Content-Based Video Indexing and Retrieval

Stephen W. Smoliar and HongJiang Zhang
National University of Singapore

Current video management tools and techniques are based on pixels rather than perceived content. Thus, state-of-the-art video editing systems can easily manipulate such things as time codes and image frames, but they cannot "know," for example, what a basketball is. Our research addresses four areas of content-based video management.

Video has become an important element of multimedia computing and communication environments, with applications as varied as broadcasting, education, publishing, and military intelligence. However, video will only become an effective part of everyday computing environments when we can use it with the same facility that we currently use text. Computer literacy today entails the ability to set our ideas down spontaneously with a word processor, perhaps while examining other text documents to develop those ideas and even using editing operations to transfer some of that text into our own compositions. Similar composition using video remains far in the future, even though workstations now come equipped with built-in video cameras and microphones, not to mention ports for connecting our increasingly popular handheld video cameras.

Why is this move to communication incorporating video still beyond our grasp? The problem is that video technology has developed thus far as a technology of images. Little has been done to help us use those images effectively. Thus, we can buy a camera that "knows" all about how to focus itself properly and even how to compensate for the fact that we can rarely hold it steady without a tripod. But no camera knows "where the action is" during a basketball game or a family reunion. A camera can give us a clear shot of the ball going through the basket, but only if we find the ball for it.

The point is that we do not use images just because they are steady or clearly focused. We use them for their content. If we wish to compose with images in the same way that we compose with words, we must focus our attention on content. Video composition should not entail thinking about image "bits" (pixels), any more than text composition requires thinking about ASCII character codes. Video content objects include basketballs, athletes, and hoops. Unfortunately, state-of-the-art software for manipulating video does not "know" about such objects. At best, it "knows" about time codes, individual frames, and clips of video and sound. To compose a video document—or even just incorporate video as part of a text document—we find ourselves thinking one way (with ideas) when we are working with text and another (with pixels) when we are working with video. The pieces do not fit together effectively, and video suffers for it.

Similarly, if we wish to incorporate other text material in a document, word processing offers a powerful repertoire of techniques for finding what we want. In video, about the only technique we have is our own memory coupled with some intuition about how to use fast forward and fast reverse buttons while viewing.

The moral of all this is that the effective use of video is still beyond our grasp because the effective use of its content is still beyond our grasp. How can we remedy this situation? At the Institute of Systems Science of the National University of Singapore, the Video Classification project addresses this question. We are currently tackling problems in four areas:

- Defining an architecture that characterizes the tasks of managing video content.
- Developing software tools and techniques that identify and represent video content.
- Applying knowledge representation techniques to the development of index construction and retrieval tools.
- Developing an environment for interacting with video objects.

In this article, we discuss each of these problem areas in detail, then briefly review a recent case study concerned with content analysis of news videos. We conclude with a discussion of our plans to extend our work into the audio domain.

Architecture for video management

Our architecture is based on the assumption that video information will be maintained in a

1070-986X/94/$4.00 ©1994 IEEE

database.[1] This assumption requires us to define tools for the construction of such databases and the insertion of new material into existing databases. We can characterize these tools in terms of a sequence of specific task requirements:

- *Parsing*, which segments the video stream into generic clips. These clips are the elemental index units in the database. Ideally, the system decomposes individual images into semantic primitives. On the basis of these primitives, a video clip can be indexed with a semantic description using existing knowledge-representation techniques.

- *Indexing*, which tags video clips when the system inserts them into the database. The tag includes information based on a knowledge model that guides the classification according to the semantic primitives of the images. Indexing is thus driven by the image itself and any semantic descriptors provided by the model.

- *Retrieval and browsing*, where users can access the database through queries based on text and/or visual examples or browse it through interaction with displays of meaningful icons. Users can also browse the results of a retrieval query. It is important that both retrieval and browsing appeal to the user's visual intuition.

Figure 1 summarizes this task analysis as an architectural diagram. The heart of the system is a database management system containing the video and audio data from video source material that has been compressed wherever possible. The DBMS defines attributes and relations among these entities in terms of a frame-based approach to knowledge representation (described further under the subhead "A frame-based knowledge base," p. 65). This representation approach, in turn, drives the indexing of entities as they are added to the database. Those entities are initially extracted by the tools that support the parsing task. In the opposite direction, the database contents are made available by tools that support the processing of both specific queries and the more general needs of casual browsing.

The next three sections discuss elements of this architecture in greater detail.

Video content parsing

Three tool sets address the parsing task. The first set segments the video source material into individual camera shots, which then serve as the basic units for indexing. The second set identifies different manifestations of camera technique in these clips. The third set applies content models to the identification of context-dependent semantic primitives.

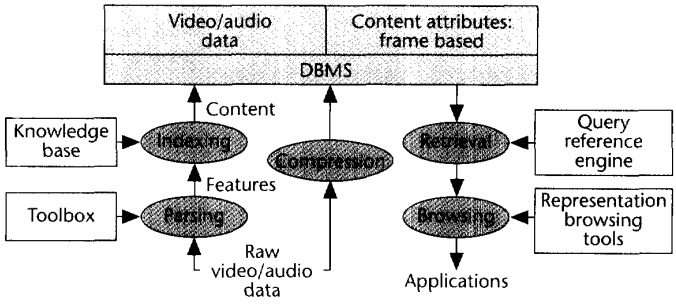

Figure 1. Diagram of video management architecture.

Locating camera shot boundaries

We decided that the most viable segmentation criteria for motion video are those that detect boundaries between camera shots. Thus, the *camera shot*—consisting of one or more frames generated and recorded contiguously and representing a continuous action in time and space—becomes the smallest unit for indexing video. The simplest shot transition is a camera cut, where the boundary lies between two successive frames. More sophisticated transition techniques include dissolves, wipes, and fade-outs—all of which take place over a sequence of frames.

In any case, camera shots can always be distinguished by significant qualitative differences. If we can express those differences by a suitable quantitative measure, then we can declare a segment boundary whenever that measure exceeds a given threshold. The key issues in locating shot boundaries, therefore, are selecting suitable difference measures and thresholds, and applying them to the comparison of video frames. We now briefly review the segmentation techniques we currently employ. (For details, see Zhang et al.[2])

The most suitable measures rely on comparisons between the pixel-intensity histograms of two frames. The principle behind this metric is that two frames with little change in the background and object content will also differ little in their overall intensity distributions. Further strengthening this approach, it is easy to define a histogram that effectively accounts for color information.[3] We also developed an automatic approach to detect the segmentation threshold on

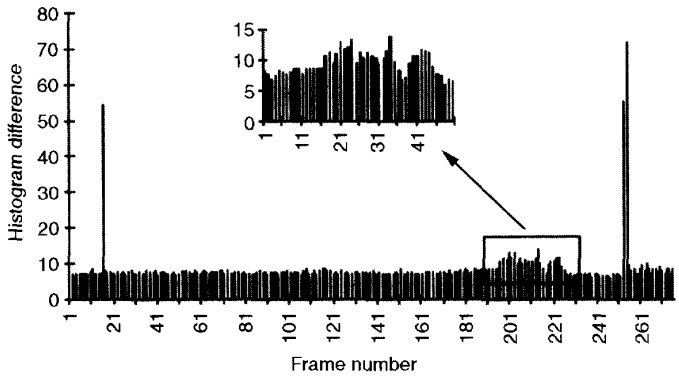

Figure 2. A sequence of frame-to-frame histogram differences obtained from a documentary video, where differences corresponding both to camera breaks and to transitions implemented by special effects can be observed.

the basis of statistics of frame difference values and a multipass technique that improves processing speed.[2]

Figure 2 illustrates a typical sequence of difference values. The graph exhibits two high pulses corresponding to two camera breaks. It also illustrates a gradual transition occurring over a sequence of frames. In this case, the task is to identify the sequence start and end points. As the inset in Figure 2 shows, the difference values during such a transition are far less than across a camera break. Thus, a single threshold lacks the power to detect gradual transitions.

A so-called *twin-comparison* approach solves this problem. The name refers to the use of two thresholds. First, a reduced threshold detects the potential starting frame of a transition sequence. Once that frame has been identified, it is compared against successive frames, thus measuring an accumulated difference instead of frame-to-frame differences. This accumulated difference must be monotonic. When it ceases to be monotonic, it is compared against a second, higher threshold. If this threshold is exceeded, we conclude that the monotonically increasing sequence of accumulated differences corresponds to a gradual transition. Experiments have shown this approach to be very effective.[2]

Shot classification

Before a system can parse content, it must first recognize and account for artifacts caused by camera movement. These movements include panning and tilting (horizontal or vertical rotation of the camera) and zooming (focal length change), in which the camera position does not change, and tracking and booming (horizontal and vertical transverse movement of the camera) and dollying (horizontal lateral movement of the camera), in which the camera position does change.[4] These operations may also occur in combinations. They are most readily detected through motion field analysis, since each operation has its own characteristic pattern of motion vectors. For example, a zoom causes most of the motion vectors to point either toward or away from a focus center, while movement of the camera itself shows up as a modal value across the entire motion field.

The motion vectors can be computed by the block-matching algorithms used in motion compensation for video compression. Thus, a system can often retrieve the vectors from files of video compressed according to standards such as MPEG and H.261. The system could also compute them in real time by using chips that perform such compression in hardware.

Content models

Content parsing is most effective with an a priori model of a video's structure.[1] Such a model can represent a strong spatial order within the individual frames of shots and/or a strong temporal order across a sequence of shots. News broadcasts usually provide simple examples of such models. For example, all shots of the anchorperson conform to a common spatial layout, and the temporal structure simply alternates between the anchorperson and more detailed footage (possibly including breaks for commercials).

Our approach to content parsing begins with identifying key features of the image data, which are then compared to domain models to identify objects inferred to be part of the domain. We then identify domain events as segments that include specific domain objects. Our initial experiments involve models for cut boundaries, typed shots, and episodes. The cut boundary model drives the segmentation process that locates camera shot boundaries. Once a shot has been isolated through segmentation, it can be compared against type models based both on features to be detected and on measures that determine acceptable similarity. Sequences of typed shots can then be similarly compared against episode models. We discuss this in more detail later, under "Case study of video content analysis."

Index construction and retrieval tools

The fundamental task of any database system is to support retrieval, so we must consider how to build indexes that facilitate such retrieval services for video. We want to base the index on semantic

properties, rather than lower level features. A knowledge model can support such semantic properties. The model for our system is a frame-based knowledge base. In the following discussion, the word "frame" refers to such a knowledge base object rather than a video image frame.

A frame-based knowledge base

An index based on semantic properties requires an organization that explicitly represents the various subject matter categories of the material being indexed. Such a representation is often realized as a semantic network, but text indexes tend to be structured as trees (as revealed by the indented representations of most book indexes). We decided that the more restricted tree form also suited our purposes.

Figure 3 gives an example of such a tree. It represents a selection of topical categories taken from a documentary video about the Faculty of Engineering at the National University of Singapore. The tree structure represents relations of specialization and generalization among these categories. Note, in particular, that categories correspond both to content material about student activities (Activity) and to classifications of different approaches to producing the video (Video_Types).

Users tend to classify material on the basis of the information they hope to extract. This particular set of categories reflects interest both in the faculty and in documentary production. Thus, the purpose of this topical organization is not to classify every object in the video definitively. Rather, it helps users who approach this material with only a general set of questions, orienting them in how to formulate more specific questions and what sorts of answers to expect.

The frame-based knowledge base is the most appropriate technology for building such a structure.[5] The *frame* is a data object that plays a role similar to that of a record in a traditional database. However, frames are grouped into classes, each of which represents some topical category. As Figure 3 illustrates, these classes tend to be organized in a specialization hierarchy. Such a hierarchy allows the representation of content in terms of one or more systems of categories that can then be used to focus attention for a variety of tasks.

The simplest of these tasks is the casual browsing of collections of items. However, hierarchical organization also facilitates the retrieval of specific items that satisfy the sorts of constraints normally associated with a database query. Like the records of a database, frames are structured as a collection of fields (usually called *slots* in frame-based systems). These slots provide different elements of descriptive information, and the elements distinguish the topical characteristics for each object represented by a frame.

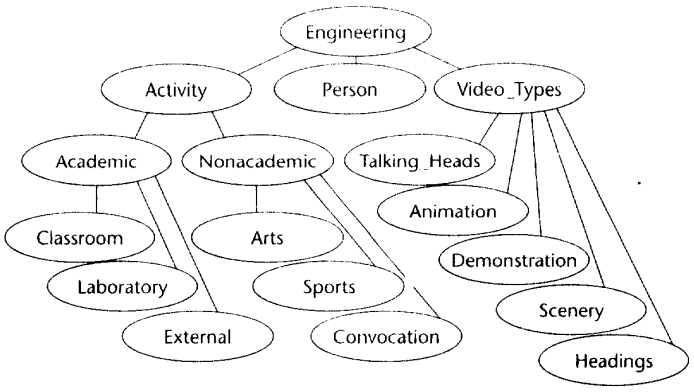

Figure 3. A tree structure of topical categories for a documentary video about engineering at the National University of Singapore.

It is important to recognize that we use frames to represent both classes (the categories) and instances (the elements categorized). As an example of a class frame, consider the Laboratory category in Figure 3. We might define the frame for it as shown in Figure 4a. Alternatively, we can define an instance of one of its subclasses in a slightly similar manner as shown in Figure 4b.

Note that not all slots need to be filled in a class definition ("void" indicates an unfilled slot), while

```
Name:  Laboratory
SuperClass:  Academic
SubClasses:  #table[Computer_Lab
     Electronic_Lab Mechanical_Lab
     Civil_Lab Chemical_Lab]
Instances:  void
Description:  void
Video:  void
Course:  void
Equipment:  void

Name:  Wave_Simulator
Class:  Civil_Lab
Description:  "Monitoring pressure
     variation in breaking waves."
Video:  WaveBreaker_CoverFrame
Course:  Civil_Eng
Equipment:#table[Computer
     Wave_Generator]
```

Figure 4. Examples of class frame `Laboratory` *(top) and subclass instance* `Wave_Simulator` *(bottom).*

they do all tend to be filled in instances. Also note that a slot can be filled by either a single value or a collection of values (indicated by the "#table [...]" construct).

For purposes of search, it is also important to note that some slots, such as Name, SuperClass, SubClasses, Instances, and Class, exist strictly for purposes of maintaining a system of frames. The remaining slots, such as Description, Video, Course, and Equipment, are responsible for the actual representation of content. These latter slots are thus the objective of all search tasks.

What is important for retrieval purposes is that we can translate knowledge of a slot's type into knowledge of how to search it.

Most frame-based knowledge bases impose no restrictions on the contents of slots: Any slot can assume any value or set of values. However, the search objective can be facilitated by strongly typing all slots. The system could enforce such a constraint through an "if-added" demon that does not allow a value to be added to a slot unless it satisfies some data typing requirement. For example, if Shot is a class whose instances represent individual camera shots from a video source, then only values that are instances of the Shot class can be added to the Video slot in frames such as those in Figure 4.

Data typing can determine whether or not any potential slot value is a frame, and it might even be able to distinguish class frames from instance frames. However, we can make typing even more powerful if we extend it to deal with classes as if they were data types. In this case, type checking would verify not only that every potential Video slot value is an instance frame but, more specifically, that it is an instance of the Shot class. Furthermore, we could subject slot values for instances of more specific classes to even more restrictive constraints. Thus, we might constrain the Video slot of the Headings frame to check whether or not the content of a representative frame of the Shot instance being assigned consists only of characters. (We could further refine this test if we knew the fonts used to compose such headings.)

What is important for retrieval purposes is that we can translate knowledge of a slot's type into knowledge of how to search it. We can apply different techniques to inspecting the contents of different slots, and we can combine those techniques by means far more sophisticated than the sorts of combinations normally associated with database query operations.

Retrieval tools

Let us now consider more specifically how we can search frames given a priori knowledge of the typing of their slots. Because a database is only as good as the retrieval facilities it supports, it must have a variety of tools, based on both text and visual interfaces. Our system's current suite of tools includes a free-text query engine and interface, the tree display of the class hierarchy, image feature-based retrieval tools, and the Clipmap.

Every frame in the knowledge base includes a Description slot with a text string as its contents. Thus, the user can provide text descriptions for all video shots in the database. The free-text retrieval tool retrieves video shots on the basis of the Description slot contents. A concept-based retrieval engine analyzes the user's query.[6] Given a free-text query specified by the user, the system first extracts the relevant terms by removing the nonfunctional words and converting those remaining into stemmed forms. The system then checks the query against a domain-specific thesaurus, after which it uses similarity measures to compare the text descriptions with the query terms. Frames whose similarity measure exceeds a given threshold are identified and retrieved, linearly ordered by the strength of the similarity.

In addition to using free text, we can formulate queries directly on the basis of the category tree itself. This tree is particularly useful in identifying all shots that are instances of a common category at any level of generalization. We can then use the tree to browse instances of related categories. The class hierarchy also allows for slot-based retrieval. Free-text retrieval provides access to Description slots, but we can search on the basis of other slots as well. For example, we can retrieve slots whose contents are other frames through queries based on their situation in the class hierarchy. We can compare slots having numeric values against numeric intervals. Furthermore, if we want to restrict a search to the instances of a particular class, then the class hierarchy can tell us which slots can be searched and what types of data they contain.

Retrieval based on the contents of Video slots will require computation of characteristic visual features. As an example, a user examining a video of a dance performance should be able to retrieve all shots of a particular dancer on the basis of costume color. Retrieval would then require con-

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.