiments with Multiple Rendering Accelerator Chips

This technical note describes some system design issues on how multiple GLINT devices can be used in parallel to achieve higher performance. The main driving force for higher performance is the simulation market which, at the low end, demands somewhere between 25–30M texture mapped pixels per second.

There are some key points before we look at different parallel organizations:

To gain any benefit from running multiple GLINTs in parallel, the overall system must be rendering bound. If the system is host bound or geometry bound, then adding in more GLINTs will not improve the systems performance.

The memory systems (i.e. local buffer and framebuffer) are duplicated for each GLINT. Recall that the texture maps are stored in the local buffer. A single GLINT places very high demands on the memory systems, and it would be very difficult to share them between multiple GLINTs. In the presently preferred embodiment there are no provisions for sharing the local buffer, so if this is necessary it would have to be done behind GLINT's back and transparently. The framebuffer can be shared (since GLINT has a SFB interface), but this is likely to be a bottle neck if shared between GLINTs.

Broadcast. In some parallel systems each GLINT will get the same (or mostly the same) primitive data and just render those pixels assigned to it. It is very desirable that this data is written by the host only once, or fetched from the host address space once if DMA is being used. This presents two issues: Firstly the PCI bus does not have any concept of broadcasting to multiple devices, and secondly GLINT does not have a dedicated FIFO status signal pin an external controller can use. Neither of these issues are insurmountable, but will require hardware to solve. However, if the application only uses a 'few' large texture mapped primitives so repeatedly sending or fetching the parameters for each GLINT will not be a problem.

To avoid problems with Antialiasing, Bitmasks for characters, or Line stipple, the area stipple table can be used to reserve scanlines to a processor.

Parallel Configurations

This section looks at some of the common ways of applying parallelism to the rendering operation. The list is not exhaustive and an interested reader is directed to the book by Whitman cited above. No one paradigm is best and the choice is very application or market dependent.

Frame Interleaving

Frame Interleaving is where a GLINT works on frame n, the next GLINT works on frame n+1, etc. Each GLINT does everything for its own frame and the video is sourced from each GLINT's framebuffer in turn. This paradigm is perhaps the simplest one with very little hardware overhead and none of the above complications regarding antialiasing, block copies, bitmasks and line stipples.

This scheme only works when the image is double buffered (normal for simulation systems) and where the increase in transport delay is acceptable. Transport delay is the time it takes for a user to see a visual change after new input stimulus to the system has occurred. With 4 GLINTs this will be 4 frame times attributable to the rendering system, plus whatever else the whole system adds.

The cost of this method is also one of the highest, as ALL the memory has to be duplicated. By contrast, the schemes where the screen is divided up can save depth and color buffer memory (but not texture memory).

Sequential frames will usually have very similar amounts of rendering, unless there is a discontinuity in the viewing position and/or orientation, so load balancing is generally good.

Frame Merging or Primitive Parallelism

Frame merging is a similar technique to frame interleaving where each GLINT has a full local buffer and framebuffer. In this case the primitives are distributed amongst the GLINTs and the resultant partial images composited using the depth information to control which fragment from the multiple buffers is displayed in each pixel position.

GLINT has not been designed to share the local buffer (where the depth information is held) so the compositing is not readily supported. Also the composition frequently needs to be done at video rate so requires some fast hardware.

Alpha blending and Antialiasing presents some problems but the bitmask, block copies and line stipple are easily accommodated. Good load balancing depends on even distribution of primitives. Not all primitives will take the same amount of time to process so a round robin distribution scheme, or a heuristic one with takes into account the expected processing time for each primitive will be needed.

Screen Subdivision—Blocks

Here the screen is divided up into large contiguous regions and a GLINT looks after each region. Primitives which overlap between regions are sent to both regions and scissor clipping used. Primitives contained wholly in one region are ideally just sent to the one GLINT.

The number of regions and the horizontal and/or vertical division of the screen can be chosen as appropriate, but horizontal bands are usually easier for the video hardware to cope with. Each GLINT only needs enough local buffer and frame buffer to cover the pixels in its own region, but texture maps are duplicated in full. Block copies are a problem when the block, or part block is moved between regions. Bit masking and line stipples can be solved with some careful clipping.

Load balancing is very poor in this paradigm, since most of the scene complexity can be concentrated into one region. Dynamically changing the size of the regions based on expected scene complexity (maybe measured from the previous frame) can alleviate the poor load balancing to some extent.

Screen Subdivision—Interleaved Scanlines

The interleave factor is every other n^{th} scanline where n is the number of GLINTs. Vertical interleaves are possible, but not supported by the GLINT rasterizer. Nearly all primitives will overlap multiple scanlines so are ideally broadcast to all GLINTs. Each GLINT will have different start values for the rasterization and interpolation parameters.

Each GLINT only needs enough local buffer and frame buffer to cover the pixels in its own region, but texture maps are duplicated in full.

Some block copies are a problem when the block is moved between non nth scanlines, but horizontal moves are available with any alignment. Bit masking can be solved with some careful clipping, but line stipples have no easy solution. Antialiasing is not normally a problem but with GLINT 300SX there is no provision for sub scanline steps as well as nth scanline steps. Load balancing is excellent in this paradigm which is the main reason it features prominently in the literature.

Thus the simplest and lowest risk method of using multiple GLINTs is Frame Interleaving, but if this is not an option, e.g. because of the transport delay or the amount of memory needed, then the next best choice is the Interleaved Scanline.

Linkage

FIG. **2B** shows how the units are connected together. Some general points are:

The order of the units can be configured in two ways. The most general order (Router, Colour DDA, Texture Units, Fog Unit, Alpha Test, LB Rd, GID/Z/Stencil, LB Wr, Multiplexer) and will work in all modes of OpenGL. However, when the alpha test is disabled it is much better to do the Graphics ID, depth and stencil tests before the texture operations rather than after. This is because the texture operations have a high processing cost and this should not be spent on fragments which are later rejected because of window, depth or stencil tests.

Router Unit Description

The Router Unit allows the order of some of the units to be changed so that texturing can be done before or after the depth test. Any texture operations will cause a loss in performance over the same non-textured rendering, so it is a good idea only to texture those pixels which pass all the depth, stencil and GID tests. OpenGL defines the order in which operations are to be performed on fragments as texture, alpha test, stencil and then depth. It is very likely that in a typical scene many textured fragments will get rejected by the depth test, say, which isn't the most effective use of the texturing capacity. If the alpha test is disabled (or cannot reject fragments) then OpenGL compatible semantics are still maintained if the order is rearranged to be stencil, depth, texture and then alpha test.

The message stream can be re-configured into either of the two orders using the RouterMode message. The reset order is texture, then depth so a to be compatible with OpenGL. Changing the pipeline order is self synchronising so the user doesn't need to wait for the message stream to empty first.

Implementation

This unit is divided into two sub-units: a switcher and a multiplexer. FIG. **5A** shows how these are connected together. The basic operation is as follows:

When the Switcher sub-unit receives a RouterMode message it makes a note of the new order, forwards the RouterMode message on and blocks all further messages until it receives a resume signal from the Multiplexer sub unit. When the resume signal is asserted the Switcher re-configures the message paths according to the new order and un-blocks the message stream so it starts to flow again.

When the Multiplexer sub-unit receives the RouterMode message it re-configures the message paths according to the new order and asserts the resume signal to the Switcher. The RouterMode message is consumed. The unit order is controlled using the RouterMode message. It uses the 0-bit of the passed message to indicate if the processing order is:

Bit 0=0	TextureDepth
Bit 0=1	DepthTexture

When the order is TextureDepth (the default after reset) the message routing is done according to FIG. **5B**. When the order is DepthTexture the message routing is done according to FIG. **5C**.

Disclosed Embodiments

Among the disclosed classes of preferred embodiments, there is provided: A method for processing graphics data through a data path comprising the steps of: (a) receiving a routing command from a data bus input; (b) stalling further input from said data bus input until previous data has exited said data path; (c) resuming said input from said data bus input; (d) if said routing command has a first value, then performing a first set of graphics processes on said data, and then performing a second set of graphics processes on said data; (e) if said routing command has a second value, thenperforming said second set of graphics processes on said data, and thenperforming said first set of graphics processes on said data, wherein some portion of said data may be eliminated by said first or second sets of graphics process according to the results of said processes; wherein steps (d) and (e) are repeated until a new routing command is received; wherein said first set of graphics processes requires a longer processing time than said second set of graphics processes.

Among the disclosed classes of preferred embodiments, there is also provided: A method for processing graphics data through a data path comprising the steps of: (a) receiving a routing command from a data bus input; (b) stalling further input from said data bus input until previous data has exited said data path; (c) resuming said input from said data bus input; (d) if said routing command has a first value, thenperforming a set of texturing processes on said data, and thenperforming a set of pixel elimination processes on said data; (e) if said routing command has a second value, thenperforming said set of pixel elimination processes on said data, and thenperforming said set of texturing processes on said data, wherein some portion of said data may be eliminated by said set of pixel elimination processes according to the results of said processes; wherein steps (d) and (e) are repeated until a new routing command is received; wherein said first set of graphics processes requires a longer processing time than said second set of graphics processes.

Among the disclosed classes of preferred embodiments, there is also provided: A method for rendering graphics data comprising the steps of: (a) receiving a routing command from a data bus input; (b) stalling further input from said data bus input until previous data has exited said data path; (c) resuming said input from said data bus input; (d) if said routing command has a first value, thenperforming a set of texturing processes on said data, and thenperforming a set of pixel elimination processes on said data; (e) if said routing command has a second value, thenperforming said set of pixel elimination processes on said data, and thenperforming said set of texturing processes on said data, wherein some portion of said data may be eliminated by said set of pixel elimination processes according to the results of said processes; (f) rendering said data and writing the results to a memory; (g) displaying the contents of said memory; wherein steps (d) and (e) are repeated until a new routing command is received;wherein said set of texturing processes requires a longer processing time than said set of pixel elimination processes.

Among the disclosed classes of preferred embodiments, there is also provided: A method for processing graphics data through a data path comprising the steps of: (a) receiving a routing command from a data bus input; (b) stalling further input from said data bus input until previous data has exited said data path; (c) resuming said input from said data bus input; (d) if said routing command has a first value, thenreading said graphics data from said data bus input; performing a color DDA process on said data;performing a texturing process on said data;performing an alpha test on said data; if the data has passed the previous test, then performing a graphics ID test on said data; if the data has passed the previous tests, then performing a stencil test on said data;if the data has passed the previous tests, then performing a depth test on said data; and if the data has passed the previous tests, then writing said data to a local bus; (e) if said routing command has a second value, thenreading said graphics data from said data bus input; performing a graphics ID test on said data;if the data has passed the previous test, then performing a stencil test on said data; if the data has passed the previous tests, then performing a depth test on said data; if the data has passed the previous tests, then performing a color DDA process on said data; if the data has passed the previous tests, then performing a texturing process on said data; if the data has passed the previous tests, then performing an alpha test on said data; if the data has passed the previous tests, then writing said data to a local bus; wherein steps (d) and (e) are repeated until a new routing command is received.

Among the disclosed classes of preferred embodiments, there is also provided: A pipelined graphics processing device, comprising:a switching device connected to a data bus input and configured to route graphics data received on said data bus according to instruction data received on said data bus; a multiplexing device connected to said switching device and to a data bus output; a first processing block connected and configured to receive said graphics data from said switching device and pass processed graphics data to said multiplexing device; anda second processing block connected and configured to receive said graphics data from said switching device and pass processed graphics data to said multiplexing device; wherein said switching device routes said graphics data according to a first data path, wherein said graphics data is processed by said first processing block and then by said second processing block, or a second data path, wherein said graphics data is processed by said second processing block before said first processing block, according to said instruction data.

Among the disclosed classes of preferred embodiments, there is also provided: A pipelined graphics processing device, comprising: a routing device connected to a data bus input and data bus output and configured to route graphics data received on said data bus according to instruction data received on said data bus; a first processing block connected and configured to receive said graphics data from said routing device and pass processed graphics data back to said routing device; anda second processing block connected and configured to receive said graphics data from said routing device and pass processed graphics data back to said routing device; wherein said routing device routes data according to a first data path, wherein said graphics data is processed by said first processing block and then by said second processing block, or a second data path, wherein said graphics data is processed by said second processing block before said first processing block, according to said instruction data.

Among the disclosed classes of preferred embodiments, there is also provided: A graphics processing subsystem, comprising: at least four functionally distinct processing units, each including hardware elements which are customized to perform a rendering operation which is not performed by at least some others of said processing units; at least some ones of said processing units being connected to operate asynchronously to one another; a frame buffer, connected to be accessed by at least one of said processing units;said processing units being mutually interconnected in a pipeline relationship, with at least some successive ones of said processing units being interconnected through a FIFO buffer; and wherein at least one said processing unit is connected to look downstream, in said pipeline relationship, past the immediately succeeding one of said processors; and wherein at least two of said processing units may be dynamically reordered in said pipeline relationship; whereby the duty cycle of said processors is increased while permitting use of a reduced depth for said FIFO.

Modifications and Variations

As will be recognized by those skilled in the art, the innovative concepts described in the present application can be modified and varied over a tremendous range of applications, and accordingly the scope of patented subject matter is not limited by any of the specific exemplary teachings given.

The foregoing text has indicated a large number of alternative implementations, particularly at the higher levels, but these are merely a few examples of the huge range of possible variations.

For example, the preferred chip context can be combined with other functions, or distributed among other chips, as will be apparent to those of ordinary skill in the art.

For another example, the described graphics systems and subsystems can be used, in various adaptations, not only in high-end PC's, but also in workstations, arcade games, and high-end simulators.

For another example, the described graphics systems and subsystems are not necessarily limited to color displays, but can be used with monochrome systems.

For another example, the described graphics systems and subsystems are not necessarily limited to displays, but also can be used in printer drivers.

What is claimed is:

1. A method for processing graphics data through a data path comprising the steps of:

(a) receiving a routing command from a data bus input;

(b) stalling further input from said data bus input until previous data has exited said data path;

(c) resuming said input from said data bus input;

(d) if said routing command has a first value, then
performing a first set of graphics processes on said data, and then
performing a second set of graphics processes on said data;

(e) if said routing command has a second value, then
performing said second set of graphics processes on said data, and then
performing said first set of graphics processes on said data, wherein some portion of said data is selectively eliminated by said first or second sets of graphics process according to the results of said processes;

wherein steps (d) and (e) are repeated until a new routing command is received;

wherein said first set of graphics processes requires a longer processing time than said second set of graphics processes.

2. The method of claim **1**, wherein said first set of graphics processes comprises the steps of:

reading said graphics data from said data bus input;

performing a color DDA process on said data;

performing a texturing process on said data; and

performing an alpha test on said data.

3. The method of claim **1**, wherein said second set of graphics processes comprises the step of if the data has passed all previous tests, then performing a graphics ID test on said data.

4. The method of claim **1**, wherein said second set of graphics processes comprises the step of if the data has passed the previous tests, then performing a stencil test on said data.

5. The method of claim **1**, wherein said second set of graphics processes comprises the steps of if the data has passed the previous tests, then performing a depth test on said data.

6. The method of claim **1**, wherein step (d) comprises steps according to the OpenGL standard.

7. The method of claim **1**, wherein step (b) is performed by a switcher connected at said data bus input.

8. The method of claim **1**, wherein a multiplexer at an output of said data path indicates when said data path is clear and step (c) can begin.

9. A method for processing graphics data through a data path comprising the steps of:

(a) receiving a routing command from a data bus input;

(b) stalling further input from said data bus input until previous data has exited said data path;

(c) resuming said input from said data bus input;

(d) if said routing command has a first value, then
performing a set of texturing processes on said data, and then
performing a set of pixel elimination processes on said data;

(e) if said routing command has a second value, then
performing said set of pixel elimination processes on said data, and then
performing said set of texturing processes on said data, wherein some portion of said data is selectively eliminated by said set of pixel elimination processes according to the results of said processes;

wherein steps (d) and (e) are repeated until a new routing command is received;

wherein said first set of graphics processes requires a longer processing time than said second set of graphics processes.

10. A method for rendering graphics data comprising the steps of:

(a) receiving a routing command from a data bus input;

(b) stalling further input from said data bus input until previous data has exited said data path;

(c) resuming said input from said data bus input;

(d) if said routing command has a first value, then
performing a set of texturing processes on said data, and then
performing a set of pixel elimination processes on said data;

(e) if said routing command has a second value, then
performing said set of pixel elimination processes on said data, and then
performing said set of texturing processes on said data, wherein some portion of said data is selectively eliminated by said set of pixel elimination processes according to the results of said processes;

(f) rendering said data and writing the results to a memory;

(g) displaying the contents of said memory;

wherein steps (d) and (e) are repeated until a new routing command is received;

wherein said set of texturing processes requires a longer processing time than said set of pixel elimination processes.

11. A method for processing graphics data through a data path comprising the steps of:

(a) receiving a routing command from a data bus input;

(b) stalling further input from said data bus input until previous data has exited said data path;

(c) resuming said input from said data bus input;

(d) if said routing command has a first value, then
reading said graphics data from said data bus input;
performing a color DDA process on said data;
performing a texturing process on said data;
performing an alpha test on said data;
if the data has passed the previous test, then performing a graphics ID test on said data;
if the data has passed the previous tests, then performing a stencil test on said data;

if the data has passed the previous tests, then performing a depth test on said data; and

if the data has passed the previous tests, then writing said data to a local bus;

(e) if said routing command has a second value, then reading said graphics data from said data bus input;

performing a graphics ID test on said data;

if the data has passed the previous test, then performing a stencil test on said data;

if the data has passed the previous tests, then performing a depth test on said data;

if the data has passed the previous tests, then performing a color DDA process on said data;

if the data has passed the previous tests, then performing a texturing process on said data;

if the data has passed the previous tests, then performing an alpha test on said data;

if the data has passed the previous tests, then writing said data to a local bus;

wherein steps (d) and (e) are repeated until a new routing command is received.

12. The method of claim **11**, wherein step (d) comprises steps according to the OpenGL standard.

13. The method of claim **11**, wherein step (b) is performed by a switcher connected at said data bus input.

14. The method of claim **11**, wherein a multiplexer at said local bus indicates when said data path is clear and step (c) can begin.

15. A pipelined graphics processing device, comprising:

a switching device connected to a data bus input and configured to route graphics data received on said data bus according to instruction data received on said data bus;

a multiplexing device connected to said switching device and to a data bus output;

a first processing block connected and configured to receive said graphics data from said switching device and pass processed graphics data to said multiplexing device; and

a second processing block connected and configured to receive said graphics data from said switching device and pass processed graphics data to said multiplexing device;

wherein said switching device routes said graphics data according to a first data path, wherein said graphics data is processed by said first processing block and then by said second processing block, or a second data path, wherein said graphics data is processed by said second processing block before said first processing block, according to said instruction data.

16. The device of claim **15**, wherein said first data path processes said graphics data according to the OpenGL standard.

17. The device of claim **15**, wherein said switching device halts all input data until the current data path is clear before switching data paths.

18. The device of claim **15**, wherein said multiplexing device is configured to determine when the current data path is clear and to allow said switching device to switch data paths.

19. A pipelined graphics processing device, comprising:

a routing device connected to a data bus input and data bus output and configured to route graphics data received on said data bus according to instruction data received on said data bus;

a first processing block connected and configured to receive said graphics data from said routing device and pass processed graphics data back to said routing device; and

a second processing block connected and configured to receive said graphics data from said routing device and pass processed graphics data back to said routing device;

wherein said routing device routes data according to a first data path, wherein said graphics data is processed by said first processing block and then by said second processing block, or a second data path, wherein said graphics data is processed by said second processing block before said first processing block, according to said instruction data.

20. A graphics processing subsystem, comprising:

at least four functionally distinct processing units, each including hardware elements which are customized to perform a rendering operation which is not performed by at least some others of said processing units; at least some ones of said processing units being connected to operate asynchronously to one another;

a frame buffer, connected to be accessed by at least one of said processing units;

said processing units being mutually interconnected in a pipeline relationship, with at least some successive ones of said processing units being interconnected through a FIFO buffer;

and wherein at least one said processing unit is connected to look downstream, in said pipeline relationship, past the immediately succeeding one of said processors;

and wherein at least two of said processing units are selectively dynamically reordered in said pipeline relationship;

whereby the duty cycle of said processors is increased while permitting use of a reduced depth for said FIFO.

21. The graphics processing subsystem of claim **20**, wherein said processing units include a texturing unit.

22. The graphics processing subsystem of claim **20**, wherein said processing units include a scissoring unit.

23. The graphics processing subsystem of claim **20**, wherein said processing units include a memory access unit which reads and writes a local buffer memory.

24. The graphics processing subsystem of claim **20**, wherein at least some ones of said processing units include internally paralleled data paths.

25. The graphics processing subsystem of claim **20**, wherein all of said processing units are integrated into a single integrated circuit.

26. The graphics processing subsystem of claim **20**, wherein all of said processing units, but not said frame buffer, are integrated into a single integrated circuit.

* * * * *

US005987256A

United States Patent [19]

Wu et al.

[11] Patent Number: 5,987,256

[45] Date of Patent: Nov. 16, 1999

[54] **SYSTEM AND PROCESS FOR OBJECT RENDERING ON THIN CLIENT PLATFORMS**

[75] Inventors: **Bo Wu**; **Ling Lu**, both of San Jose, Calif.

[73] Assignee: **Enreach Technology, Inc.**, San Jose, Calif.

[21] Appl. No.: **08/922,898**

[22] Filed: **Sep. 3, 1997**

[51] **Int. Cl.**[6] **G06F 9/45**

[52] **U.S. Cl.** **395/707**

[58] **Field of Search** 395/707

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,504,917	4/1996	Austin et al.	345/522
5,513,127	4/1996	Gard et al.	395/200.53
5,551,015	8/1996	Goettelmann et al.	395/707
5,586,020	12/1996	Isozaki	395/707
5,826,089	10/1998	Ireton	395/707

OTHER PUBLICATIONS

Blickstein et al. The GEM optimizing compiler system; Digital Technical Journal vol. 4 No. 4 Special Issue 1992 pp. 121–136.

Primary Examiner—Robert W. Downs

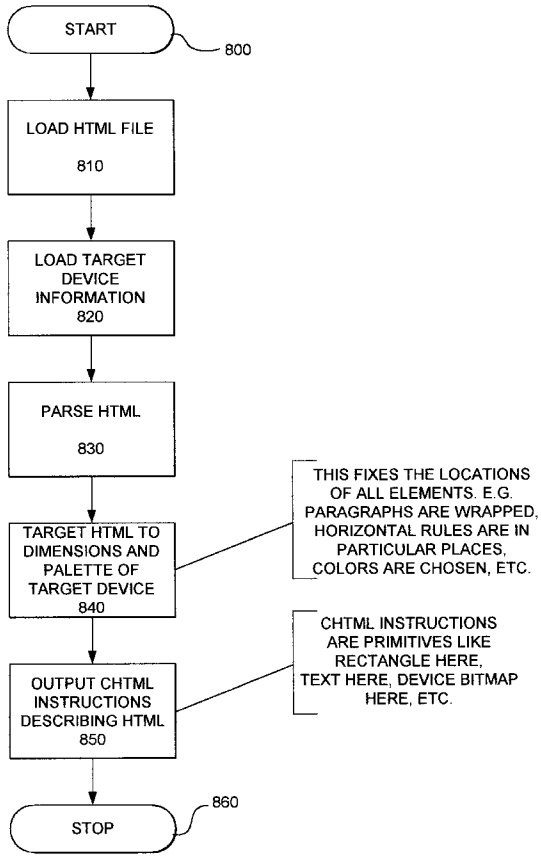
Assistant Examiner—Wei Zhen

Attorney, Agent, or Firm—Wilson Sonsini Goodrich & Rosati

[57] **ABSTRACT**

A system for processing an object specified by an object specifying language such as HTML, JAVA or other languages relying on relative positioning, that require a rendering program utilizing a minimum set of resources, translates the code for use in a target device that has limited processing resources unsuited for storage and execution of the HTML rendering program, JAVA virtual machine, or other rendering engine for the standard. Data concerning such an object is generated by a process that includes first receiving a data set specifying the object according to the object specifying language, translating the first data set into a second data set in an intermediate object language adapted for a second rendering program suitable for rendering by the target device that utilizes actual target display coordinates. The second data set is stored in a machine readable storage device, for later retrieval and execution by the thin client platform.

14 Claims, 11 Drawing Sheets



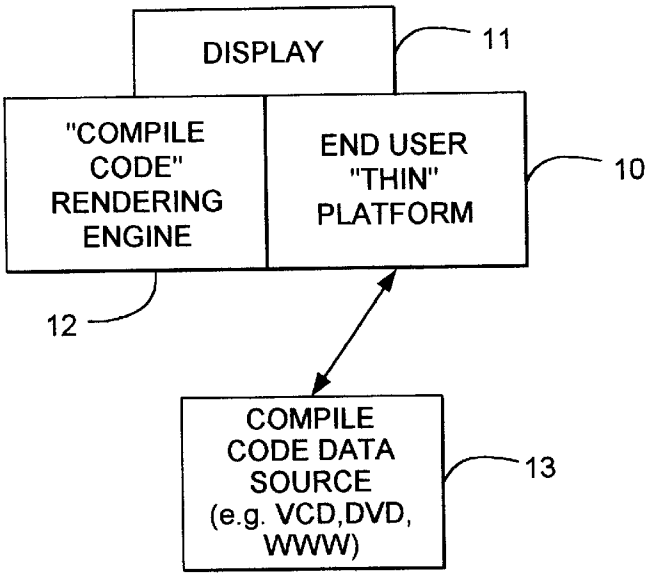

DISPLAY
11
"COMPILE CODE" RENDERING ENGINE
END USER "THIN" PLATFORM
10
12
COMPILE CODE DATA SOURCE (e.g. VCD,DVD, WWW)
13

FIG. 1

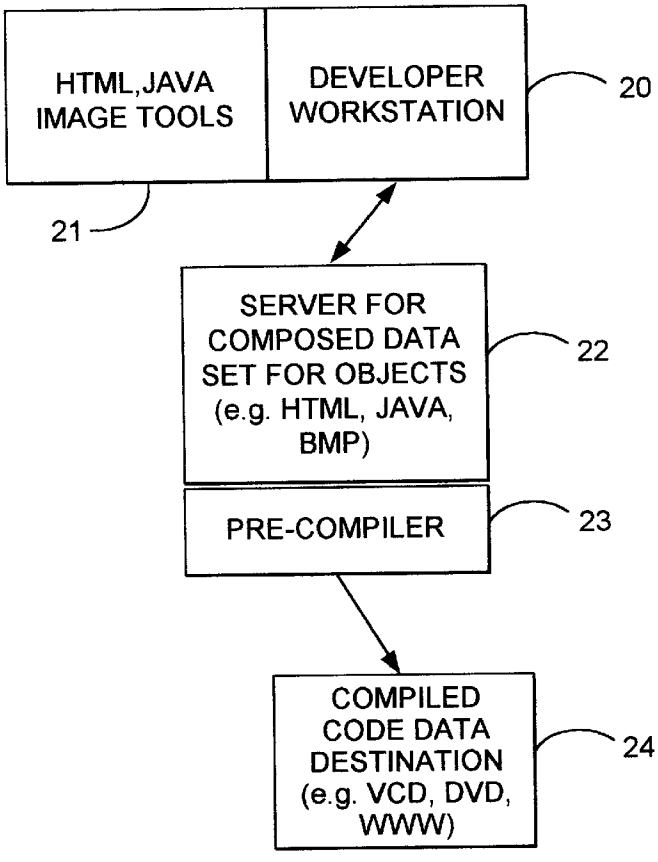

HTML,JAVA IMAGE TOOLS
DEVELOPER WORKSTATION
20
21
SERVER FOR COMPOSED DATA SET FOR OBJECTS (e.g. HTML, JAVA, BMP)
22
PRE-COMPILER
23
COMPILED CODE DATA DESTINATION (e.g. VCD, DVD, WWW)
24

FIG.2

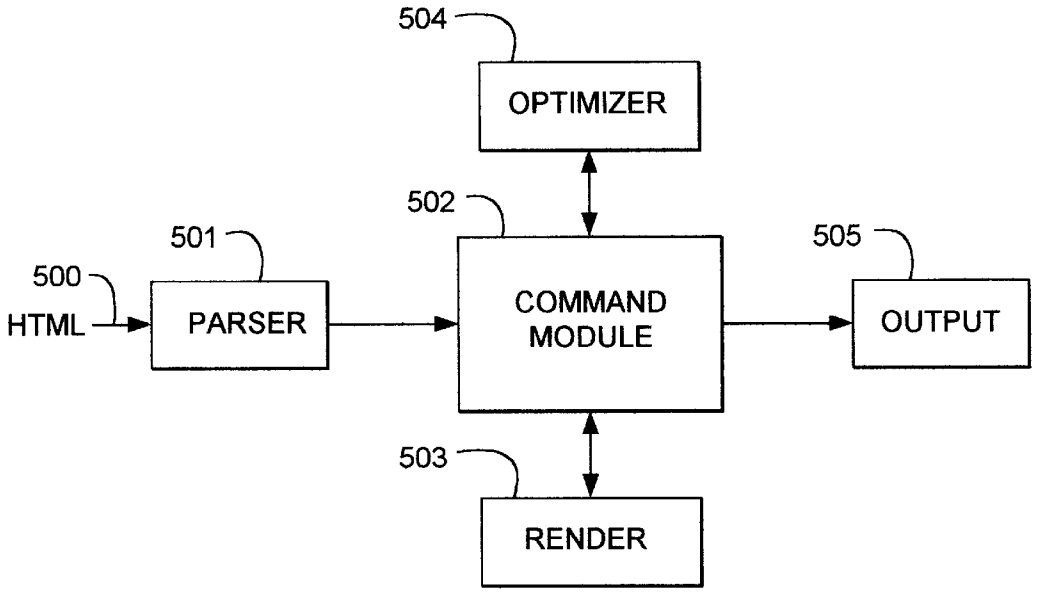

504
OPTIMIZER
502
501
505
500
HTML
PARSER
COMMAND MODULE
OUTPUT
503
RENDER

FIG. 3

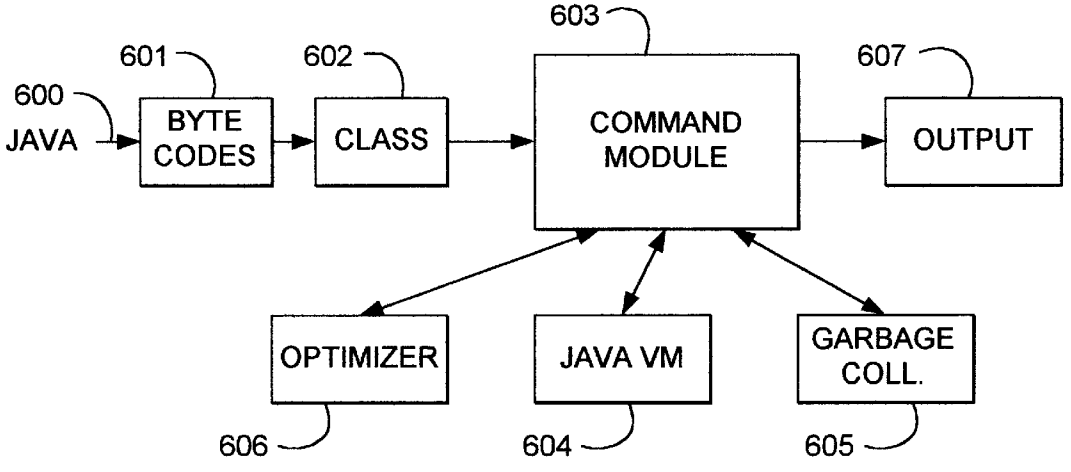

603
601
602
607
600
JAVA
BYTE CODES
CLASS
COMMAND MODULE
OUTPUT
OPTIMIZER
JAVA VM
GARBAGE COLL.
606
604
605

FIG.4

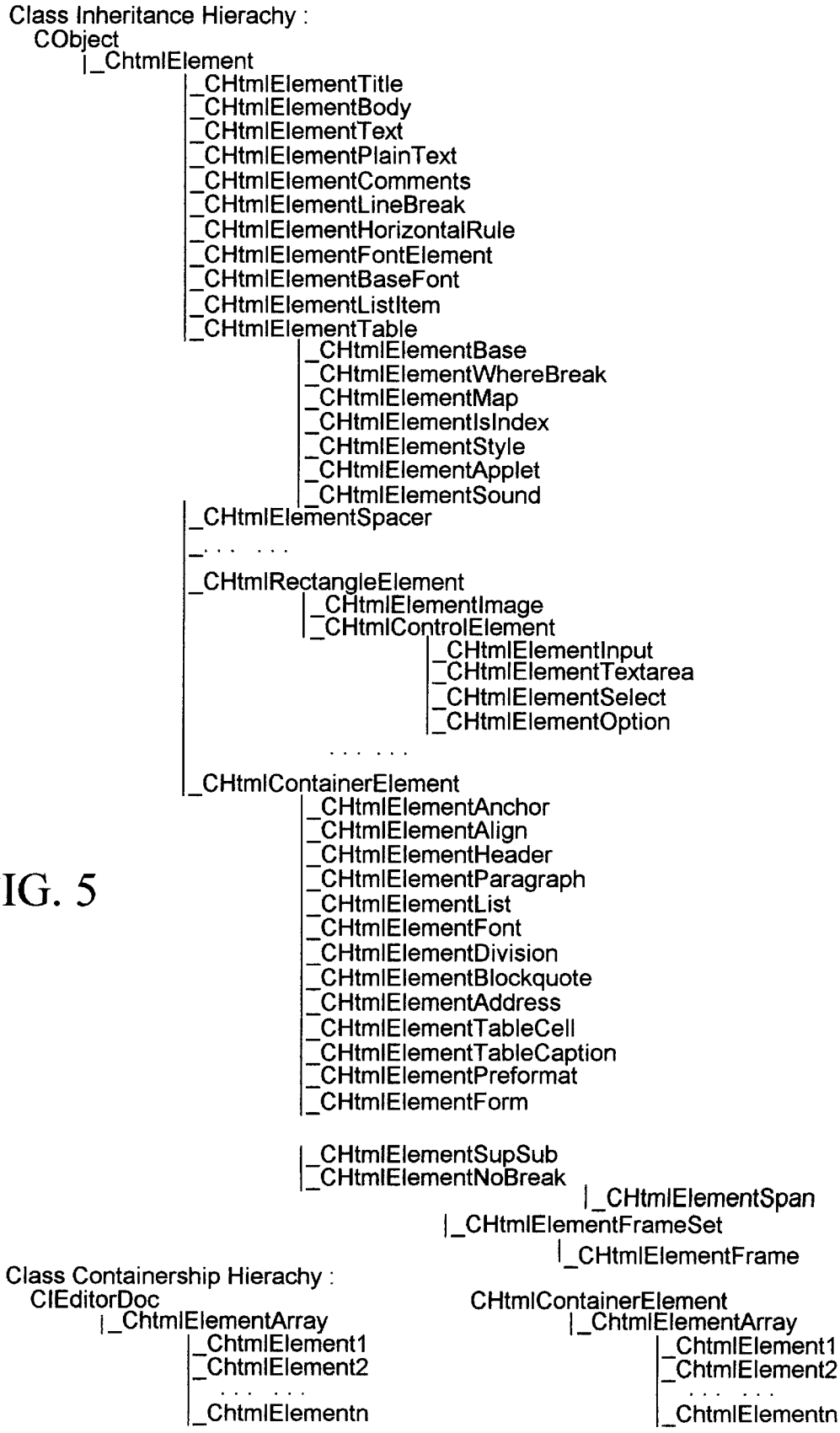

Class Inheritance Hierachy :
CObject
|_ChtmlElement
|_CHtmlElementTitle
|_CHtmlElementBody
|_CHtmlElementText
|_CHtmlElementPlainText
|_CHtmlElementComments
|_CHtmlElementLineBreak
|_CHtmlElementHorizontalRule
|_CHtmlElementFontElement
|_CHtmlElementBaseFont
|_CHtmlElementListItem
|_CHtmlElementTable
|_CHtmlElementBase
|_CHtmlElementWhereBreak
|_CHtmlElementMap
|_CHtmlElementIsIndex
|_CHtmlElementStyle
|_CHtmlElementApplet
|_CHtmlElementSound
|_CHtmlElementSpacer
|_. . . . . .
|_CHtmlRectangleElement
|_CHtmlElementImage
|_CHtmlControlElement
|_CHtmlElementInput
|_CHtmlElementTextarea
|_CHtmlElementSelect
|_CHtmlElementOption
. . . . . .
|_CHtmlContainerElement
|_CHtmlElementAnchor
|_CHtmlElementAlign
|_CHtmlElementHeader
|_CHtmlElementParagraph
|_CHtmlElementList
|_CHtmlElementFont
|_CHtmlElementDivision
|_CHtmlElementBlockquote
|_CHtmlElementAddress
|_CHtmlElementTableCell
|_CHtmlElementTableCaption
|_CHtmlElementPreformat
|_CHtmlElementForm
|_CHtmlElementSupSub
|_CHtmlElementNoBreak
|_CHtmlElementSpan
|_CHtmlElementFrameSet
|_CHtmlElementFrame
Class Containership Hierachy :
CIEditorDoc
|_ChtmlElementArray
|_ChtmlElement1
|_ChtmlElement2
. . . . . .
|_ChtmlElementn
CHtmlContainerElement
|_ChtmlElementArray
|_ChtmlElement1
|_ChtmlElement2
. . . . . .
|_ChtmlElementn

FIG. 5

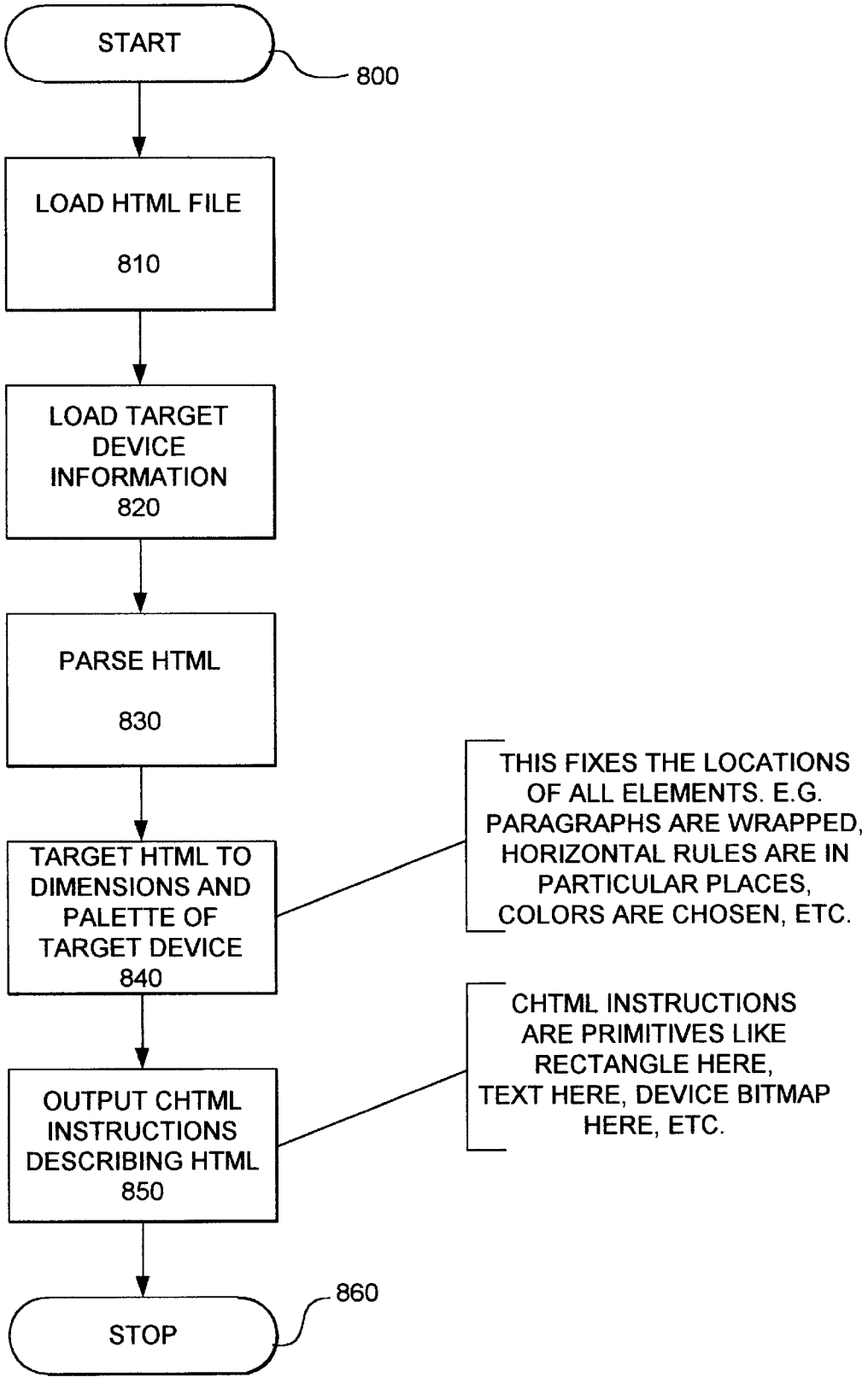

START
800
LOAD HTML FILE
810
LOAD TARGET DEVICE INFORMATION
820
PARSE HTML
830
TARGET HTML TO DIMENSIONS AND PALETTE OF TARGET DEVICE
840
THIS FIXES THE LOCATIONS OF ALL ELEMENTS. E.G. PARAGRAPHS ARE WRAPPED, HORIZONTAL RULES ARE IN PARTICULAR PLACES, COLORS ARE CHOSEN, ETC.
OUTPUT CHTML INSTRUCTIONS DESCRIBING HTML
850
CHTML INSTRUCTIONS ARE PRIMITIVES LIKE RECTANGLE HERE, TEXT HERE, DEVICE BITMAP HERE, ETC.
STOP
860

FIG. 6

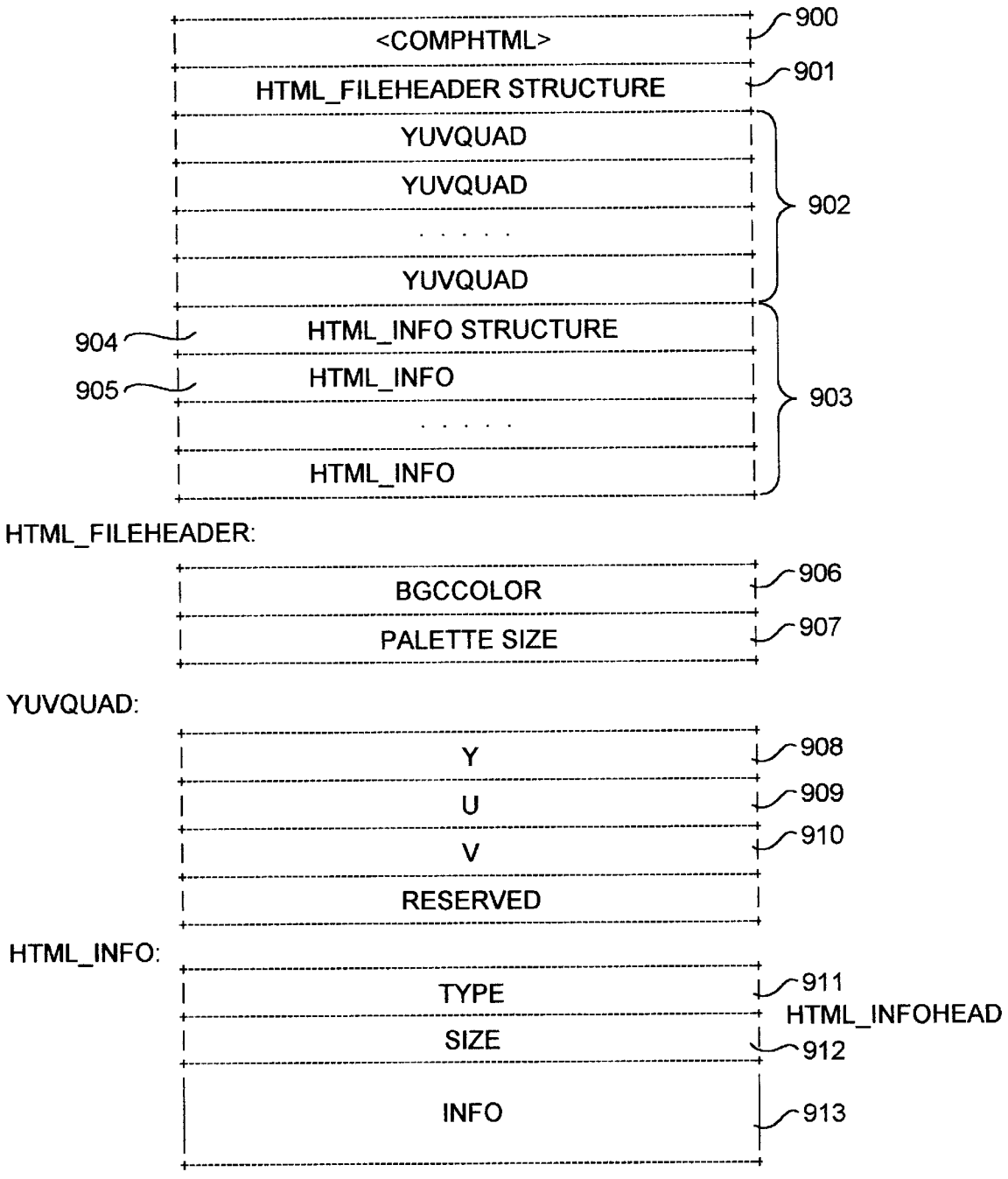

<COMPHTML>
900
HTML_FILEHEADER STRUCTURE
901
YUVQUAD
YUVQUAD
. . . . .
YUVQUAD
902
HTML_INFO STRUCTURE
904
HTML_INFO
905
. . . . .
HTML_INFO
903
HTML_FILEHEADER:
BGCCOLOR
906
PALETTE SIZE
907
YUVQUAD:
Y
908
U
909
V
910
RESERVED
HTML_INFO:
TYPE
911
HTML_INFOHEAD
SIZE
912
INFO
913

FIG. 7

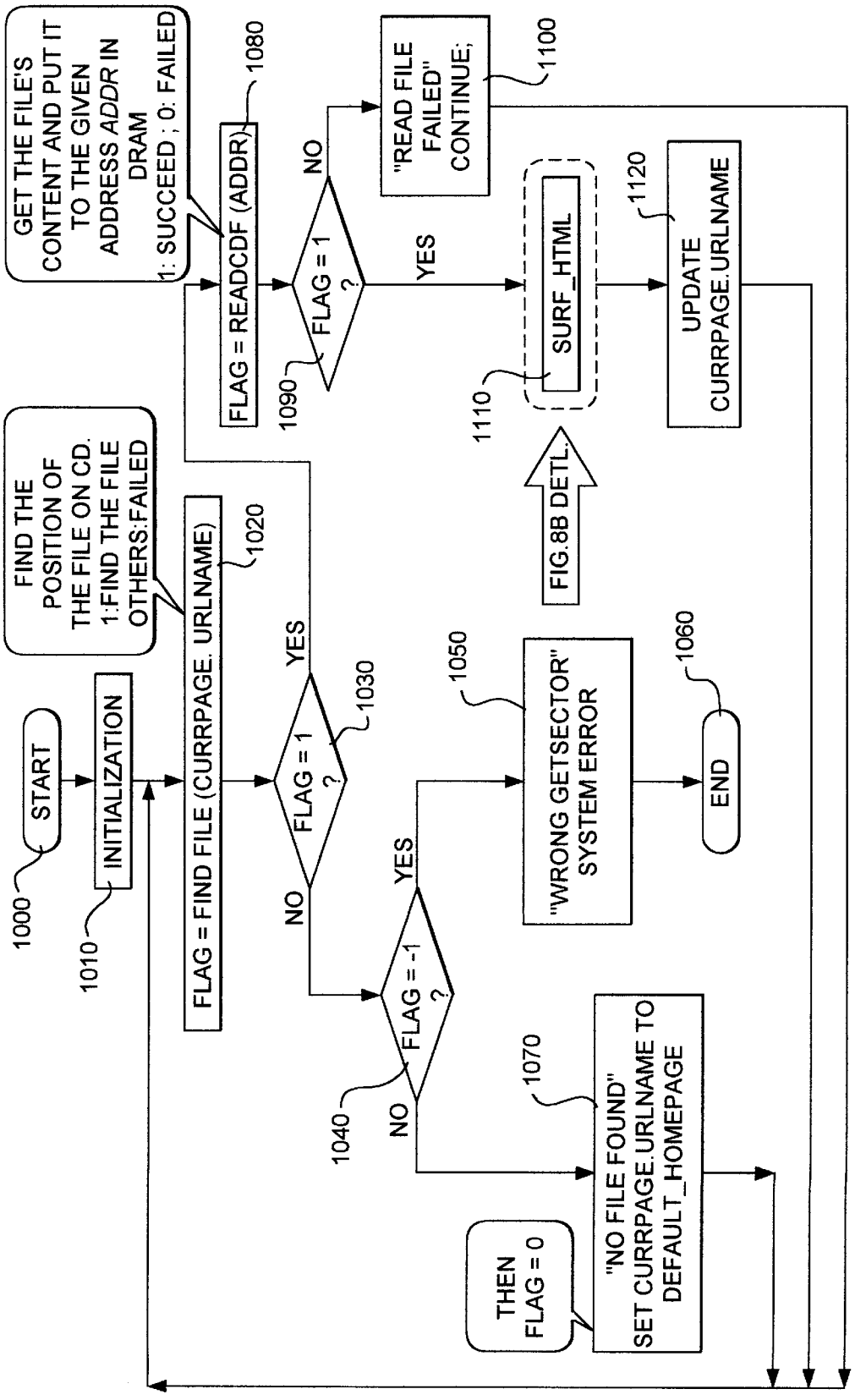

START
1000
INITIALIZATION
1010
FIND THE POSITION OF THE FILE ON CD. 1:FIND THE FILE OTHERS:FAILED
FLAG = FIND FILE (CURRPAGE. URLNAME)
1020
FLAG = 1 ?
1030
YES
NO
FLAG = -1 ?
1040
YES
NO
"WRONG GETSECTOR" SYSTEM ERROR
1050
END
1060
THEN FLAG = 0
"NO FILE FOUND" SET CURRPAGE.URLNAME TO DEFAULT_HOMEPAGE
1070
GET THE FILE'S CONTENT AND PUT IT TO THE GIVEN ADDRESS ADDR IN DRAM 1: SUCCEED ; 0: FAILED
FLAG = READCDF (ADDR)
1080
FLAG = 1 ?
1090
YES
NO
"READ FILE FAILED" CONTINUE;
1100
SURF_HTML
1110
FIG.8B DETL.
UPDATE CURRPAGE.URLNAME
1120

FIG. 8A

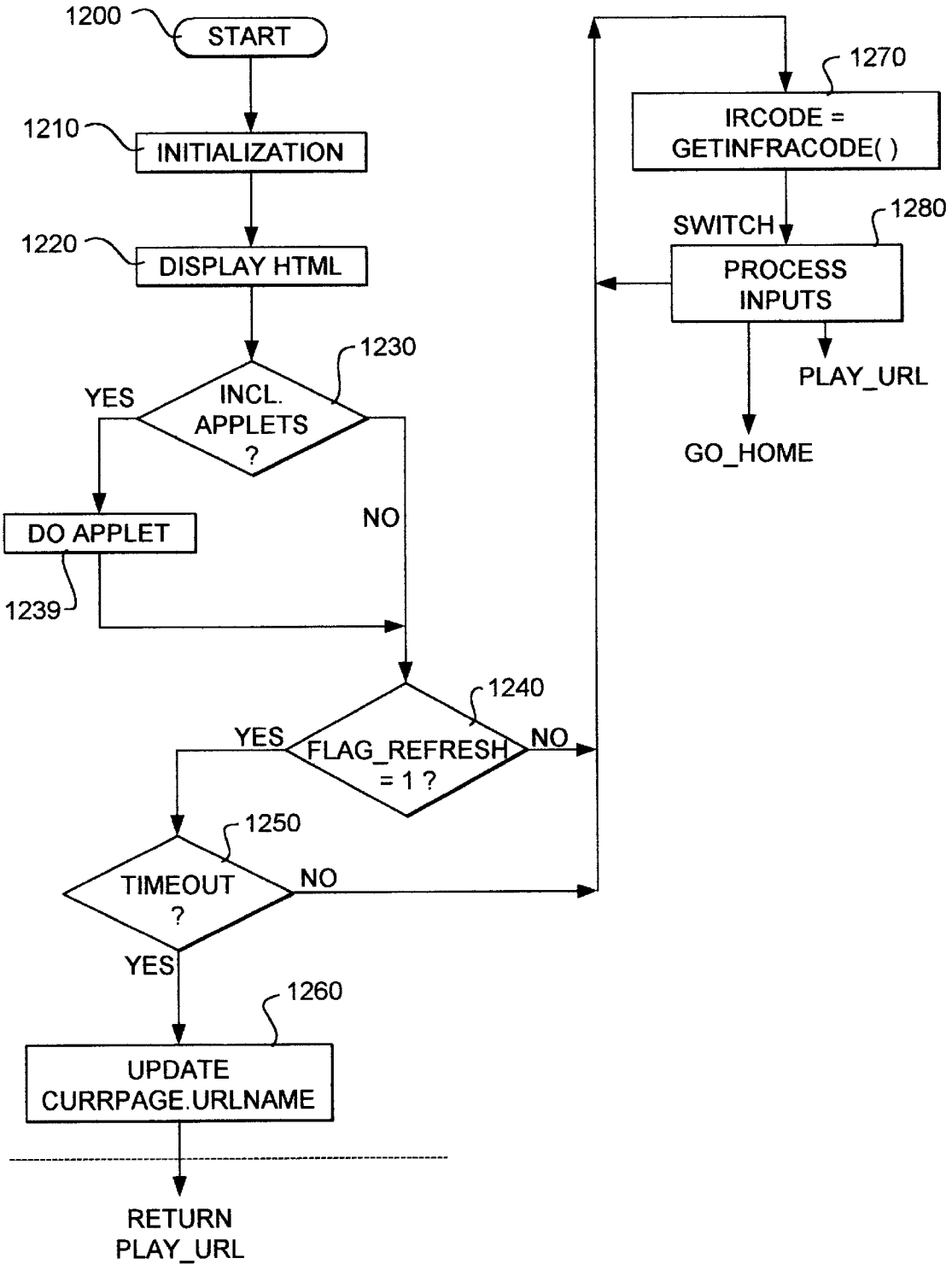

1200
START
1210
INITIALIZATION
1220
DISPLAY HTML
1230
INCL. APPLETS ?
YES
NO
DO APPLET
1239
1240
FLAG_REFRESH = 1 ?
YES
NO
1250
TIMEOUT ?
NO
YES
1260
UPDATE CURRPAGE.URLNAME
RETURN PLAY_URL
1270
IRCODE = GETINFRACODE( )
SWITCH
1280
PROCESS INPUTS
PLAY_URL
GO_HOME

FIG.8B

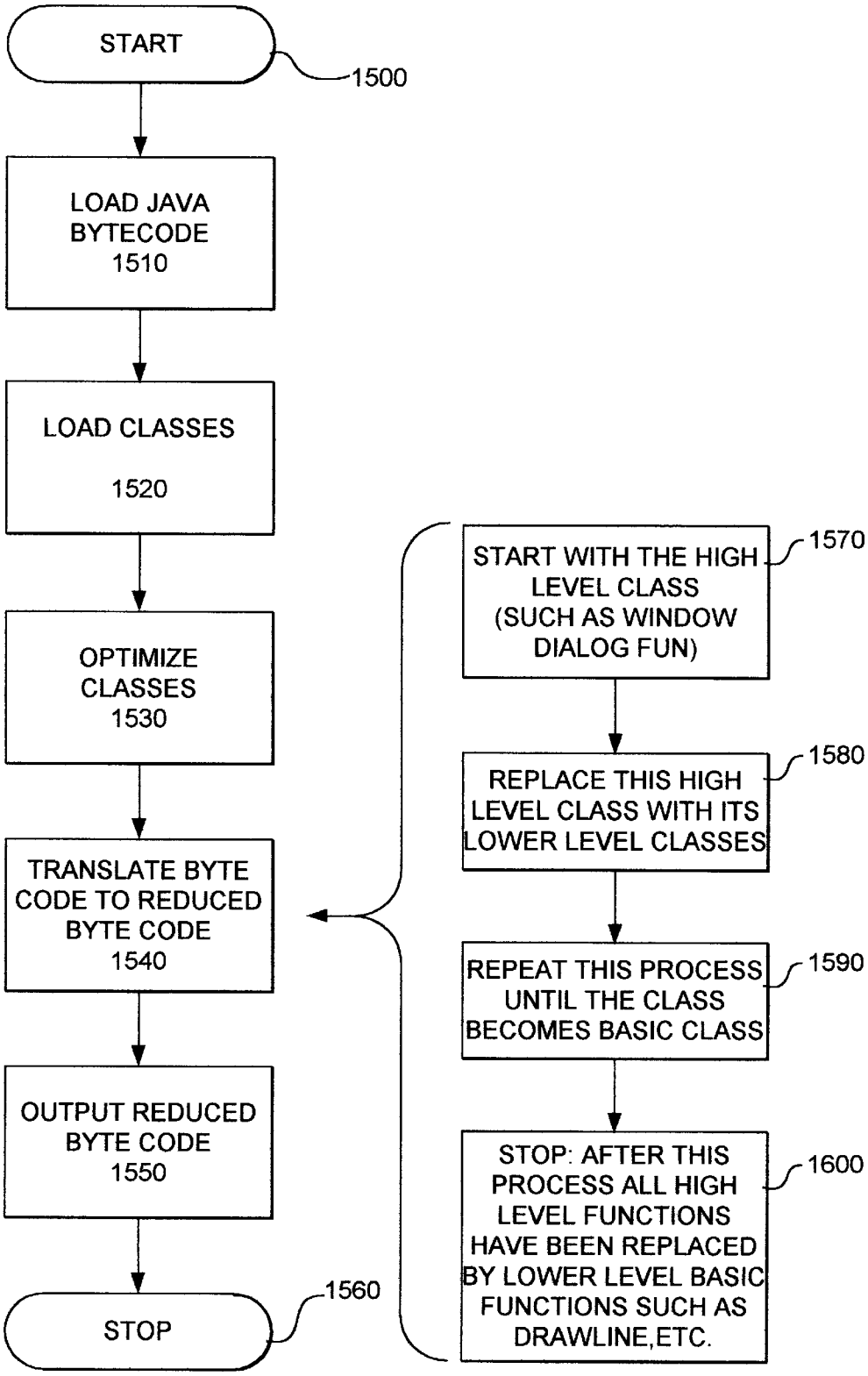

START
1500
LOAD JAVA BYTECODE 1510
LOAD CLASSES 1520
OPTIMIZE CLASSES 1530
TRANSLATE BYTE CODE TO REDUCED BYTE CODE 1540
OUTPUT REDUCED BYTE CODE 1550
STOP
1560
START WITH THE HIGH LEVEL CLASS (SUCH AS WINDOW DIALOG FUN)
1570
REPLACE THIS HIGH LEVEL CLASS WITH ITS LOWER LEVEL CLASSES
1580
REPEAT THIS PROCESS UNTIL THE CLASS BECOMES BASIC CLASS
1590
STOP: AFTER THIS PROCESS ALL HIGH LEVEL FUNCTIONS HAVE BEEN REPLACED BY LOWER LEVEL BASIC FUNCTIONS SUCH AS DRAWLINE,ETC.
1600

FIG. 9

FIG. 9A

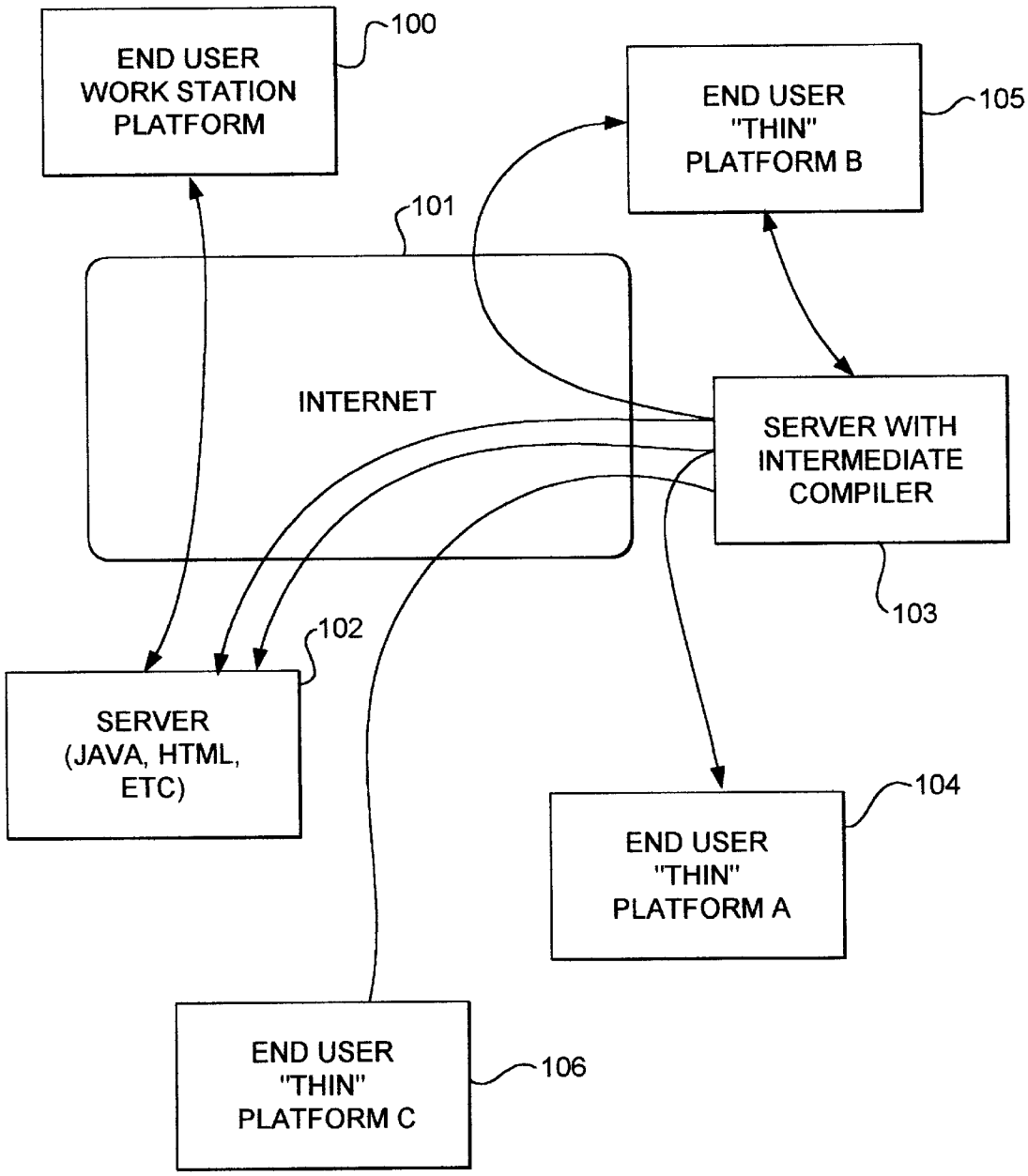

END USER
WORK STATION
PLATFORM
100
END USER
"THIN"
PLATFORM B
105
101
INTERNET
SERVER WITH
INTERMEDIATE
COMPILER
103
102
SERVER
(JAVA, HTML,
ETC)
104
END USER
"THIN"
PLATFORM A
END USER
"THIN"
PLATFORM C
106

FIG. 10

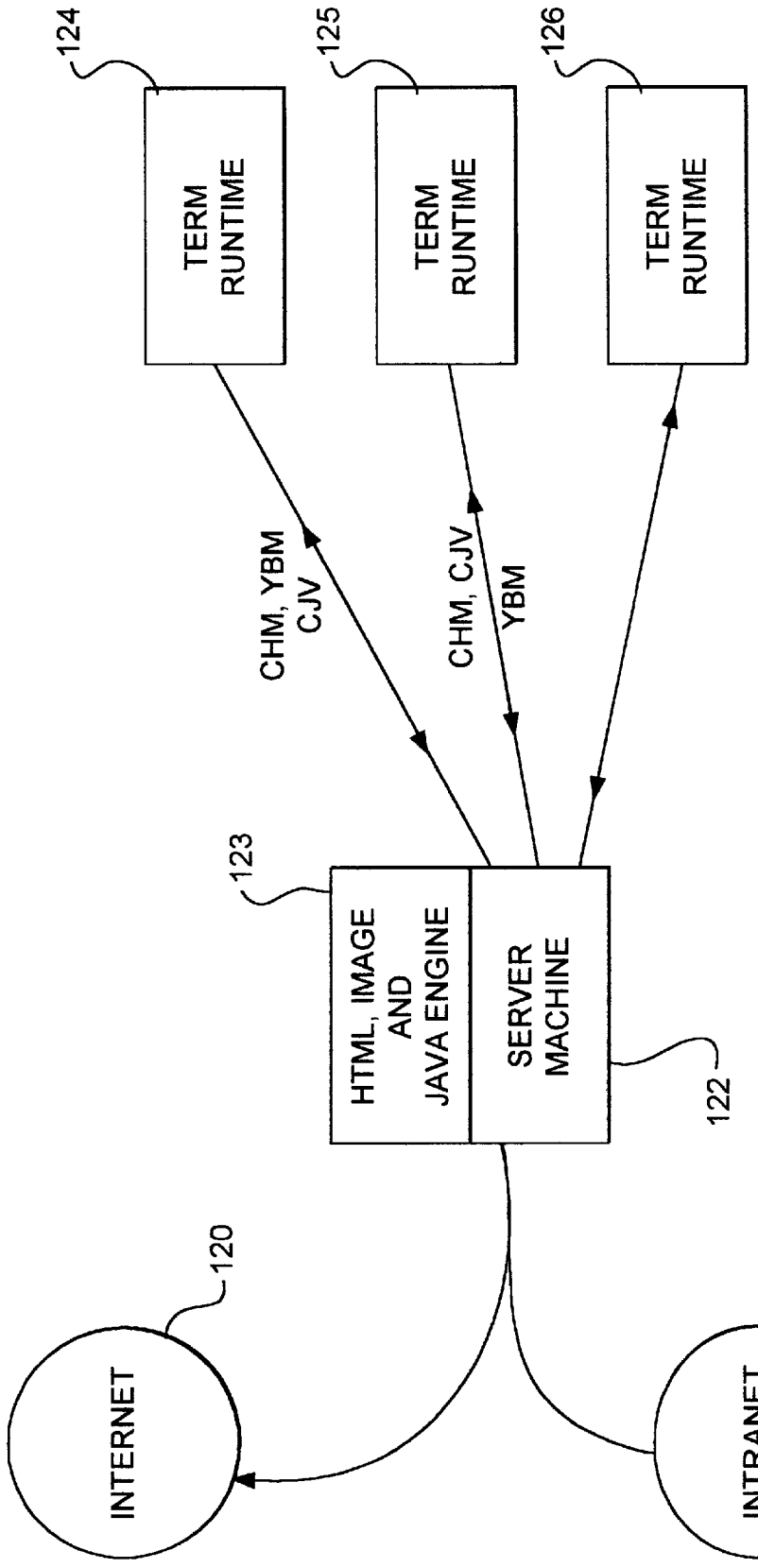

124
125
126
TERM RUNTIME
TERM RUNTIME
TERM RUNTIME
CHM, YBM CJV
CHM, CJV YBM
123
HTML, IMAGE AND JAVA ENGINE
SERVER MACHINE
122
120
121
INTERNET
INTRANET

FIG. 11

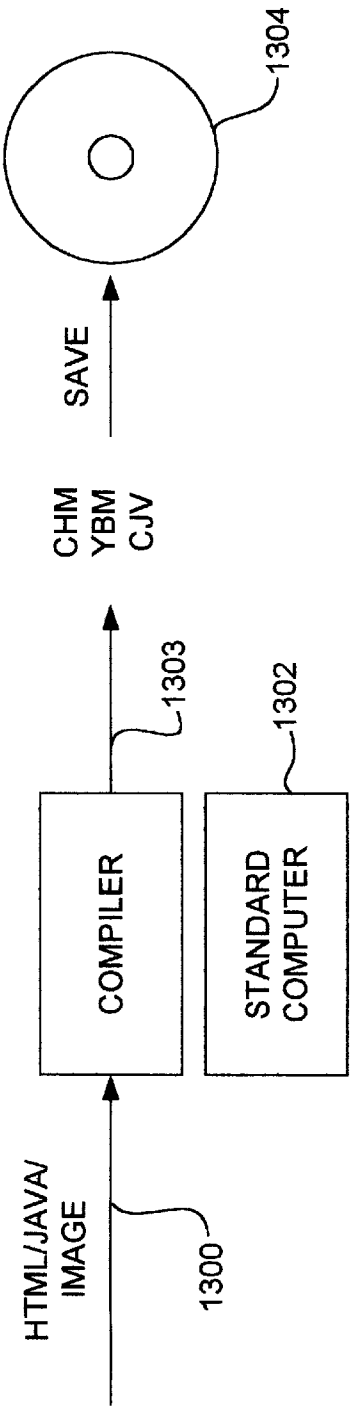


FIG. 12A

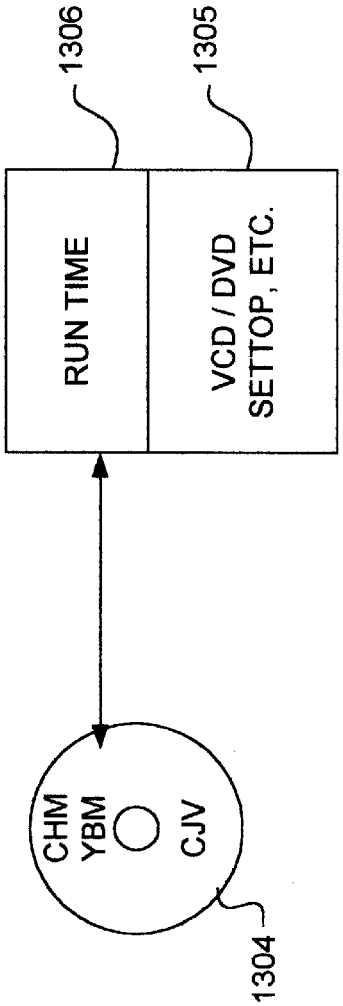


FIG. 12B

SYSTEM AND PROCESS FOR OBJECT RENDERING ON THIN CLIENT PLATFORMS

COPYRIGHT DISCLAIMER

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention generally relates to a method of providing full feature program processing according to a variety of standard language codes such as HTML, JAVA and other standard languages, for execution on a thin client platform. More particularly the invention relates to methods for compiling and rendering full feature standard HTML and JAVA programs into a format which is efficient for a limited processing resource platforms.

2. Description of Related Art

Standard HTML and JAVA programs, and other hypertext languages, are designed for computers having a significant amount of data processing resources, such as CPU speed and memory bandwidth, to run well. One feature of these object specifying languages is the ability to specify a graphic object for display using relative positioning. Relative positioning enables the display of the graphic object on displays having a wide range of dimensions, resolutions, and other display characteristics. However, relative positioning of graphic objects requires that the target device have computational resources to place the graphic object on the display at specific coordinates. Thus, there are a number of environments, such as TV set top boxes, hand held devices, digital video disk DVD players, compact video disk VCD players or thin network computer environments in which these standard object specifying languages are inefficient or impractical. The original HTML and JAVA programs run very slowly, or not at all, in these types of thin client environments. To solve these problems, simpler versions of HTML and JAVA have been proposed, which have resulted in scripting out some of the features. This trades off some of the nice functionality of HTML and JAVA, which have contributed to their wide acceptance. Furthermore, use in thin client environments of the huge number of files that are already specified according to these standards, is substantially limited.

SUMMARY OF THE INVENTION

The present invention provides a system and method for processing an Display object specified by an object specifying language such as HTML, JAVA or other languages relying on relative positioning, that require a rendering program utilizing a minimum set of resources, for use in a target device that has limited processing resources unsuited for storage and execution of the HTML rendering program, JAVA virtual machine, or other rendering engine for the standard. Thus, the invention can be characterized as a method for storing data concerning such an object that includes first receiving a data set specifying the object according to the object specifying language, translating the first data set into a second data set in an intermediate object language adapted for a second rendering program suitable for rendering by the target device that utilizes actual target display coordinates. The second data set is stored in a machine readable storage device, for later retrieval and execution by the thin client platform.

The object specifying language according to alternative embodiments comprises a HTML standard language or other hypertext mark up language, a JAVA standard language or other object oriented language that includes object specifying tools.

The invention also can be characterized as a method for sending data concerning such an object to a target device having limited processing resources. This method includes receiving the first data set specifying the object according to the first object specifying language, translating the first data set to a second data set in an intermediate object language, and then sending the second data set to the target device. The target device then renders the object by a rendering engine adapted for the intermediate object language. The step of sending the second data set includes sending the second data set across a packet switched network such as the Internet or the World Wide Web to the target device. Also, the step of translating according to one aspect of the invention includes sending the first data set across a packet switched network to a translation device, and executing a translation process on the translation device to generate the second data set. The second data set is then transferred from the translation device, to the target device, or alternatively from the translation device back to the source of the data, from which it is then forwarded to the target device.

According to other aspects of the invention, the step of translating the first data set includes first identifying the object specifying language of the first data set from among a set of object specifying languages, such as HTML and JAVA. Then, a translation process is selected according to the identified object specifying language.

According to yet another aspect of the invention, before the step of translating the steps of identifying the target device from among a set of target devices, and selecting a translation process according to the identified target device, are executed.

In yet another alternative of the present invention, a method for providing data to a target device is provided. This method includes requesting for the target device a first data set from a source of data, the first data set specifying the object according to the object specifying language; translating the first data set to a second data set in an intermediate language adapted for execution according to a second rendering program by the target device. The second data set is then sent to this target device. This allows a thin platform target device to request objects specified by full function HTML, JAVA and other object specifying languages, and have them automatically translated to a format suitable for rendering in the thin environment.

Thus, the present invention provides a method which uses a computer to automatically compile standard HTML, JAVA and other programs so that such programs can run both CPU and memory efficiently on a thin client platform such as a TV set top box, a VCD/DVD player, a hand held device, a network computer or an embedded computer. The automatic compilation maintains all the benefits of full feature HTML and JAVA or other language.

The significance of the invention is evident when it is considered that in the prior art, standard HTML and JAVA were reduced in features or special standards are created for the thin client environment. Thus according to the prior art

approaches, the standard programs and image files on the Internet need to be specially modified to meet the needs of special thin client devices. This is almost impossible considering the amount of HTML and JAVA formatted files on the Web. According to the invention each HTML file, compiled JAVA class file or other object specifying language data set is processed by a standard full feature HTML browser JAVA virtual machine, or other complementary rendering engine, optimized for a target platform on the fly, and then output into a set of display oriented language codes which can be easily executed and displayed on a thin client platform. Furthermore, the technique can use in general to speed up the HTML and JAVA computing in standard platforms.

Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description and the claims which follow.

BRIEF DESCRIPTION OF THE FIGURES

FIG. **1** is a simplified diagram of a end user thin platform for execution of a compiled code data source according to the present invention.

FIG. **2** is a simplified diagram of a user workstation and server for precompiling a composed data set according to the present invention.

FIG. **3** is a simplified diagram of a precompiler for a HTML formatted file.

FIG. **4** is a simplified diagram of a precompiler for a JAVA coded program.

FIG. **5** is a class inheritance hierarchy for a precompiler for HTML.

FIG. **6** is a flow chart for the HTML precompiler process.

FIG. **7** illustrates the compiled HTML structure according to one embodiment of the present invention.

FIGS. **8A–8B** illustrate a compiled HTML run time engine for execution on the thin platform according to the present invention.

FIG. **9** is a flow chart of the process for precompiling a JAVA program according to the present invention.

FIG. **9A** is a flow chart of one example process for translating the byte codes into a reduced byte code in the sequence of FIG. **9**.

FIG. **10** is a schematic diagram illustrating use of the present invention in the Internet environment.

FIG. **11** is a schematic diagram illustrating use of the present invention in a "network computer" environment.

FIG. **12A** is a schematic diagram illustrating use of the present invention in an off-line environment for producing a compiled format of the present invention and saving it to a storage medium.

FIG. **12B** illustrates the off-line environment in which the stored data is executed by thin platform.

DETAILED DESCRIPTION

A detailed description of preferred embodiments of the present invention is provided with respect to FIGS. **1–12A** and **12B**. FIGS. **1–2** illustrated simplified implementation of the present invention. FIGS. **3–9** and **9A** illustrate processes executed according to the present invention. FIGS. **10–12A** and **12B** illustrate the use of the present invention in the Internet environment or other packet switched network environment.

FIG. **1** illustrates a "thin" platform which includes a limited set of data processing resources represented by box **10**, a display **11**, and a "compiled code" rendering engine **12** for a display oriented language which relies on the data processing resources **10**. The end user platform **10** is coupled to a compiled code data source **13**. A compiled code data sources comprises, for example a VCD, a DVD, or other computer readable data storage device. Alternatively, the compiled code data source **13** consists of a connection to the World Wide Web or other packet switched or point-to-point network environment from which compiled code data is retrieved.

The limited data processing resources of the thin platform **10** include for example a microcontroller and limited memory. For example, 512k of RAM associated with a 8051 microcontroller, or a 66 MHz MIPS RISC CPU and 512k of dynamic RAM may be used in a representative thin platform. Other thin user platforms use low cost microprocessors with limited memory. In addition, other thin platforms may comprise high performance processors which have little resources available for use in rendering the compiled code data source. Coupled with the thin platform is a compiled code rendering engine **12**. This rendering engine **12** is a relatively compact program which runs efficiently on the thin platform data processing resources. The rendering engine translates the compiled code data source data set into a stream of data suitable for the display **11**. In this environment, the present invention is utilized by having the standard HTML or JAVA code preprocessed and compiled into a compiled HTML/JAVA format according to the present invention using the compiler engine described in more detail below on a more powerful computer. The compiled HTML/JAVA codes are saved on the storage media. A small compiled HTML/JAVA run time engine **12** is embedded or loaded into the thin client device. The run time engine **12** is used to play the compiled HTML/JAVA files on the thin platform **10**. This enables the use of a very small client to run full feature HTML or JAVA programs. The machine can be used both online, offline or in a hybrid mode.

FIG. **2** illustrates the environment in which the compiled code data is generated according to the present invention. Thus for example, a developer workstation **20** is coupled with image rendering tools such as HTML, JAVA, or other image tools **21**. The workstation **20** is coupled to a server for the composed data **22**. The server includes a precompiler **23** which takes the composed data and translates it into the compiled code data. Compiled code data is then sent to a destination **24** where it is stored or rendered as suits the needs of a particular environment. Thus for example, the destination may be a VCD, DVD or the World Wide Web.

According to the environment of FIG. **2** compiled HTML and JAVA "middleware" is implemented on an Internet server. Thus the thin set top box or other compiled code data destination **24** is coupled to the Internet/Intranet through the compiled HTML/JAVA middleware **22**, **23**. A small compiled HTML/JAVA run time engine is embedded in the thin destination device. All the HTML/JAVA files created in the workstation **20** go through the middleware server **22** to reach the thin client devices. The HTML/JAVA files are converted to the compiled format on the fly by the precompiler **23** on the middleware server **22**. The server **22** passes the compiled code onto the destination device. This allows for most software updates of precompiler techniques to be made in the server environment without the need to update the destination devices. Also, any changes in the run time engine that need to be executed in the destination device **24** can be provided through the link to the server **22**.

FIGS. **3** and **4** illustrate simplified diagrams of the precompilers for HTML and JAVA respectively. In FIG. **3**,

standard HTML files are received at input **500** and applied to a HTML parser **501**. The output of the parser is applied to a command module **502** which includes a HTML rendering engine **503**, and memory resident HTML objects optimizing engine **504**. The output consists of the compiled HTML output engine **505** generates the output with simplified graphics primatives.

The basic class inheritance hierarchy for the HTML precompiling is shown in FIG. **5**. The process of translating a HTML file to the compiled HTML structure of the present invention is illustrated in FIG. **6**. The process begins at point **800** in FIG. **6**. The first step involves loading the HTML file into the rendering device. Next information concerning the target device is loaded (step **820**). The HTML file is then parsed by searching for HTML tags, and based on such tags creating the class structure of FIG. **5** (step **830**).

Using the parameters of the target device, and the parsing class structure set up after the parsing process, the algorithm does HTML rendering based on a class hierarchy adapted to the dimensions and palette of the target device (step **840**). This fixes the coordinates of all the graphic objects specified by the HTML code on the screen of the target device. For example, the paragraphs are word wrapped, horizontal rules are placed in particular places, the colors are chosen, and other device specific processes are executed.

After the rendering, all the display information is saved back into the class structure of FIG. **5**. Finally the process goes through the class hierarchy and outputs the rendering information in compiled HTML format (step **850**). The compiled HTML instructions are primitives that define rectangles, text, bitmaps and the like and their respective locations. After outputting the compiled instructions, the process is finished (step **860**).

A simplified pseudo code for the HTML compilation process is provided in Table 1.

TABLE 1

Copyright EnReach 1997

```
function convert_html (input : pointer) : chtmlfile;
      // this takes a pointer to an HTML file and translates it into a CHTML binary file
begin
      deviceInfo := LoadDeviceInfo( );    // Loads size and colors of target device
      Parse HTML file                     // use a parser to break the HTML file up into
                                          // tags represented in a fashion suitable for display
      For each HTML tag (<IMG . . . > = 1 tag, <P> a paragraph </P> = 1 tag),
select a sequence of CHTML instructions to render the tag on the output device.
As instructions are selected, colors and positioning are optimized based on the
device size and palette.
            CHTML instructions include:
                  TITLE          string
                  TEXT           formatted text at a specific position,
                                 complex formatting will
                                 require multiple CHTML TEXT instructions
                  IMAGE          image information including image-map,
                                 animation info, image data
                  ANCHOR         HTML reference
            Basic geometric instructions such as: SQUARE, FILLEDSQUARE, CIRCLE,
            FILLEDCIRCLE, and LINE, permit the complex rendering required by some
            HTML instructions to be decomposed into basic drawing instructions. For
            example, the bullets in front of lists can be described in CHTML instructions
            as squares and circles at specific locations.
            CHTML instructions including TEXT and IMAGE instructions can be
            contained within anchors. The CHTML compiler must properly code all
            instructions to indicate if an instruction is contained in an anchor.
      The CHTML instructions can then be written to the output file along with some header
      information.
end;
```

Table 2 sets forth the data structure for the precompiling process.

TABLE 2

Copyright EnReach 1997

```
/* HTML font structure */
typedef struct tagHTMLFont
{
      char name[64];
      int size;
      int bold;
      int italic;
      int underline;
      int strikeout;
} HTMLFont;
/* FG point structure */
typedef struct tagFGPoint
```

TABLE 2-continued


```
{
    int fX;
    int fY;
} FGPoint;
/* FG rectangle structure */
typedef struct tagFGRect
{
    int fLeft;
    int fTop;
    int fRight;
    int fBottom;
} FGRect;
/* html node types, used by hType attribute in HTML_InfoHead structure */
#define HTML_TYPE_TITLE     0           /* title of the html page */
#define HTML_TYPE_TEXT      1           /* text node */
#define HTML_TYPE_CHINESE   2           /* chinese text node */
#define HTML_TYPE_IMAGE     3           /* image node */
#define HTML_TYPE_SQUARE    4           /* square frame */
#define HTML_TYPE_FILLEDSQUARE 5        /* filled square */
#define HTML_TYPE_CIRCLE    6           /* circle frame */
#define HTML_TYPE_FILLEDCIRCLE 7        /* filled circle */
#define HTML_TYPE_LINE      8           /* line */
#define HTML_TYPE_ANCHOR    9           /* anchor node */
#define HTML_TYPE_ANIMATION    10       /* animation node */
#define HTML_TYPE_MAPAREA   11          /* client side image map area node */
/* header info of compiled html file */
typedef struct tagHTML_FileHead
{
    unsigned int fBgColor;                  /* background color index */
    unsigned int fPaletteSize;              /* size of palette */
} HTML_FileHead;
/* header info of each html node */
typedef struct tagHTML_InfoHead
{
    unsigned int hType;                     /* type of the node */
    unsigned int hSize;                     /* size of htmlInfo */
} HTML_InfoHead;
/* html info structure */
typedef struct tagHTML_Info
{
    HTML_InfoHead htmlHead;                 /* header info */
    unsigned char htmlInfo[1];              /* info of the html node */
} HTML_Info;
/* html title structure */
typedef struct tagHTML_Title
{
    unsigned int textLen;                   /* length of text buffer */
    char textBuffer[1];                     /* content of text buffer */
} HTML_Title;
/* html text structure */
typedef struct tagHTML_Text
{
    FGPoint dispPos;                        /* display coordinates */
    int anchorID;                   /* anchor id if it's inside an anchor, -1 if not */
    HTMLFont textFont;                      /* font of the text */
    unsigned int textColor;                 /* color index of the text */
    unsigned int textLen;                   /* length of text buffer */
    char textBuffer[1];                     /* content of text buffer */
} HTML_Text
/* html chinese structure */
typedef struct tagHTML_Chinese
{
    FGPNT dispPos;                          /* display coordinates */
    int anchorID;                   /* anchor id if it's inside an anchor, -1 if not */
    unsigned int textColor;                 /* color index of the text */
    unsigned int bufLen;                    /* length of the bitmap buffer (16* 16) */
    char textBuffer[1];                     /* content of text buffer */
} HTML_Chinese;
/* html image structure */
typedef struct tagHTML_Image
{
    FGRect dispPos;                         /* display coordinates */
    int anchorID;                   /* anchor id if it's inside an anchor, -1 if not */
```

TABLE 2-continued

Copyright EnReach 1997

```
    int animationID; /* animation id if it supports animation, -1 if not */
    int animationDelay;                    /* delay time for animation */
    char mapName[64];              /* name of client side image map, empty if no
image map */
    void *data;                            /* used to store image
data */
    unsigned int fnameLen;             /* length of the image file name */
    char fname[1];                         /* image filename */
} HTML_Image;
/* square structure */
typedef strnct tagHTML_Square
{
    FGRect dispPos;                        /* display coordinates */
    unsigned int borderColor; /* border color index */
} HTML_Square;
/* filled square structure */
typedef struct tagHTML_FilledSquare
{
    FGRect dispPos;                        /* display coordinates */
    unsigned int brushColor; /* the inside color index */
} HTML_FilledSquare;
/* circle structure */
typedef struct tagHTML_Circle
{
    FGRect dispPos;                        /* display coordinates */
    unsigned int borderColor; /* border color index */
} HTML_Circle
/* circle structure */
typedef struct tagHTML_FilledCircle
{
    FGRect dispPos;                        /* display coordinates */
    unsigned int brushColor; /* the inside color index */
} HTML_FilledCircle;
/* line structure */
typedef struct tagHTML_Line
{
    FGPoint startPos;                      /* line starting position */
    FGPoint endPos;                        /* line end position */
    int style;                             /* style of the line (solid, dashed, dotted,
etc.) */
    unsigned int penColor;                 /* pen color index */
} HTML_Line;
/* anchor structure */
typedef struct tagHTML_Anchor
{
    int anchorID;                          /* id of the anchor */
    unsigned int hrefLen;.             /* length of href */
    char href[1];                          /* url of the anchor */
} HTML_Anchor;
/* animation structure */
typedef struct tagHTML_Animation
{
    int animationID;                   /* id of the animation */
    unsigned int frameTotal; /* total number of animation frames */
    long runtime;                          /* animation runtime */
} HTML_Animation;
#define SHAPE_RECTANGLE 0
#define SHAPE_CIRCLE 1
#define SHAPE_POLY 2
/* image map area structure */
typedef struct tagHTML_MapArea
{
    char mapName[64];                      /* name of client side image map
*/
    intshape;                              /* shape of the area */
    int numVer;                            /* number of vertix */
    int coords[6][2];              /* coordinates */
    unsigned int hrefLen;          /* length of href */
    char href[1];                          /* url the area pointed to */
} HTML_MapArea;
```

An example routine for reading this file into the thin platform memory follows in Table 3.

TABLE 3


```
reading this file:
#define BLOCK_SIZE 256
/* returns number of nodes */
long read_chm(const char *filename,          /* input: .chm file name */
    HTML_Info ***ppNodeList, /* output: array of (HTML_Info *)
                                                    including anchors. */
    YUVQUAD **ppPalette,                                /* output: page palette */
    unsigned int *palette_size)                         /* output: palette size */
{
    int fd;
    char head[12];
    long total_nodes = 0;
    long max_nodes = 0;
    HTML_FileHead myFileHead;
    HTML_InfoHead myInfoHead;
    HTML_Info *pNodeInfo;
    void *pNodeData;
    long i;
    HTML_InfoHead *pHead;
    if (!ppNodeList || !ppPalette || !palette_size)
        return 0;
    (*ppNodeList) = NULL;
    (*ppPalette) = NULL;
    (*palette_size) = 0
    /* open file */
    fd = _open(filename,_O_BINARY|_O_RDONLY);
    if(fd < 0)
        return 0;
    /* read header and check for file type */
    if (_read(fd, head, 10) != 10)
    {
        _close(fd);
        return 0;
    }
    if (strncmp(head, "<COMPHTML>", 10))
    {
        _close(fd);
        return 0;
    }
    /* read file header */
    if(_read(fd, &myFileHead, sizeof(HTML_FileHead)) !=
sizeof(HTML_FileHead))
    {
        _close(fd);
        return 0;
    }
    (*palette_size) = myFileHead.fpaletteSize;
    /* read the palette */
    if ((*palette_size) > 0)
    {
        (*ppPalette) = (YUVQUAD *)malloc(sizeof(YUVQUAD)*
(*palette_size));
        if (_read(fd, (*ppPalette), sizeof(YUVQUAD) * (*palette_size))
            != (int) (sizeof(YUVQUAD) * (*palette_size)))
        {
            _close(fd);
            return 0;
        }
    }
    /* read anchors along with other html nodes */
    while (1)
    {
        if (_read(fd, &myInfoHead, sizeof(HTML_InfoHead))
            != sizeof(HTML_InfoHead))
        {
            break;
        }
        if (myInfoHead.hSize > 0)
        {
            pNodeInfo = (HTML_Info *) malloc(myInfoHead.hSize +
sizeof(HTML_InfoHead));
            if (!pNodeInfo)
                break;
            memcpy(pNodeInfo, &myInfoHead,
sizeof(HTML_InfoHead));
            if (_read(fd, &pNodeInfo[sizeof(HTML_InfoHead)],
myInfoHead.hSize)
```

TABLE 3-continued


```
                != (int)myInfoHead.hSize)
            {
                break;
            }
            /* check if we need to do memory allocation */
            if (total_nodes >= max_nodes)
            {
                if(!max_nodes)
                {
                    /* no node in the list yet */
                    (*ppNodeList) = (HTML_Info **)
malloc(
                        sizeof(HTML_Info *)*
BLOCK_SIZE);
                }
                else
                {
                    (*ppNodeList) = (HTML_Info **)
realloc((*ppNodeList),
                        max_nodes + sizeof(HTML_Info
*) * BLOCK_SIZE);
                }
                if (!(*ppNodeList))
                    break;
                max_nodes += BLOCK_SIZE;
            }
            (*ppNodeList)[total_nodes] = pNodeInfo;
            total_nodes++;
        }
    }
    _close(fd);
    /* test our data */
    for (i = 0; i < total_nodes; i++)
    {
        pNodeInfo = (*ppNodeList)[i];
        pHead = (HTML_InfoHead *) pNodeInfo;
        pNodeData = pNodeInfo + sizeof(HTML_InfoHead);
        if(pHead->hType == HTML_TYPE_TEXT)
        {
            HTML_Text *pText = (HTML_Text *) pNodeData;
        }
        else if(pHead->hType == HTML_TYPE_IMAGE)
        {
            HTML_Image *pImage = (HTML_Image *) pNodeData;
            if (pImage->fnameLen > 0)
            {
                /* load the image file */
                pImage->data = load_ybm(pImage->fname);
            }
        }
        else if (pHead->hType == HTML_TYPE_ANCHOR)
        {
            HTML_Anchor *pAnchor = (HTML_Anchor *)
pNodeData;
        }
        else if(pHead->hType == HTML_TYPE_ANIMATION)
        {
            HTML_Animation *pAnimation = (HTML_Animation *)
pNodeData;
        }
        else if(pHead->hType == HTML_TYPE_MAPAREA)
        {
            HTML_MapArea *pMapArea = (HTML_MapArea *)
pNodeData;
        }
        else if (pHead->hType == HTML_TYPE_LINE)
        {
            HTML_Line *pLine = (HTML_Line *) pNodeData;
        }
        else if(1)Head->hType == HTML_TYPE_SQUARE)
        {
            HTML_Square *pSquare = (HTML_Square *) pNodeData;
        }
        else if(pHead->hType == HTML_TYPE_CIRCLE)
        {
            HTML_Circle *pCircle = (HTML_Circle *) pNodeData;
        }
```

TABLE 3-continued

Copyright EnReach 1997

```
            else if (pHead->hType == HTML_TYPE_FILLEDSQUARE)
            {
                HTML_FilledSquare *pFilledSquare =
(HTML_FilledSquare *) pNodeData;
            }
            else if (pHead->hType == HTML_TYPE_FILLEDCIRCLE)
            {
                HTML_FilledCircle *pFilledCircle = (HTML_FilledCircle
*) pNodeData;
            }
            else if(pHead->hType == HTML_TYPE_TITLE)
            {
                HTML_Title *pTitle = (HTML_Title *) pNodeData;
            }
    }
    return total_nodes;
}
```

The compiled HTML file structure is set forth in FIG. **7** as described in Table 2. The file structure begins with a ten character string COMPHTML **900**. This string is followed by a HTML file header structure **901**. After the file header structure, a YUV color palette is set forth in the structure **902** this consists of an array of YUVQUAD values for the target device. After the palette array, a list **903** of HTML information structures follows. Usually the first HTML information structure **904** consists of a title. Next, a refresh element typically follows at point **905**. This is optional. Next in the line is a background color and background images if they are used in this image. After that, a list of display elements is provided in proper order. The anchor node for the HTML file is always in front of the nodes that it contains. An animation node is always right before the animation image frames start. The image area nodes usually appear at the head of the list.

The HTML file header structure includes a first value BgColor at point **906** followed by palette size parameters for the target device at point **907**. The YUVQUAD values in the color palette consist of a four word structure specifying the Y, U, and V values for the particular pixel at points **908–910**. The HTML information structures in the list **903** consist of a type field **911**, a size field **912**, and the information which supports the type at field **913**. The type structures can be a HTML_Title, HTML_Text, HTML_Chinese, HTML_Xxge, HTML_Square, HTML_FilledSquare, HTML_Circle, HTML_FilledCircle, HTML_Line, HTML_Author, HTML_Animation, . . .

Functions that would enable a thin platform to support viewing of HTML-based content pre-compiled according to the present invention includes the following:

General graphics functions

int DrawPoint (int x, int y, COLOR color, MODE mode);

int DrawLine (int x1, int y1, int x2, int y2, COLOR color, MODE mode);

int DrawRectangle(int x1, int y1, int x2, int y2, COLOR color, MODE mode);

int FillRectangle(int x1, int y1, int x2, int y2, COLOR color, MODE mode);

int ClearScreen(COLOR color);

Color palette

int ChangeYUVColorPalette( );

Bitmap function

int BitBlt(int dst_x1, int dst_y1, int dst_x2, int dst_y2, unsigned char *bitmap, MODE mode);

String drawing functions

int GetStringWidth(char *str, int len);

int GetStringHeight(char *str, int len);

int DrawStringOnScreen(int x, int y, char *str, int len, COLOR color, MODE mode);

Explanation

All (x, y) coordinates are based on the screen resolution of the target display device (e.g. 320×240 pixels).

COLOR is specified as an index to a palette.

MODE defines how new pixels replace currently displayed pixels (COPY, XOR, OR, AND).

Minimum support for DrawLine is a horizontal or vertical straight line, although it would be nice to have support for diagonal lines.

The ChangeYUVColorPalette function is used for every page.

BitBlt uses (x1, y1) and (x2, y2) for scaling but it is not a requirement to have this scaling functionality.

String functions are used for English text output only. Bitmaps are used for Chinese characters.

FIGS. **8**A and **8**B set forth the run time engine suitable for execution on a thin client platform for display of the compiled HTML material which includes the function outlined above in the "display" step **1220** of FIG. **8**B.

The process of FIG. **8**A starts at block **1000**. The ran time engine is initialized on the client platform by loading the appropriate elements of the run time engine and other processes known in the art (step **1010**). The next step involves identifying the position of the file, such as on the source CD or other location from which the file is to be retrieved and setting a flag (step **1020**). The flag is tested at step **1030**. If the flag is not set, then the algorithm branches to block **1040** at which the flag is tested to determine whether it is −1 or not. If the flag is −1, then the algorithm determines that a system error has occurred (step **1050**) and the process ends at step **1060**. If the flag at step **1040** is not −1, then the file has not been found (step **1070**). Thus after step **1070** the algorithm returns to step **1020** to find the next file or retry.

If at step **1030**, the flag is set to 1 indicating that the file was found, then the content of the file is retrieved using a program like that in Table 3, and it is stored at a specified address. A flag is returned if this process succeeds set equal to 1 otherwise it is set equal to 0 (step **1080**). Next the flag is tested (step **1090**). If the flag is not equal to 1 then reading

of the file failed (step **1100**). The process then returns to step **1020** to find the next file or retry.

If the flag is set to 1, indicating that the file has been successfully loaded into the dynamic RAM of the target device, then the "Surf_HTML" process is executed (step **1110**). The details of this process are illustrated in FIG. **8**B. Next the current page URL name is updated according to the HTML process (step **1120**). After updating the current URL name, the process returns to step **1020** to find the next file.

FIG. **8**B illustrates the "Surf_HTML" process of step **1110** in FIG. **8**A. This process starts at point **1200**. The first part is initialization step **1210**. A display routine is executed at step **1220** having the fixed coordinate functions of the precompiled HTML data set. First, the process determines whether applets are included in the file (step **1230**). If they are included, then the applet is executed (step **1240**). If no applets are included or after execution of the applet, then a refresh flag is tested (step **1240**). If the flag is equal to 1, then it is tested whether a timeout has occurred (step **1250**). If a timeout has occurred, then the current page is updated (step **1260**) and the process returns set **1210** of FIG. **8**B, for example.

If at block **1240** the refresh flag was not equal to 1, or at block **1250** the timeout had not expired, then the process proceeds to step **1270** to get a user supplied input code such as an infrared input signal provided by a remote control at the target device code. In response to the code, a variety of process are executed as suits a particular target platform to handle the user inputs (step **1280**). The process returns a GO_HOME, or a PLAY_URL command, for example, which result in returning the user to a home web page or to a current URL, respectively. Alternatively the process loops to step **1270** for a next input code.

As mentioned above, FIG. **4** illustrates the JAVA precompiler according to the present invention. The JAVA precompiler receives standard full feature JAVA byte codes as input on line **600**. Byte codes are parsed at block **601**. A JAVA class loader is then executed at block **602**. The classes are loaded into a command module **603** which coordinates operations of a JAVA virtual machine **604**, a JAVA garbage collection module **605**, and a JAVA objects memory mapping optimizing engine **606**. The output is applied by block **607** which consists of a compiled JAVA bytecode format according to the present invention.

The process is illustrated in FIG. **9** beginning at block **1500**. First the JAVA bytecode file is loaded (block **1510**). Next, the JAVA classes are loaded based on the interpretation of the bytecode (step **1520**). Next the classes are optimized at step **1530**. After optimizing the classes, the byte codes are translated to a reduced bytecode (step **1540**). Finally the reduced bytecode is supplied (step **1550**) and the algorithm stops at step **1560**. Basically the process receives a JAVA source code file which usually has the format of a text file with the extension JAVA. The JAVA compiler includes a JAVA virtual machine plus compiler classes such as SUN-.TOOLS.JAVAC which are commercially available from Sun Micro Systems. The JAVA class file is parsed which typically consists of byte codes with the extension .CLASS. A class loader consists of a parser and bytecode verifier and processes other class files. The class structures are processed according to the JAVA virtual machine specification, such as the constant pool, the method tables, and the like. An interpreter and compiler are then executed. The JAVA virtual machine executes byte codes in methods and outputs compiled JAVA class files starting with "Main". The process of loading and verifying classes involves first finding a class. If the class is already loaded a read pointer to the class is returned, if not, the class is found from the user specified class path or directory, in this case a flash memory chunk. After finding the class, the next step is executed. This involves loading the bytes from the class file. Next, class file bytes are put into a class structure suitable for run time use, as defined by the JAVA virtual machine specification. The process recursively loads and links the class to its super classes. Various checks and initializations are executed to verify and prepare the routine for execution. Next, initialization is executed for the method of the class. First the process ensures that all the super classes are initialized, and then cause the initialization method for the class. Finally, the class is resolved by resolving a constant pool entry the first time it is encountered. A method is executed with the interpreter and compiler by finding the method. The method may be in the current class, its super class or other classes as specified. A frame is created for the method, including a stack, local variables and a program counter. The process starts executing the bytecode instructions. The instructions can be stack operations, branch statements, loading/storing values, from/to the local variables or constant pool items, or invoking other methods. When an invoked method is a native function, the implemented platform dependent function is executed.

In FIG. **9**A, the process of translating JAVA byte codes into compiled byte codes (step **1504** of FIG. **9**) is illustrated. According to the process FIG. **9**A, the high level class byte codes are parsed from the sequence. For example, Windows dialog functions are found (**1570**). The high level class is replaced with its lower level classes (**1580**). This process is repeated until all the classes in the file become basic classes (**1590**). After this process, all the high level functions have been replaced by lower level level basic functions, such as draw a line, etc. (**1600**).

JAVA byte codes in classes include a number of high level object specifying functions such as a window drawing function and other tool sets. According to the present invention, these classes are rendered by the precompiler into a set of specific coordinate functions such as those outlined above in connection with the HTML precompiler. By precompiling the object specifying functions of the JAVA byte code data set, significant processing resources are freed up on the thin client platform for executing the other programs carried in a JAVA byte code file. Furthermore, the amount of memory required to store the run time engine and JAVA class file for the thin client platform according to the present invention which is suitable for running a JAVA byte code file is substantially reduced.

FIG. **10** illustrates one environment in which use of the present invention is advantageous. In particular, in the Internet environment a wide variety of platforms are implemented. For example, an end user workstation platform **100** is coupled to the Internet **101**. An Internet server platform **102** is also coupled to the Internet **101** and includes storage for JAVA data sets, HTML data sets, and other image files. A server **103** with an intermediate compiler according to the present invention for one or more of the data sets available in the Internet is coupled to the Internet **101** as well. A variety of "thin" platforms are also coupled to the Internet and/or the server **103**. For example, an end user thin platform A **104** is coupled to the server **103**. End user thin platform B **105** is coupled to the server **103** and to the Internet **101**. End user thin platform C **106** is coupled to the Internet **101** and via the Internet all the other platforms in the network. A variety of scenarios are thus instituted. The source of data sets for end user platform C **106** consists of the World Wide Web. When it requests a file from server

102, the file is first transferred to the intermediate compiler at server 103, and from server 103 to the end user platform 106. End user platform A 104 is coupled directly to the server 103. When it makes a request for a file, the request is transmitted to the server 103, which retrieves the file from its source at server 102, translates it to the compiled version and sends it to platform A 104. End user platform B is coupled to both the server 103 and to the Internet 101. Thus, it is capable of requesting files directly from server 102. The server 102 transmits the file to server 103 from which the translated compiled version is sent to platform B 105. Alternatively, platform B may request a file directly from server 103 which performs all retrieval and processing functions on behalf of platform B.

FIG. 11 illustrates an alternative environment for the present invention. For example, the Internet 120 and an Intranet 121 are connected together. A server 122 is coupled to the Intranet 121 and the Internet 120. The server 122 includes the HTML and JAVA intermediate compiling engines according to the present invention as represented by block 123. The server 122 acts as a source of precompiled data sets for thin client platforms 124, 125 and 126 each of which has a simplified run time engine suitable for the compiled data sets. Thus the powerful HTML/JAVA engine resides on the network server 122. The thin network computers 124, 125, 126 are connected to the server have only the simplified run time engine for the compiled image set. Thus, very small computing power is required for executing the display. Thus computing tasks are done using the network server, but displayed on a thin network computer terminals 124–126.

FIGS. 12A and 12B illustrate the off-line environment for use of the present invention. In FIG. 12A, the production of the compiled files is illustrated. Thus, a standard object file, such as an HTML or JAVA image, is input online 1300 to a compiler 1301 which runs on a standard computer 1302. The output of the compiler on line 1303 is the compiled bitmap, compiled HTML or compiled JAVA formatted file. This file is then saved on a non-volatile storage medium such as a compact disk, video compact disk or other storage medium represented by the disk 1304.

FIG. 12B illustrates the reading of the data from the disk 1304 and a thin client such as a VCD box, a DVD box or a set top box 1305. The run time engine 1306 for the compiled data is provided on the thin platform 1305.

Thus, off-line fill feature HTML and JAVA processing is provided for a run time environment on a very thin client such as a VCD/DVD player. The standard HTML/JAVA objects are pre-processed and compiled into the compiled format using the compiler engine 1301 on a more powerful computer 1302. The compiled files are saved on a storage medium such as a floppy disk, hard drive, a CD-ROM, a VCD, or a DVD disk. A small compiled run time engine is embedded or loaded into the thin client device. The run time engine is used to play the compiled files. This enables use of a very small client for running full feature HTML and JAVA programs. Thus, the machine can be used in both online, and off-line modes, or in a hybrid mode.

The foregoing description of a preferred embodiment of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. It is intended that the scope of the invention be defined by the following claims and their equivalents.

What is claimed is:

1. A method of translating a document on a first device for use on a second device, the document being in a standard HTML language, the method comprising:
   - reading the document;
   - reading a profile describing characteristics of the second device, the profile including a display resolution and a supported image format; and
   - translating the document on the first device according to the profile, the translating including
     - retrieving a plurality of images referenced by the document,
     - generating a color palette for the second platform using the plurality of images and the document,
     - executing the document according to the standard HTML language using the profile to generate a plurality of drawing instructions for displaying the document on the second device,
     - translating the plurality of images from respective formats to the supported image format, and
     - outputting a translated document, the translated document including at least a reference to the color palette, the plurality of images in the supported image format, and the plurality of drawing instructions.

2. The method of claim 1 wherein the reading the document further comprises retrieving the document from a world wide web (WWW) site based on a uniform resource locator (URL).

3. The method of claim 1 wherein the profile includes a maximum number of colors for the color palette.

4. The method of claim 3 wherein the generating the color palette using the plurality of images and the document comprises:
   - creating a set of colors comprised of all colors used in the plurality of images and all colors used in the document;
   - reducing the set of colors to contain no more than the maximum number of colors for the color palette.

5. The method of claim 1 wherein the document includes a plurality of references to a plurality of images, each of the plurality of references comprising a URL, and the retrieving a plurality of images referenced by the document further comprises retrieving respective images using the plurality of references.

6. The method of claim 1 wherein the executing the document according to the standard HTML language using the profile to generate a plurality of drawing instructions for displaying the document on the second device further comprises:
   - executing the document for display on the second device according to the display resolution;
   - positioning HTML elements in the document according to the display resolution;
   - word wrapping HTML text elements in the document according to the display resolution; and
   - generating a plurality of text drawing elements.

7. The method of claim 6 wherein the generating a plurality of text drawing elements further comprises generating a text element for each text segment, a text segment comprised of one or more characters from the document, the one or more characters sharing a font, a size, a style, and a color, and the text segment occupying not more than one line in the font at the size in the style, each text element including an absolute position at which the text segment should be displayed on the second device.

8. The method of claim 6 further comprising generating a plurality of graphics drawing elements including:

generating a plurality of line elements;

generating a plurality of rectangle elements; and

generating a plurality of circle elements.

9. The method of claim **6** further comprising generating a plurality of link elements, each link element including a URL of a corresponding linked item.

10. The method of claim **1** wherein the supported image format includes a color palette indexed bitmap format and the translating the plurality of images from respective formats to the supported image format comprises:

decoding each of the plurality of images into a red-green-blue bitmap format;

selecting a color in the color palette for pixels in each of the plurality of images; and

outputting a color palette indexed bitmap format for each of the plurality of images.

11. The method of claim **10** wherein the document comprises a plurality of Java classes, and wherein the executing the document according to the standard HTML language using the profile to generate a plurality of drawing instructions for displaying the document on the second device comprises:

loading and verifying the plurality of Java classes;

initializing methods associated with the plurality of Java classes; and

replacing calls to complex drawing operations with a plurality of graphics drawing elements and a plurality of text drawing elements.

12. The method of claim **1** wherein the translated document includes a plurality of text elements and a plurality of graphics drawing elements.

13. The method of claim **1** wherein the standard HTML language comprises a Java language program.

14. The method of claim **1** wherein the translating the document on the first device for use on the second device further comprises:

receiving a request at the first device over a packet switched network from the second device, the request including a URL;

retrieving the document using the URL responsive to the request; and

providing the translated document to the second device over the packet switched network.

* * * * *

Visualization of Large Terrains in Resource-Limited Computing Environments

Boris Rabinovich Craig Gotsman

Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
[borisr|gotsman]@cs.technion.ac.il

Abstract

We describe a software system supporting interactive visualization of large terrains in a resource-limited environment, i.e. a low-end client computer accessing a large terrain database server through a low-bandwidth network. By "large", we mean that the size of the terrain database is orders of magnitude larger than the computer RAM. Superior performance is achieved by manipulating both geometric and texture data at a continuum of resolutions, and, at any given moment, using the best resolution dictated by the CPU and bandwidth constraints. The geometry is maintained as a Delaunay triangulation of a dynamic subset of the terrain data points, and the texture compressed by a progressive wavelet scheme.

A careful blend of algorithmic techniques enables our system to achieve superior rendering performance on a low-end computer by optimizing the number of polygons and texture pixels sent to the graphics pipeline. It guarantees a frame rate depending *only* on the size and quality of the rendered image, independent of the viewing parameters and scene database size. An effic ent paging scheme minimizes data I/O, thus enabling the use of our system in a low-bandwidth client/server data-streaming scenario, such as on the Internet.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; D.4.4 [Operating Systems]: Communications Management—Network Communication.
Keywords: Terrain rendering, level-of-detail, interactive graphics

1 Introduction

Terrain visualization is an important component of many civilian and military applications [10, 3]. The input to the terrain visualization problem is usually a large Digital Terrain Map (DTM), consisting of elevation data sampled on a regular grid, and corresponding aerial and/or satellite texture data, which is mapped onto the reconstructed terrain surface. The output is rendered images of the terrain surface, usually as part of a "fly hrough" sequence.

The advent of the World-Wide-Web suggests the running of this type of application over the Internet, in a client/server scenario. The server is a very large remote database, accessed by the client, usually a low-end computer, over a narrow-bandwidth line (3 KByte/sec is typical for the contemporary Internet). The two bottlenecks that have to be overcome are the bandwidth in delivering relevant terrain data from the server to the client, and the CPU power required at the client for rendering this data.

The key to eff cient terrain *rendering* is eff cient online manipulation of both the geometric and texture data, especially when the scene database at the server is orders of magnitude larger that the size of client system RAM. Naive terrain rendering algorithms convert each DTM cell (bounded by four adjacent grid points) into two 3D triangles, and render (send through the graphics pipeline) all such triangles in a region determined by the viewing frustum. They also map the texture data at its highest resolution onto these polygons. This is a very ineffic ent procedure, as for low pitch angles, the number of these triangles and texture pixels (*texels*) may be extremely large. Each individual triangle projection to image space is very small, and many texels may be condensed to one image pixel, contributing negligibly to the image. One remedy to this problem, adopted in a number of works over the past few years (e.g. [8]) is to maintain the scene data at a number of discrete *levels-of-detail*. Since terrain areas at large viewing distances project to small image areas, there is no point rendering them in full detail. At any given moment during the animation, the appropriate level-of-detail is used to render the image. To do this effectively, pieces of the scene must be taken from multiple levels (foreground areas from a high-detail version, and background areas from a low-detail version), requiring methods to "stitch" together pieces of different models in a continuous fashion, so that there are no holes or breaks along the seams. This has proven to be a major problem for the geometric data, since there usually is no topological correlation between the different levels of detail. De Berg and Dobrint [1], Cohen-Or and Levanoni [5], and Lindstrom et al. [12] have provided partial solutions to the stitching problem.

In this paper we use a different approach to maintaining the terrain geometry, proposed independently by Klein and Huttner [11] and Delepine [6]. The geometry is treated in a *continuous-resolution* fashion. We do not maintain multiple geometric models (at different levels of detail), rather continuously update one model online to represent in an optimal way the projection of the terrain contained in the viewing frustum. As a result, the number of poly-

gons in the approximation is more or less constant, independent of the viewing parameters (for a f xed frame rate). For the texture,we employ a progressive wavelet compression scheme [2], which enables the extraction of texture at a continuum of resolutions from arbitrary prefi es of the encoded bit stream.

Our ultimate goal is to render any terrain image in time proportional to the *image* resolution (in pixels), and not to the scene complexity, number of DTM points in the viewing frustrum, texture resolution, etc. We are motivated by the (simple) observation that an image of fi ed resolution can contain only a bounded amount of information, therefore any algorithm rendering such an image should not use more than a bounded number of polygons and texels. Such algorithms are called *output-sensitive*. Most algorithms are not output-sensitive, and in order that they be such, require careful design. Our system contains a careful blend of techniques, some borrowed from computational geometry, which together achieve a high degree of output sensitivity, enabling adequate performance in a limited-resource environment.

Since one server may be accessed simultaneously by a large number of clients, is is crucial to minimize the amount of work the server performs per client. If this load is minimized, the server will be scalable, able to support a virtually unlimited number of clients. We adhere to this principle throughout our implementation.

Using these methods, we have developed a client application achieving terrain visualization at interactive rates on a low-end SGI (O_2)workstation, accessing a server database over a network with bandwidth comparable to the Internet. This paper describes the architecture and algorithms incorporated into our system.

2 System Overview

The large terrain scene resides on the server disk, partitioned into geometry and texture *tiles* of f xed size. A raw geometry tile contains a matrix of elevation heights, and a texture tile a matrix of texels. Tiling schemes are standard in terrain visualization applications (e.g. [4]). The server processes requests for geometry and texture data received from remote clients. In a preprocessing step at the server, applied independently to each tile (thus enabling a scene consisting of an unlimited number of tiles), the DTM points are assigned "grades" related to their importance in approximating the terrain surface. These grades are obtained from the *simplification* algorithm of Heckbert and Garland [9]. Using these grades as a third dimension, the DTM points in each tile are organized into a 3D octree, which will enable effic ent answers to future geometric queries. The client maintains online a geometry cache containing DTM points from a small subset of the server's geometry tiles. Even from these tiles, only the relevant upper levels of the corresponding octrees are imported to the client. Which levels are relevant is determined on the f y by the client.

At any given moment, a subset of the geometry cache points are maintained at the client in a dynamic Delaunay triangulation, our primary geometric data structure. To maintain the triangulation, we use the algorithms of Devillers, Meiser and Teillaud [7] for eff cient insertion and deletion into a 2D Delaunay triangulation. Delaunay triangulations are commonly considered to be suitable for terrain visualization purposes. A DTM point deserves to be in the triangulation if its grade is greater than a threshold, which is proportional to the distance of the point from the viewpoint. Section 3 elaborates on the details of how we handle the geometry.

The texture data is maintained at the server in tiles, compressed using the progressive wavelet scheme of Buccigrossi and Simoncelli [2]. This scheme compresses the data to approximately 30% of its raw size with negligble loss, and, more important, allows the decoding of the texture data from any prefi of the bit stream. Naturally, using more bits will result in a higher quality result. Client requests for texture data at a given resolution result in the streaming of the prefi of minimal length suffic ng for the required resolution. Section 4 describes our handling of the texture in more detail.

The client graphics pipeline, sometimes supported in hardware, is fed relevant triangles and texels. This pipeline takes care of the basic rendering operations, e.g. perspective projection, hidden surface elimination, and texture mapping. The main issues we address in our implementation are the minimization of data transmitted from the server to the client caches and subsequently fed to the graphics pipeline.

Typical triangulations and rendered images generated by our client system are shown in Fig. 2.

3 Geometry Processing

3.1 Data Reduction

A typical DTM is supplied on a regular grid, and this data is usually highly redundant. If the surface is to be approximated by a piecewise-linear 2D function (a collection of planar polygons), a small number of large polygons suffic to approximate the surface well in planar regions. On the other hand, terrain areas with high curvature, such as ridges and ravines, require a large number of small polygons to achieve a satisfactory approximation (see Fig. 2). By this argument, is it obvious that some DTM points are more important than others. Heckbert and Garland [9] have described a procedure which starts off with a small number of DTM points (usually the four corners of the DTM coverage), and incrementally adds points whose contribution to the surface approximation is most significan . The contribution of a point to the approximation is quantif ed by its vertical distance from the piecewise-linear approximation built with all previous points. The larger this distance - the more important the point is. The incremental procedure is done effic ently using a priority queue mechanism.

We use the Heckbert and Garland procedure at the server as a preprocessing operation on each tile to assign each DTM point a numeric "grade" - precisely the vertical distance described in the previous paragraph. This grade is stored with the point, and used later to determine online whether the point is required for the terrain approximation. This decision is based on the grade and the point's distance from the viewpoint. To facilitate eff cient decision-making, we build a 3D octree of the DTM points, the grade serving as the third dimension. The grid structure of the points in the XY plane facilitates a f xed quadtree structure in this plane, which, in turn, facilitates the organization of the data stored in the tile in a

record of fi ed length. This hierarchical spatial data structure will enable effic ent range reporting of points.

3.2 View Frustum Culling

The f rst step in frame generation is to determine which DTM tiles are relevant to the current view. In principle, if the terrain surface were planar, the intersection of the viewing frustum with the terrain surface (the view *footprint*) would be a trapezoid, whose four vertex positions could be easily computed (see Fig. 3). Since the terrain surface is not planar, the footprint terrain is bounded by a region which is the union of *two* trapezoids, formed on horizontal planes whose elevations coincide with the minimal and maximal elevations in the projection area, repectively.

The footprint is "scan-converted" by the client to determine which DTM tiles intersect it, and what resolution data (which levels of the octree) are required. This data is requested from the server. For every tile received, the octree structure of its points enables to eff ciently determine which tile points are actually contained in the footprint. Effic ency is achieved by pruning off large sets of the points corresponding to branches of the octree close to its root. The remaining points are then tested, as described in Section 3.3, to determine if they are required for the terrain approximation and rendering.

3.3 Continuous Resolution

Each DTM point has a grade quantifying its importance in the terrain approximation. This grade is traded off with distance from the viewpoint. In other words, more distant points are considered less significan . In practice, the client considers a virtual cone centered at the viewpoint, and calculates which DTM points in the geometry cache have a grade positioning them *inside* the cone (see Fig. 3). We would like to be able to determine this set of points in time proportional mainly to their number (and not to the total number of points in the viewing frustum). In computational-geometric terminology, this is called *output-sensitive range reporting*. We achieve this again using the tile octree. The complexity of the range reporting procedure is $O(\sqrt{N}+k)$, where N is the number of points in the viewing frustum, and k the number of points in the answer to the query ([13], p.79). Using this virtual cone also implies that a small change in the viewpoint induces a small change in the DTM points used for the rendering, thus ensuring the temporal continuity of the rendered images.

3.4 Caching

Portions of geometry tiles are imported from the server on demand and stored in the client cache. Only the neccesary upper levels of the tile octree are imported, possible due to the f xed structure of the octree. Hence a typical snapshot of the client cache contents would reveal a few (foreground) tiles from which almost the entire data content has been read, and many (background) tiles with a very sparse content. A prediction mechanism, based on the viewpoint trajectory, enables the loading of tiles in advance, resulting in smooth streaming of geometry from server to client.

3.5 Dynamic Delaunay Triangulation

The piecewise linear surface induced by the *Delaunay triangulation* of the 2D projection of the DTM points is generally considered the most suitable for surface approximation. This is because the minimal angle in the triangulation is maximized, eliminating long "slivery" triangles. Hence, the client constantly maintains a Delaunay triangulation of the DTM points contributing to the approximation of the terrain in the footprint. Many $O(n \log n)$ time algorithms exist for the Delaunay triangulation of n points, but not many are able to effic ently support update of the triangulation upon insertion or deletion of points. We use the algorithm of DeVillers et al [7], which inserts points in $O(\log n)$ and deletes points in $O(\log \log n)$ average time using a hierarchical data structure. Care must be taken to slightly perturb the spatial positions of the DTM points, otherwise degeneracies in the Delaunay triangulation and unstable numerics may occur.

At the client, points which were in the footprint corresponding to the previous frame, and are no longer in the current footprint, are removed from the triangulation - the main geometric data structure maintained online by the client. New points which were previously not in the footprint, and now are, are inserted into the triangulation. The turnover of points in the triangulation depends on the viewpoint velocity. Theoretically, very large velocities could cause successive frames to see totally different regions of the terrain, requiring the formation of an entirely different triangulation between frames. In practice, however, this does not occur. Typically, 99% of the footprint areas overlap between successive frames.

Pseudo-code of the f ow of control in the client while rendering a single frame appears in Fig. 1.

4 Texture Processing

The texture data must also be manipulated at multiple resolutions, since image foreground pixels contain high resolution texels, and image background pixels contain low resolution texels. The resolution of the texels contributing to any given image pixel is essentially a function of the viewing distance to that scene point. The server texture database is also organized in tiles, storing the texels compressed to approximately 30% of their original volume, using a progressive wavelet scheme. This results in a bit stream sorted by importance.

A typical low-end client computer contains a texture buffer of limited capacity (e.g. 1024x1024 pixels) with a pyramid structure on top of it. By supplying appropriate texture coordinates for the rendered triangle vertices, the graphics hardware/software maps texels from the texture buffer to the image pixels in the interior of the projected triangles. Each level of the texture pyramid contains texels representing the same terrain area, at decreasing resolutions. However, since not all texels, especially not at *all* resolutions, will contribute to the terrain image (see Fig. 4), there is no need to import them from the server. We optimize network bandwidth by loading *only* those texture tiles which intersect the view footprint, at the appropriate resolution, if they are not yet loaded. By this we mean we calculate the number of encoded bits of the texture stream required to reconstruct the texture tile at the appropriate res-

olution (the lower the required resolution, the less bits required). In any case, we use any bits available at rendering time, even though there might be less than required (if the network temporarily slows down). Which tiles are relevant can be easily determined from the geometry of the footprint. Occasionally, it is neccesary to shift the contents of the texture buffer, due to the movement of the viewpoint.

5 Experimental Results

We have implemented the procedures described in Sections 2 - 4 as a prototype client/server system, the client running on a R5000 SGI O_2, at 180MHz with 64MB RAM, based on the OpenGL API, and an X/Motif GUI. This client accesses the scene database server over a 3 KByte/sec network. The main parameters influenc ng the overall performance of the system are the size of the visualization window, i.e. the number of rendered image pixels, and the fl ght velocity. This performance is measured in the client frame rate, and the quality of the imagery delivered at that frame rate. There is an obvious tradeoff between the two, which is controlled by two independent "resolution" parameters, one for geometry, and one for texture. Increasing these parameters increases the number of triangles and/or texture bytes used for the rendering process, thus increasing the image quality, but decreasing the frame rate, due to higher rendering and bandwidth overhead. There is, however, a point beyond which the resolution parameter saturates, i.e. the marginal increase in image quality is insignif cant.

The geometric resolution parameter, namely, the average number of triangles rendered per image pixel, is controlled by the angle of the cone used for culling DTM points, as described in Section 3.3. The smaller the angle, the narrower the cone, admitting less DTM points into the Delaunay triangulation, in turn implying less triangles for the same number of image pixels (see also Fig. 3). The texture resolution is controlled by specifying the fraction of the texture tile bit stream imported and decoded to texels for the foreground image pixels. The resolution of the background image pixels is derived from this.

Keeping the resolution parameters and velocity fi ed causes the system to maintain a f xed frame rate. Increasing the velocity would slow down the system, as the turnover of points in the Delaunay triangulation and turnover of texture tiles in the texture buffer increases, incurring more CPU and bandwidth overhead. By trial and error, it seems that reasonable image quality is obtained at a geometric resolution of 0.06 triangles and 0.5 texture bytes per output image pixel. Any more than that imposes an unneccesary load on the system, slowing it down, and any less than that results in poor quality images (see Fig. 2). A telltale sign of insuff cient geometric resolution (triangles per image pixel) is if there are "jumps" (also known as "popping") in the terrain surface during animation, due to the triangles being too large and crude. A telltale sign of insuff cient texture resolution (texels per image pixel) are blurred images.

Fig. 5 shows the speed/quality tradeoffs we are able to achieve with our system at different "fl ght" velocity parameters, when one of the geometric/texture resolution parameters is f xed, and the other varied. Velocity is measured as the percentage of non-overlapping area between footprints corresponding to successive frames. The figu e shows that approximately 3 frames/sec are achievable with reasonable quality, when the image size is f xed at 300x400 pixels, and fly ng at an average (3%) velocity. Higher velocities result in a larger turnover of geometry and texture, slowing down the system frame rate. Our system accesses a scene database server covering the northern part of Israel, containing a total of 10^7 DTM points and 10^8 texels. The client uses a geometry cache of size 2MB RAM, and texture buffer of 1024x1024 texels.

6 Conclusion

In the long-term, our techniques will support client/server terrain visualization applications over the Internet. A large scene database resides at a central server site, and is accessed (perhaps simultaneously) by a number of low-end clients over the Internet for visualization purposes. This application requires tight optimization of the available network bandwidth and client rendering power.

The ever-increasing user appetite for larger and richer geometric scenes has forced computer graphics practitioners to develop output-sensitive rendering algorithms whose computational complexity is not sensitive to the complexity of the input scene, rather to the complexity of the output image. We have implemented this for the terrain visualization application by rendering at geometric and texture level-of-detail which changes continuously along the spatial and temporal dimensions. Our algorithm satisfie almost all of the f ve requirements from such an algorithm, as formulated in [12].

Use of other sophisticated data optimization techniques, such as *occlusion culling* [14], in which large portions of the geometry inside the view frustrum are effic ently culled because they are invisible, can further reduce the rendering load.

Temporal aliasing sometimes occurs in our implementation. The use of *morphing* techniques to alleviate this, such as that of Cohen-Or and Levanoni [5], are not directly applicable, again due to the dynamic nature of our Delaunay triangulation. Alternatives are being investigated.

Acknowledgements

We thank Olivier DeVillers for providing code implementing the algorithm of [7], Paul Heckbert for code implementing the algorithm of [9], and R. Buccigrossi for code implementing the algorithm of [2].

This research was supported by the Technion V.P.R. Fund - Promotion of Sponsored Research.

References

[1] M. De Berg and K. Dobrindt. On levels of detail in terrains. In *11th Annual ACM Symposium on Computational Geometry*. ACM, 1994.

[2] R.W. Buccigrossi and E.P. Simoncelli. Progressive wavelet image coding based on a conditional probability model. In *Proceedings of Int'l Conf. Acoustics Speech and Signal Processing*. IEEE, 1997.

[3] D. Cohen and C. Gotsman. Photorealistic terrain imaging and fl ght simulation. *IEEE Computer Graphics and Applications*, 14(2):10–12, March 1994.

[4] D. Cohen-Or, U. Lerner, E. Rich, and V. Shenkar. A real-time photo-realistic visual fly hrough. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):255–265, September 1996.

[5] D. Cohen-Or and Y. Levanoni. Temporal continuity of levels of detail in Delaunay triangulated terrain. In *Proceedings of Visualization '96*. IEEE Computer Society Press, 1996.

[6] T. Delepine. Online terrain level-of-detail. In *Proceedings of ITECH*, 1997.

[7] O. Devillers, S. Meiser, and M.Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Computational Geometry: Theory and Applications*, 2:55–80, 1992.

[8] L. De Floriani. A pyramidal data structure for triangle-based surface representation. *IEEE Computer Graphics and Applications*, 9(2):67–78, 1989.

[9] P. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fie ds. Technical Report CMU-CS-95-181, School of Computer Science,Carnegie Mellon University,Pittsburg ,PA , 15213, 1995.

[10] K. Kaneda, F. Kato, E. Nakamae, T. Nishita, Tanaka, and Nogushi. Three-dimensional terrain modeling and display for environmental assessment. *Computer Graphics (Proceedings of SIGGRAPH'89)*, 23(3):207–214, 1989.

[11] R. Klein and T. Huttner. Simple camera-dependent approximation of terrain surfaces for fast visualization and animation. In *Proceedings of Visualization '96 (late breaking topics)*. IEEE Computer Society Press, 1996.

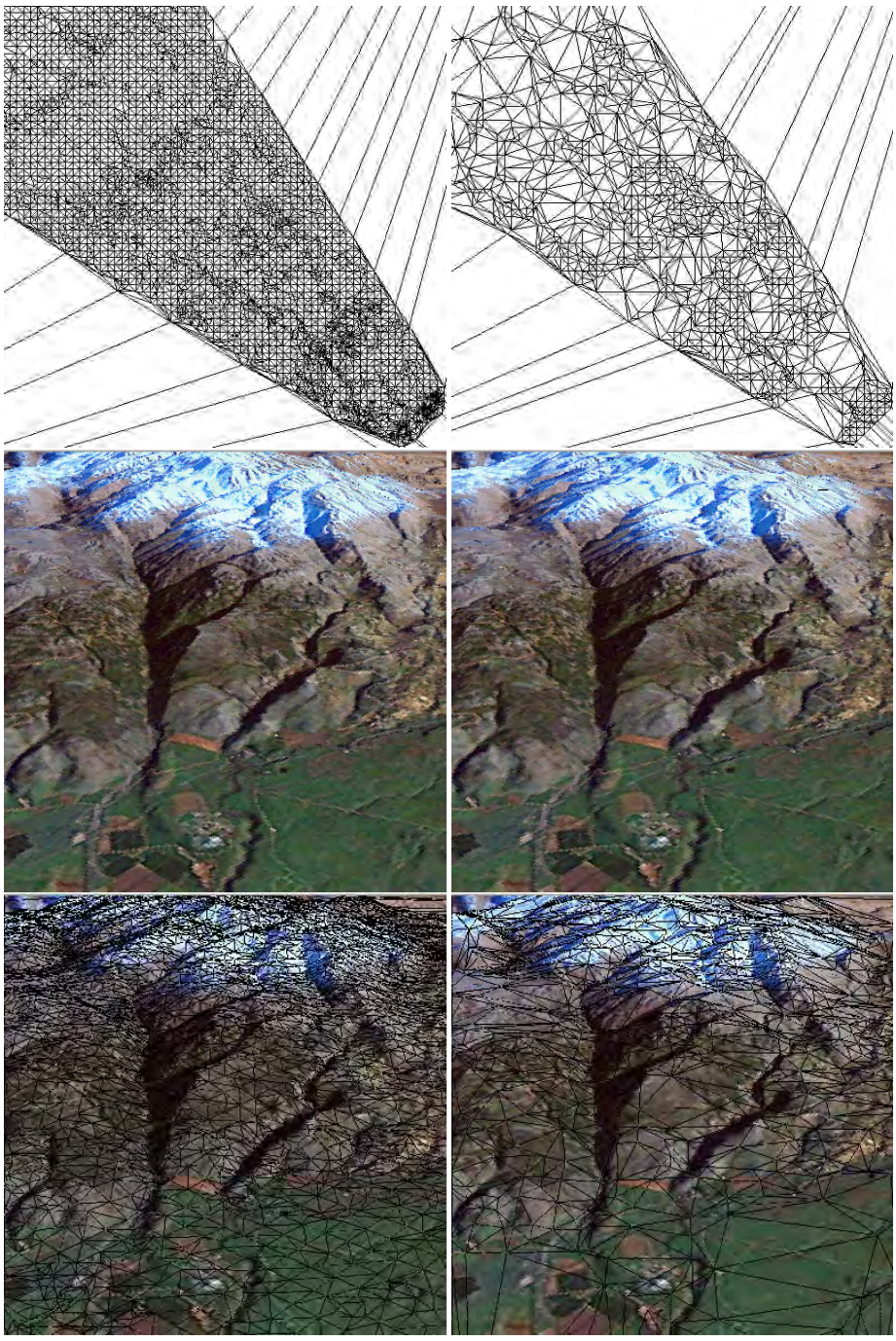
[12] P. Lindstrom, D. Koller, L.F. Hodges W. Ribarsky, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fie ds. In *Proceedings of SIGGRAPH '96*, 1996.

[13] M. Shamos and F. Preparata. *Computational Geometry*. Springer, 1989.

[14] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):249–258, 1996 (Proceedings of Eurographics, Poitiers, France, August 1996).

1. Calculate view frustum and bound terrain footprint by rectangle.
2. Scan-convert the rectangle and for each geometry tile in it:
   (a) If the tile is not in the footprint, but was in it in the previous frame, then:
      - Remove all its points from the Delaunay triangulation.
   (b) If the tile is in the footprint, but was not in the previous frame, then:
      - Request tile from server at appropriate resolution.
      - Search in tile octree for appropriate voxels.
      - Insert the points from these voxels in Delaunay triangulation.
   (c) If tile is in the footprint and was also in the previous frame, then:
      - Search in tile octree for appropriate voxels.
      - Find difference from previous frame.
      - Insert (Delete) difference points in (from) Delaunay triangulation.
3. For each texture tile in the bounding rectangle:
   (a) If the texture tile is in the footprint, but was not in the previous frame, then:
      - Calculate required resolution.
      - Request the appropriate bit stream pref x from the server.
   (b) If texture tile is in the footprint, and was also in the previous frame, then:
      - Calculate its resolution.
      - If this resolution is higher than that of the previous frame, then request more of the bit stream from the server.
4. For every tenth frame check the actual performance (frames/sec) against the required performance and adjust the geometric and/or texture resolution parameters to achieve that performance.
5. Render image.

Figure 1: Pseudo-code of the client algorithm.



(a) (b)

Figure 2: Terrain meshes (Delaunay triangulated) and views rendered at different data resolutions. (a) High resolution: 0.08 triangles/pixel and 1 texels/pixel. (b) Equivalent quality at lower resolution: 0.02 triangles and 0.8 texels/pixels. Note how more DTM points are used in foreground areas or areas of high curvature.

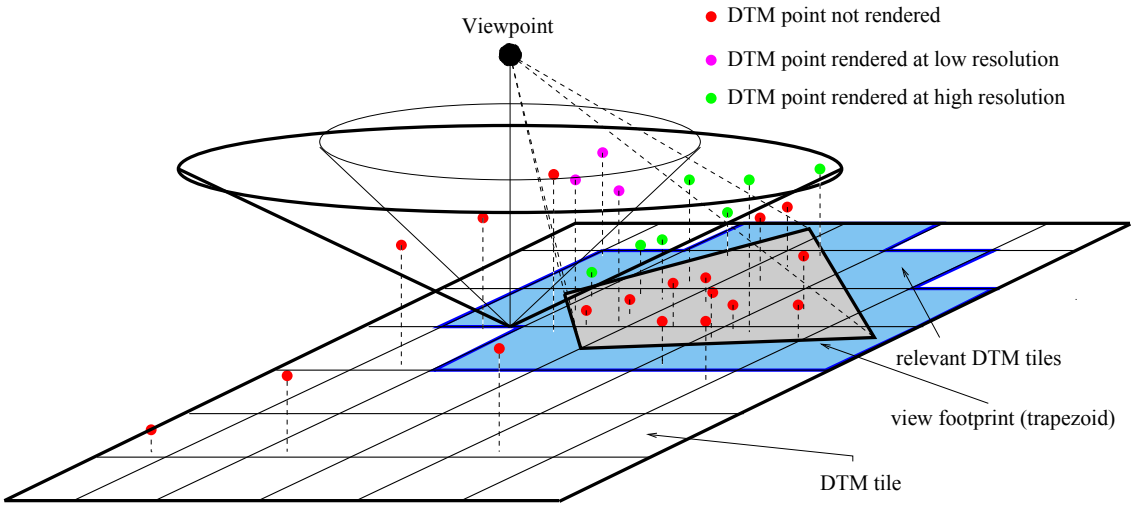


Figure 3: Determining the DTM points of the rendered Delaunay triangulation for a given view at different geometric resolutions. The narrow cone represents a low-resolution view, and the wide one a high resolution. The "elevations" of the DTM points are their precalculated grades. All points within the footprint with grade above the relevant cone are included in the triangulation. This range-reporting operation is performed effic ently using an octree structure on the points in each tile. Note that more points are admitted in the view foreground than in its background.

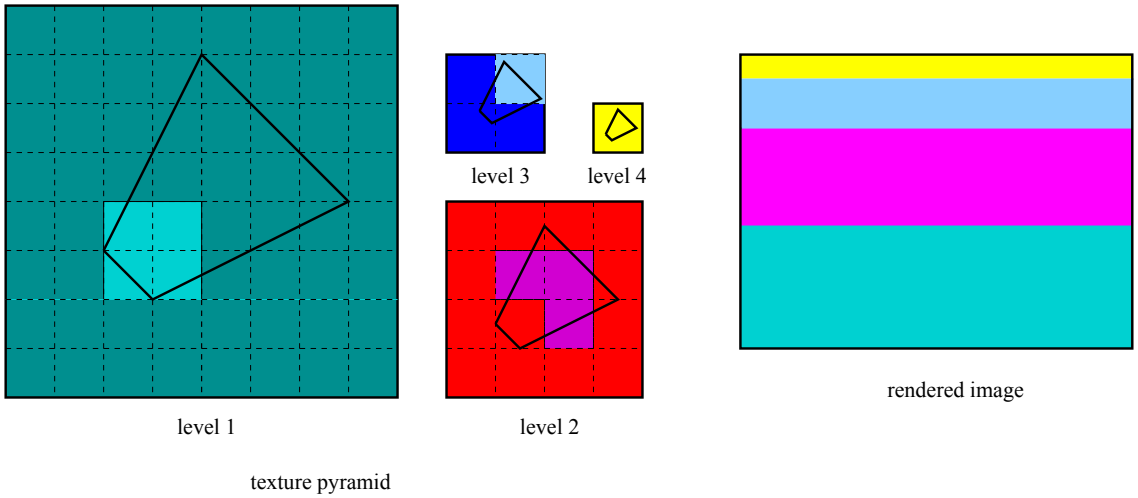
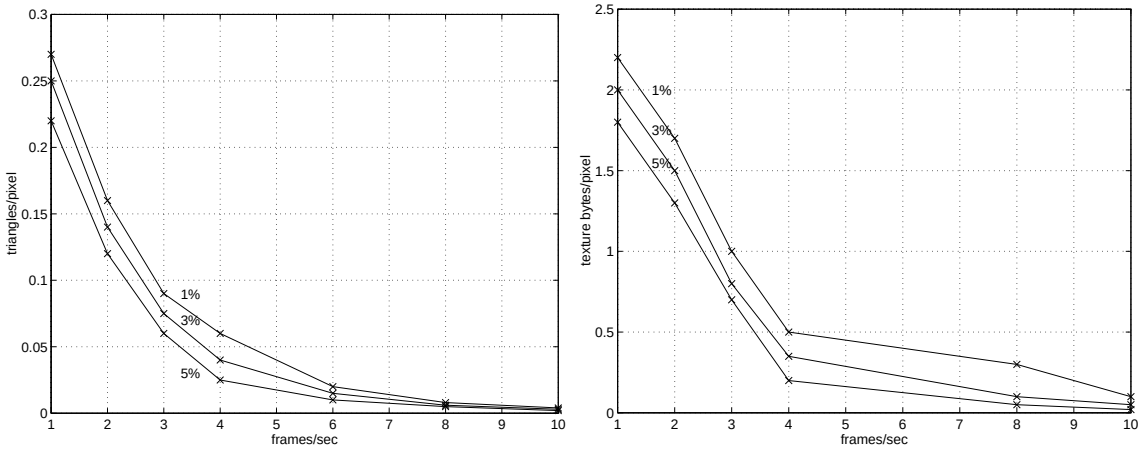


Figure 4: The contribution of individual tiles in the texture buffer to the rendered image corresponding to the marked footprint. Those tiles not contributing need not reside in the texture buffer at all, and are not streamed and decoded from the server.



(a) (b)

Figure 5: Speed/resolution tradeoff in our prototype visualization client while rendering 300x400 pixel images on a R5000 SGI O_2, accessing the scene database server over a 3 KByte/sec network. (a) Varying only geometric resolution. The texture resolution is f xed to 0.5 compressed texture bytes per pixel. (b) Varying only texture resolution. The geometric resolution is f xed to 0.06 triangles/pixel. The individual curves correspond to different fl ght velocities, which inf uence the turnover of data in system caches and bandwidth overhead.

PROCEEDINGS
Visualization '97

October 19 – 24, 1997

Phoenix, Arizona

Sponsored by
IEEE Computer Society Technical Committee on Computer Graphics

In cooperation with
ACM SIGGRAPH

The Association for Computing Machinery
1515 Broadway
New York, NY 10036

APPENDIX R

ACM ISBN: 1-58113-011-2
ACM Order Number: 428978

IEEE Computer Society Press Order Number: PR08262
IEEE Catalog Number: 97CB36155
IEEE ISBN: 0-8186-8262-0
IEEE ISBN - Library Binding: 0-8186-8263-9
IEEE ISBN - Microfiche: 0-8186-8264-7
ISSN: 1070-2385

Additional copies may be ordered from:

ACM Order Department
P.O. Box 12114
Church Street Station
New York, NY 10257 USA
Tel: +1-212-626-0500
Fax: +1-212-944-1318
E-mail: orders@acm.org

ACM European Service Center
108 Cowley Road
Oxford OX4 1JF
United Kingdom
E-mail: acm_europe@acm.org

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264 USA
Tel: +1-714-821-8380
Fax: +1-714-821-4641
E-mail: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331 USA
Tel: +1-908-981-1393
Fax: +1-908-981-9667
E-mail: mis.custserv@computer.org

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
Belgium
Tel: +32-2-770-2198
Fax: +32-2-770-8505
E-mail: euro.ofc@computer.org

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
Japan
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553
E-mail: tokyo.ofc@computer.org

Table of Contents

APPENDIX R

Preface 11
Conference Committee 13
Program Committee 14

Keynote Address: Global Tele-Immersion 15
Tom DeFanti

Capstone Address: Dissolving Descartes: Perception and the Construction of Reality 16
Mark Pesce

Papers

Session 2B: Volume Rendering I

A Comparison of Normal Estimation Schemes 19
Torsten Möller, Raghu Machiraju, Klaus Mueller, Roni Yagel
Color Plate 525

Collision Detection for Volumetric Models 27
Taosong He, Arie Kaufman
Color Plate 526

The VSBUFFER: Visibility Ordering of Unstructured Volume Primitives by Polygon Drawing 35
Rüdiger Westermann, Thomas Ertl
Color Plate 527

Volume Rendering of Abdominal Aortic Aneurysms 43
Roger C. Tam, Christopher G. Healey, Borys Flak, Peter Cahoon
Color Plate 528

Session 3A: Vector Fields

Auralization of Streamline Vorticity in Computational Fluid Dynamics Data 51
Christopher R. Volpe, Ephraim P. Glinert

Singularities in Nonuniform Tensor Fields 59
Yingmei Lavin, Yuval Levy, Lambertus Hesselink

Visualization of Higher Order Singularities in Vector Fields 67
Gerik Scheuermann, Hans Hagen, Heinz Krüger, Martin Menzel, Alyn Rockwood

Principal Stream Surfaces 75
Wenli Cai, Pheng-Ann Heng
Color Plate 529

Session 3B: Terrain Visualization

ROAMing Terrain: Real-time Optimally Adapting Meshes 81
Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, Mark B. Mineev-Weinstein

Visualization of Height Field Data with Physical Models and Texture Photomapping 89
Dru Clark, Michael J. Bailey
Color Plate 530

Visualization of Large Terrains in Resource-Limited Computing Environments 95
Boris Rabinovitch, Craig Gotsman

Building and Traversing a Surface at Variable Resolution 103
Leila De Floriani, Paola Magillo, Enrico Puppo
Color Plate 531

Session 4A: Information Visualization

Multivariate Visualization Using Metric Scaling 111
Pak Chung Wong, R. Daniel Bergeron
Color Plate 532

Visualizing the Behavior of Higher Dimensional Dynamical Systems 119
Rainer Wegenkittl, Helwig Löffelmann, Eduard Gröller
Color Plate 533

Displaying Data in Multidimensional Relevance Space with 2D Visualization Maps 127
Jackie Assa, Daniel Cohen-Or, Tova Milo
Color Plate 534

Session 4B: MultiResolution

Multiresolution Tetrahedral Framework for Visualizing Regular Volume Data 135
Yong Zhou, Baoquan Chen, Arie Kaufman
Color Plate 535

Haar Wavelets over Triangular Domains with Applications to Multiresolution Models for Flow over a Sphere 143
Gregory M. Nielson, Il-Hong Jung, Junwon Sung
Color Plate 536

Wavelet-based Multiresolutional Representation of Computational Field Simulation Datasets 151
Zhifan Zhu, Raghu Machiraju, Bryan Fry, Robert J. Moorhead
Color Plate 537

Session 5A: User Interfaces & Interaction

Dynamic Color Mapping of Bivariate Qualitative Data 159
Penny Rheingans
Color Plate 538

The Contour Spectrum 167
Chandrajit L. Bajaj, Valerio Pascucci, Daniel R. Schikore
Color Plate 539

Constrained 3D Navigation with 2D Controllers 175
Andrew J. Hanson, Eric A. Wernert
Color Plate 540

Session 5B: Volume Rendering II

Two-Phase Perspective Ray Casting for Interactive Volume Navigation 183
Martin L. Brady, Kenneth Jung, HT Nguyen, Thinh Nguyen
Color Plate 541

Accelerated Volume Rendering Using Homogenous Region Encoding 191
Jason L. Freund, Kenneth Sloan
Color Plates 542-543

An Anti-Aliasing Technique for Splatting 197
J. Edward Swan II, Klaus Mueller, Torsten Möller, Naeem Shareef, Roger A. Crawfis, Roni Yagel
Color Plate 544

A Topology Modifying Progressive Decimation Algorithm 205
William J. Schroeder
Color Plate 545

Efficient Subdivision of Finite-Element Datasets into Consistent Tetrahedra 213
Guy Albertelli, Roger A. Crawfis

Interval Volume Tetrahedrization 221
Gregory M. Nielson, Junwon Sung
Color Plate 546

Computing the Separating Surface for Segmented Data 229
Gregory M. Nielson, Richard Franke

Session 6B: Visualization Systems

Application-Controlled Demand Paging for Out-of-Core Visualization 235
Michael B. Cox, David Ellsworth
Color Plate 547

GADGET: Goal-Oriented Application Design Guidance for Modular Visualization Environments 245
Issei Fujishiro, Yuriko Takeshima, Yoshihiko Ichikawa, Kyoko Nakamura
Color Plate 548

Collaborative Visualization 253
Jason D. Wood, Helen Wright, Ken W. Brodlie
Color Plate 549

VizWiz: A Java Applet for Interactive 3D Scientific Visualization on the Web 261
Cherilyn K. Michaels, Michael J. Bailey
Color Plate 550

Session 7A: Data Extraction

Image Synthesis From A Sparse Set of Views 269
Qian Chen, Gérard G. Medioni
Color Plate 551

Virtual*ized* Reality: Constructing Time-Varying Virtual Worlds from Real World Events 277
Peter W. Rander, PJ Narayanan, Takeo Kanade
Color Plate 552

Extracting Feature Lines from 3D Unstructured Grids 285
Kwan-Liu Ma, Victoria L. Interrante
Color Plate 553

I/O Optimal Isosurface Extraction 293
Yi-Jen Chiang, Cláudio T. Silva
Color Plate 554

CAVEvis: Distributed Real-Time Visualization of Time-Varying Scalar and Vector Fields Using the CAVE Virtual Reality Theater 301
Vijendra S. Jaswal
Color Plate 555

Fast Oriented Line Integral Convolution for Vector Field Visualization via the Internet 309
Rainer Wegenkittl, Eduard Gröller

UFLIC: A Line Integral Convolution Algorithm For Visualizing Unsteady Flows 317
Han-Wei Shen, David L. Kao
Color Plate 556

The Motion Map: Efficient Computation of Steady Flow Animations 323
Bruno Jobard, Wilfrid Lefer

Session 8A: Compression

Integrated Volume Compression and Visualization 329
Tzi-cker Chiueh, Chuan-kai Yang, Taosong He, Hanspeter Pfister, Arie Kaufman
Color Plate 557

Multiresolution Compression And Reconstruction 337
Oliver G. Staadt, Markus H. Gross, Roger Weber
Color Plate 558

Optimized Geometry Compression for Real-time Rendering 347
Mike M. Chow
Color Plate 559

Session 9A: Polygonal Surfaces

Architectural Walkthroughs Using Portal Textures 355
Daniel G. Aliaga, Anselmo A. Lastra
Color Plate 560

Repairing CAD Models 363
Gill Barequet, Subodh Kumar
Color Plate 561

Dynamic Smooth Subdivision Surfaces for Data Visualization 371
Chhandomay Mandal, Hong Qin, Baba C. Vemuri
Color Plate 562

Session 10A: Surface Simplification

Smooth Hierarchical Surface Triangulations 379
Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Gregory L. Schlussmann, Isaac J. Trotts

The Multilevel Finite Element Method for Adaptive Mesh Optimization and Visualization of Volume Data 387
Roberto Grosso, Christoph Lürig, Thomas Ertl
Color Plate 563

Simplifying Polygonal Models Using Successive Mappings 395
Jonathan Cohen, Dinesh Manocha, Marc Olano
Color Plate 564

Controlled Simplification of Genus for Polygonal Models 403
Jihad El-Sana, Amitabh Varshney
Color Plate 565

Case Studies

Session 2C: Flow Visualization

Vortex Identification - Applications in Aerodynamics 413
David Kenwright, Robert Haimes
Color Plate 566

exVis 1.0: Developing a Wind Tunnel Data Visualization Tool 417
Samuel P. Uselton
Color Plate 567

Strategies for Effectively Visualizing 3D Flow with Volume LIC 421
Victoria Interrante, Chester Grosch
Color Plate 568

Towards Efficient Visualization Support for Single-block and Multi-block Datasets 425
Jean M. Favre
Color Plate 569

Session 3C: Medical Visualization

Brushing Techniques for Exploring Volume Datasets 429
Pak Chung Wong, R. Daniel Bergeron
Color Plate 570

Interactive Volume Rendering for Virtual Colonoscopy 433
Suya You, Lichan Hong, Ming Wan, Kittiboon Junyaprasert, Arie Kaufman, Shigeru Muraki, Yong Zhou, Mark Wax, Zhengrong Liang
Color Plate 571

DNA Visual And Analytic Data Mining 437
Patrick Hoffman, Georges Grinstein, Kenneth Marx, Ivo Grosse, Eugene Stanley
Color Plate 572

An Interactive Cerebral Blood Vessel Exploration System 443
Anna Puig, Dani Tost, Isabel Navazo
Color Plate 573

Session 5C: Educational Visualization

Instructional Software for Visualizing Optical Phenomena 447
David C. Banks, John T. Foley, Kiril N. Vidimce, Ming-Hoe Kiu
Color Plate 574

Wildfire Visualization 451
James Ahrens, Patrick McCormick, James Bossert, Jon Reisner, Judith Winterkamp
Color Plate 575

Visualization of Geometric Algorithms in an Electronic Classroom 455
Maria Shneerson, Ayellet Tal
Color Plate 576

Session 6C: Web & Virtual Reality

Collaborative Augmented Reality: Exploring Dynamical Systems 459
Anton Fuhrmann, Helwig Löffelmann, Dieter Schmalstieg
Color Plate 577

Visualizing Customer Segmentations Produced by Self Organizing Maps 463
Holly Rushmeier, Richard Lawrence, George Almasi
Color Plate 578

Pearls Found on the way to the Ideal Interface for Scanned-probe Microscopes 467
Russell M. Taylor II, Jun Chen, Shoji Okimoto, Noel Llopis-Artime, Vernon L. Chi, Fredrick P. Brooks Jr., Mike Falvo, Scott Paulson, Pichet Thiansanthaporn, Dave Glick, Sean Washburn, Richard Superfine
Color Plate 579

Viewing IGES Files Through VRML 471
Jed Marti

Session 7C: Engineering and Computational Geometry

Visualization of Plant Growth 475
Jeremy J. Loomis, Xiuwen Liu, Zhaohua Ding, Kikuo Fujimura, Michael L. Evans, Hideo Ishikawa
Color Plate 580

Determination of Unknown Particle Charges in a Thunder Cloud Based Upon Detected Electric Field Vectors .479
Dan Drake, Thomas Simpson, Larry Smithmeir, Penny Rheingans
Color Plate 581

Interactive Visualization of Aircraft and Power Generation Engines 483
Lisa Sobierajski Avila, William Schroeder
Color Plate 582

Efficient visualization of physical and structural properties in crash-worthiness simulations 487
Sven Kuschfeldt, Thomas Ertl, Michael Holzner
Color Plate 583

Session 9B: Math & Statistics

Visualization of Rotation Fields 491
Mark A. Livingston
Color Plate 584

Isosurface Extraction Using Particle Systems 495
Patricia Crossno, Edward Angel
Color Plate 585

A Visualization of Music 499
Sean M. Smith, Glen M. Williams

Panels

Terascale Visualization: Approaches, Pitfalls, and Issues 507
Organizers: *Carol Hunter, Roger Crawfis*
Panelists: *Michael Cox, Roger Crawfis, Bernd Hamann, Chuck Hansen, Carol Hunter, Mark Miller*

Information Exploration Shootout Project and Benchmark Data Sets:
Evaluating how Visualization does in Analyzing Real-World Data Analysis Problems 511
Organizer: *Georges Grinstein*
Panelists: *Sharon Laskowski, Bernice Rogowitz, Graham Wills*

Perceptual Measures for Effective Visualizations 515
Organizer: *Holly Rushmeier*
Panelists: *Harrison Barrett, Penny Rheingans, Sam Uselton, Andrew Watson*

Author Index 519
Cover Image Credits 521
Color Plate Section 523

APPENDIX S

User Datagram Protocol (UDP) (Windows CE 5.0)

Windows CE 5.0

Send Feedback

UDP provides a connectionless, unreliable transport service. Connectionless means that a communication session between hosts is not established before exchanging data. UDP is often used for one-to-many communications that use broadcast or multicast IP datagrams. The UDP connectionless datagram delivery service is unreliable because it does not guarantee data packet delivery and no notification is sent if a packet is not delivered. Also, UDP does not guarantee that packets are delivered in the same order in which they were sent.

Because delivery of UDP datagrams is not guaranteed, applications using UDP must supply their own mechanisms for reliability, if needed. Although UDP appears to have some limitations, it is useful in certain situations. For example, Winsock IP multicasting is implemented with UDP datagram type sockets. UDP is very efficient because of low overhead. Microsoft networking uses UDP for logon, browsing, and name resolution. UDP can also be used to carry IP multicast streams for applications such as Microsoft® Windows Media®.

See Also

Core Protocol Stack for IPv4 | User Datagram Protocol (UDP) and Name Resolution for IPv4

Send Feedback on this topic to the authors

Feedback FAQs