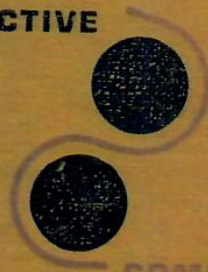


**PRINCIPLES OF INTERACTIVE
COMPUTER GRAPHICS**



**COMPUTER
SCIENCE
SERIES**

Library of Congress Cataloging in Publication Data

Newman, William M 1939-
Principles of interactive computer graphics.

(McGraw-Hill computer science series) (McGraw-Hill series in artificial intelligence)

Bibliography : p.

Includes index.

1. Computer graphics. 2. Interactive computer systems. I. Sproull, Robert F., joint author.

II. Title.

T385.N48 1979
ISBN 0-07-046338-7

001.55

78-23825

PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS

Copyright © 1979, 1973 by McGraw-Hill, Inc. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher.

5 6 7 8 9 0 D O D O 8 3 2 1 0

The editors were Charles E. Stewart and Frances A. Neal; the production supervisor was Dominick Petrellese. R.R. Donnelley & Sons Company was printer and binder.

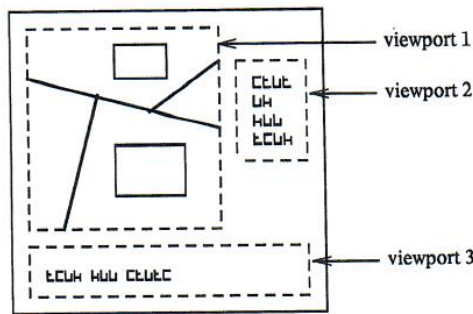


Figure 5-13 Use of viewports for map display (1), command menu (2), and user messages (3).

5-6 THE WINDOWING TRANSFORMATION

The windowing transformation is so named because it involves specifying a *window* in the world-coordinate space surrounding the information we wish displayed. This is by no means the only way such a transformation can be specified; for example, we can define the scale factor and translation to be applied to the picture, or in place of the translation we can define a world-coordinate point we wish transformed to a certain spot on the screen, say the screen center. Each of these methods may prove convenient in certain circumstances; the windowing transformation has the advantage of letting us specify directly the rectangle of interest in the world-coordinate picture definition.

In addition to the window, we can also define a *viewport*, a rectangle on the screen where we would like the window's contents displayed. In doing so, we exploit the ability of our clipping algorithm to clip to any right rectangular boundary. It is often useful to specify a viewport smaller than the screen, for we can then leave room for command menus, system messages, and so forth. Each such part of the picture may be displayed in a separate viewport, as shown in Figure 5-13.

Figure 5-14 illustrates the full windowing transformation from world to screen. We use the window to define *what* we want to display; we use the viewport to specify *where* on the screen to put it. Thus we can scan over a large picture by keeping the window size constant and varying its position; changing the window size alters the picture magnification. Normally we shall take care to keep the window and viewport similar in shape. In Figure 5-14 however, we see the distortion that can be achieved by making the shapes different.

The actual transformation we apply to each point is very simple. Let us suppose that the edges of the window are at $x = W_{xl}$, $x = W_{xr}$, $y = W_{yb}$, and $y = W_{yt}$ (see Figure 5-14), all measured in *world coordinates*, and that the corresponding edges of the viewport are at $x = V_{xl}$, $x = V_{xr}$, $y = V_{yb}$, and $y = V_{yt}$, all measured in *screen coordinates*. Then the point (x_w, y_w) in world coordinates transforms into the point (x_s, y_s) in screen coordinates, as follows:

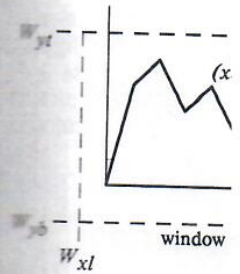


Figure 5-14 The windowing transformation.

$$x_s = \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}} x_w + \frac{V_{xl} V_{xr} - V_{yb} V_{yt}}{W_{xr} - W_{xl}}$$

$$y_s = \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}} y_w + \frac{V_{yb} V_{yt} - V_{xl} V_{xr}}{W_{yt} - W_{yb}}$$

These expressions (x_w, y_w) within the window's left-hand corner and in the viewport. Adding the position of (x_s, y_s) Equations 5-2 reduce

$$x_s = a$$

We therefore arrange that the window and viewport are defined by involving only two multiplications. The complete windowing transformation is achieved by transforming the entire picture against the viewport. The lines in a picture defined with shared endpoints. A more dramatic gain is gained by transforming right rectangles, and since

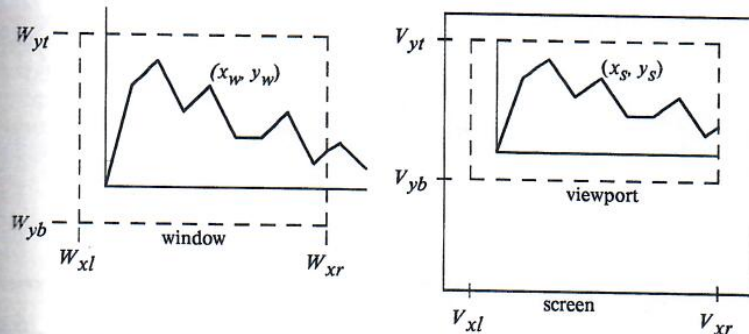


Figure 5-14 The windowing transformation.

$$x_s = \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}}(x_w - W_{xl}) + V_{xl}$$

$$y_s = \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}}(y_w - W_{yb}) + V_{yb} \quad (5-2)$$

These expressions are derived by determining the position of the point (x_w, y_w) within the window as a fraction of full displacement from the bottom left-hand corner and interpreting this as a fraction of full displacement across the viewport. Adding the offset of the viewport's bottom left-hand corner gives the position of (x_s, y_s) on the screen.

Equations 5-2 reduce to the form

$$x_s = ax_w + b \quad y_s = cy_w + d \quad (5-3)$$

We therefore arrange to compute the values of a , b , c , and d when the window and viewport are defined, so that we can transform each point by a computation involving only two multiplications and two additions.

The complete windowing transformation can then be applied to a picture by transforming the endpoints of each line, using Equations 5-3, and clipping the line against the viewport boundary. In the interests of transforming the picture quickly we naturally wish to speed up the computation, and we can do so in a couple of ways. The first of these is based on the observation that the lines in a picture definition are often connected as a sequence of line segments with shared endpoints. We can avoid transforming each shared endpoint twice by comparing the world coordinates of each point and its predecessor.

A more dramatic improvement in the speed of transformation can be gained by transforming only visible lines. Since both window and viewport are right rectangles, and since both circumscribe the displayed information, we may

Figure 5-13 Use of viewports for map display (1), command menu (2), and error messages (3).

use it involves specifying a region of the information we wish to display. Such a transformation can be achieved by a factor and translation to be applied. In this way we can define a world-coordinate spot on the screen, say the bottom-left corner. It is more convenient in certain cases to use the advantage of letting us use the same world-coordinate picture.

A *viewport*, a rectangle on the screen, is displayed. In doing so, we clip to any right rectangular region smaller than the screen, for example, for error messages, and so forth. A separate viewport, as shown

transformation from world to screen coordinates. We want to display; we use the world coordinates. Thus we can scan over a large region of the picture by varying its position; changing the position. Formally we shall take care to transform the picture. In Figure 5-14 however, we use two different shapes.

The point is very simple. Let us use (W_{xl}, W_{yb}) for the window's bottom-left corner, (W_{xr}, W_{yt}) for the window's top-right corner, and (V_{xl}, V_{yb}) for the viewport's bottom-left corner, (V_{xr}, V_{yt}) for the viewport's top-right corner, and (x_w, y_w) for the point in world coordinates, as follows:

use either of them as a clipping region. The more efficient implementation of the windowing transformation therefore clips *first*, using the window as clipping region, and then performs the transformation of Equations 5-3 on those lines which are at least partially visible. When a small portion of a very large picture is being viewed, the advantage of performing clipping before transformation is considerable.

EXERCISES

5-1 Program the Sutherland-Hodgman polygon clipping algorithm in the language of your choice, using recursion if possible.

5-2 The Sutherland-Hodgman algorithm can be used to clip lines against a nonrectangular boundary. What uses might this have? What modifications to the algorithm would be necessary? What restrictions would apply to the shape of the clipping region?

5-3 Some displays possess hardware for displaying circular arcs. Design an algorithm to clip circles and generate arcs suitable for passing to such a display.

5-4 Extend the clipping program given on page 66 to perform the complete viewing transformation of a line from world coordinates to screen coordinates.

5-5 Application programs often use floating-point numbers to define pictures, whereas the display uses integers. Should the conversion from floating-point to integer format be done before or after clipping? How should numbers be rounded? Does the decision whether to clip to the window or viewport depend on the relative speeds of integer and floating-point arithmetic?

5-6 Under what circumstances would midpoint clipping be preferable to the use of the program on page 66?

5-7 What additional logic is needed in the clipping algorithm to keep track of linked visible line segments?

5-8 The Cohen-Sutherland clipping algorithm is optimized in favor of clipping pictures much larger than the window. What features would you look for in an algorithm to clip pictures only slightly larger than the window? Devise such an algorithm.

5-9 Hand-simulate the polygon clipping algorithm to verify that it produces the same result as shown in Figure 5-9. Copy the figure of the arrow to graph paper and step through the flow chart of Figure 5-11.

