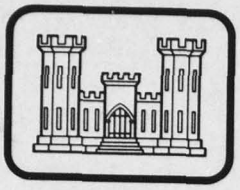
AD-A070 907 GEORGE WASHINGTON UNIV WASHINGTON DC DEPT OF ELECTRI--ETC F/G 9/2
RASTER GRAPHICS EXTENSIONS TO THE GRAPHICS COMPATIBILITY SYSTEM--ETC(U)
MAR 79 J D FOLEY, J TEMPLETON, D DASTYLAR DACA39-78-M-0073
UNCLASSIFIED WES-MP-0-79-3 NL

| OF |
AD
A070907

END
DATE
FILMED
8-79
DDC

(12) LEVEL II



MISCELLANEOUS PAPER O-79-3

AD A070907

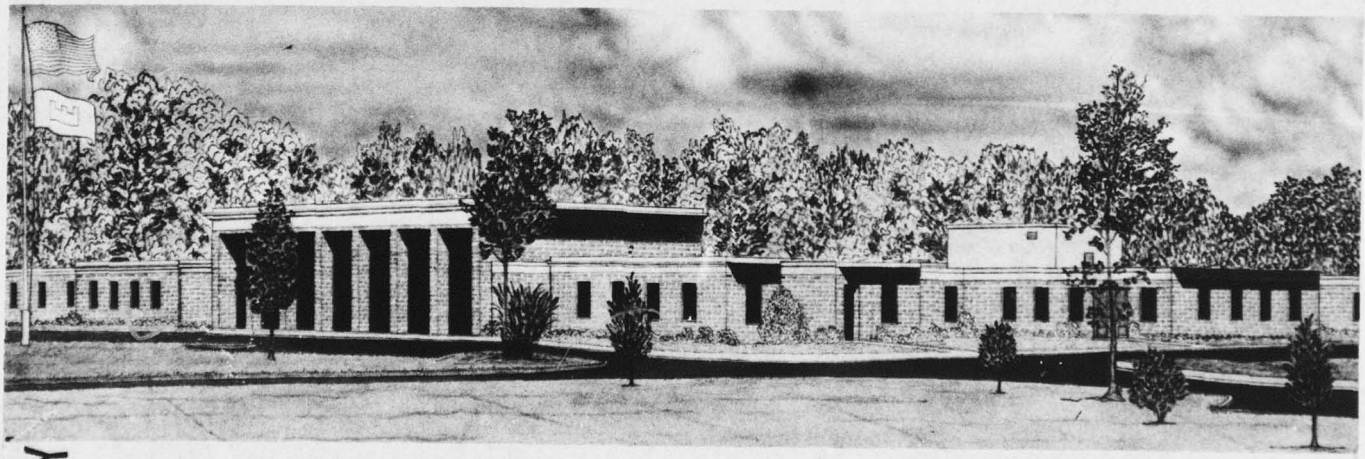
RASTER GRAPHICS EXTENSIONS TO THE GRAPHICS COMPATIBILITY SYSTEM (GCS)

by

James D. Foley, James Templeton, Dara Dastylar

Department of Electrical Engineering and Computer Science
George Washington University
Washington, D. C. 20052

March 1979
Final Report



Prepared for Office, Chief of Engineers, U. S. Army
Washington, D. C. 20314

Under Contract No. DACA39-78-M-0073

Monitored by Automatic Data Processing Center
U. S. Army Engineer Waterways Experiment Station
P. O. Box 631, Vicksburg, Miss. 39180

Destroy this report when no longer needed. Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

(18) WES

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (19) MP- Miscellaneous Paper O-79-3	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* (6) RASTER GRAPHICS EXTENSIONS TO THE GRAPHICS COMPATIBILITY SYSTEM (GCS),	5. TYPE OF REPORT & PERIOD COVERED (9) Final report,
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) James D. Foley, James Templeton, Dara Dastylar	8. CONTRACT OR GRANT NUMBER(s) (15) Contract No. DACA39-78-M-0073
9. PERFORMING ORGANIZATION NAME AND ADDRESS George Washington University, Department of Electrical Engineering and Computer Science, Washington, D. C. 20052	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Integrated Software Research & Development Program, AT11
11. CONTROLLING OFFICE NAME AND ADDRESS (11) Office, Chief of Engineers, U. S. Army Washington, D. C. 20314	12. REPORT DATE March 1979
	13. NUMBER OF PAGES 76
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* U. S. Army Engineer Waterways Experiment Station, Automatic Data Processing Center, P. O. Box 631, Vicksburg, Miss. 39180	15. SECURITY CLASS. *(of this report)* Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

(12) 81 p.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

DDC RECEIVED JUL 9 1979 B

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computer graphics
Graphic arts
Graphics Compatibility System
Raster graphics

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

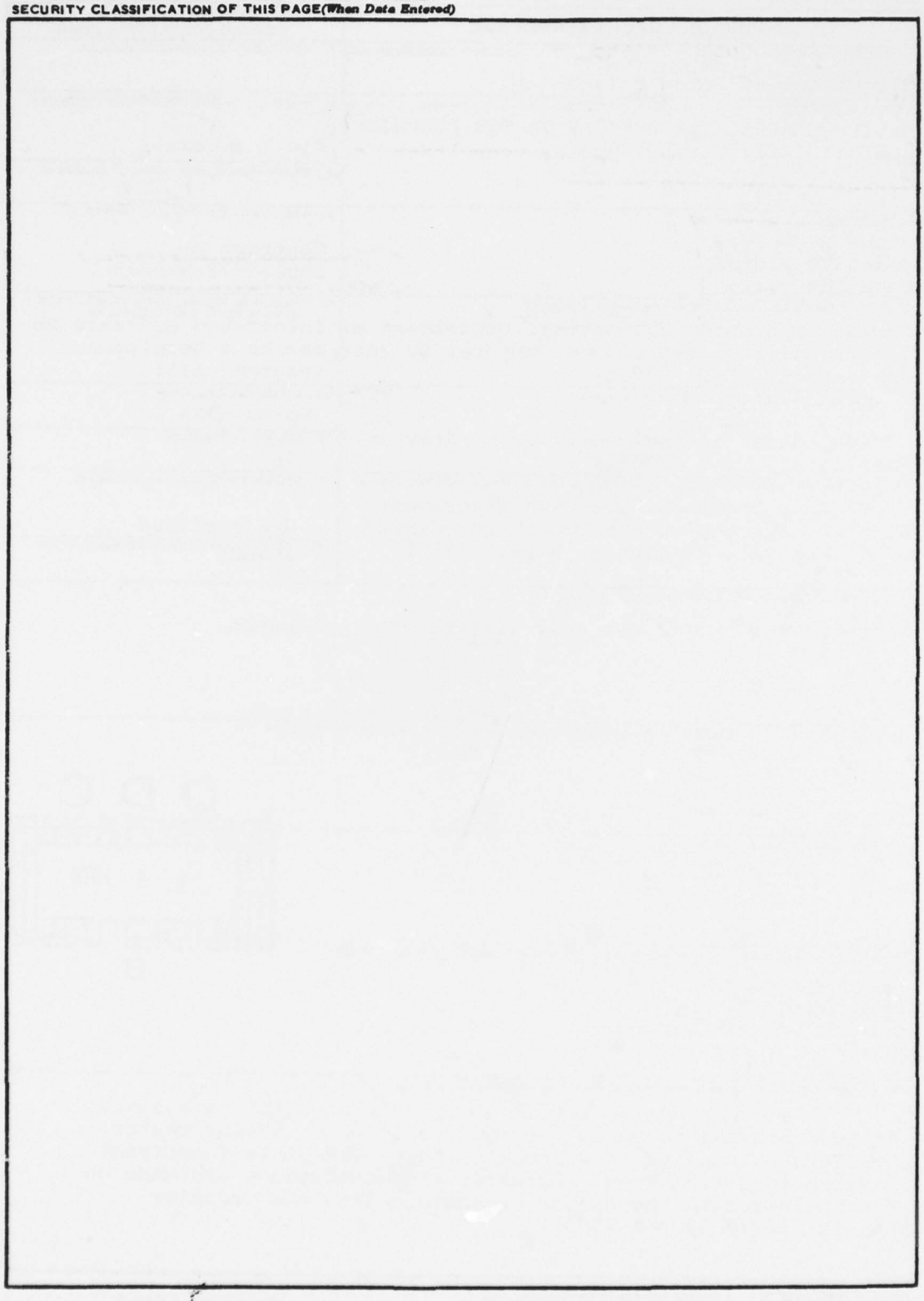
This report discusses a set of architectural extensions to line drawing graphics packages to support various raster displays. A series of subroutine calls and their functional descriptions are given. Specific recommendations are made on how to integrate the raster extensions into the Graphics Compatibility System (GCS).

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

411 264



PREFACE

This report is the result of work performed under Contract No. DACA39-78-M-0073, dated 27 February 1978, between the U. S. Army Engineer Waterways Experiment Station (WES), Vicksburg, Miss., and Dr. James D. Foley, Graphics Consultant, for development of raster graphics extensions to the Graphics Compatibility System (GCS). The authors are Dr. James D. Foley, Mr. James Templeton, and Mr. Dara Dastylar. The task was directed by the Automatic Data Processing (ADP) Center, WES, as part of the Integrated Software Research and Development Program, AT11, Engineering Software Research, Graphics Interfaces for Scientific Applications, Improvement and Refinement of Computer Graphics Software Package, sponsored by the Office, Chief of Engineers, U. S. Army.

Mr. James M. Jones II, R&D Software Group, ADP Center, WES, monitored the contract under the general supervision of Dr. N. Radhakrishnan, Special Technical Assistant to the Chief of the ADP Center, and Mr. D. L. Neumann, Chief of the ADP Center.

Director of WES during the period of the contract was COL John L. Cannon, CE. Mr. F. R. Brown was Technical Director.

CONTENTS

RASTER GRAPHICS EXTENSIONS TO THE GRAPHICS COMPATIBILITY SYSTEM (GCS)

PART I: HOW TO INTEGRATE RASTER GRAPHICS CONCEPTS INTO GCS

1. Introduction

The 3D version of the Graphics Compatibility System (GCS-3D) contains several basic concepts that allow GCS to be expanded to deal with raster graphics. The most important capability that a raster graphics system has, that a line drawing system does not, is the ability to display a polygon as a filled in area of color. This leads to a new problem, namely: when two polygons overlap on the screen one of them must obscure the other. Therefore, a method such as a hidden surface algorithm is required to display pictures in a consistent, controllable way. The 3D data structure already in-use by GCS-3D makes this extension possible without a major revision of the system.

To access some capabilities of raster systems, either new or expanded concepts must be added to a line oriented graphics package. Some examples are: images made of colored pixels, shaded polygons, direct view surface pixel operations, etc. These can only be implemented by new sets of subroutines and options.

The basic set of concepts present in GCS-3D can also be found in the original CORE Graphics System. Virtually all the additions required to extend one system for raster graphics can be directly applied to the other. Two extensive documents, already written by our group, will serve as detailed references:

Raster Graphics Extensions to the Core Graphics System

Reference Manual: Raster Graphics Extensions to the Core System.

These documents are included as part of this report.

2. Hidden Surfaces

As noted earlier, to determine which of two overlapping polygons obscures the other, one must consider depth information about each polygon. Within a single window this information is just the distance from the observer to the polygon in virtual world coordinates. However, when several different windows are displayed in different viewport areas on the screen, a more sophisticated way of relating depth information about polygons in the windows is required. Three dimensional device viewports are quite convenient for resolving this problem. Each user 3D window is mapped into a 3D device viewport, so that a particular polygon in the virtual world occupies a specific location in 3D device space. The location in device space is then used to define the depth of the polygon when the hidden surface algorithm is applied.

Using this approach there are several ways of controlling which of two polygons obscures the other one. A simple depth priority system can be constructed by keeping polygons in planes normal to the Z axis. Then all points on an object are at the same depth from the observer. This is called 2½D. It simplifies a hidden surface algorithm into a simple depth sort. One way to achieve this priority between two windows is to place each into a flat (in the Z dimension) plane with one plane in front of the other. An even simpler method of making one polygon overwrite another is to place both polygons on the same depth plane. Then the only way to determine which is the obscuring polygon is with temporal priority (the second overwrites the first).

GCS-3D already has the concepts of 3D device space and 3D device viewports (see the GCS subroutine U3AREA) and a 3D data structure to support these notions for hidden surface algorithms and 2½D depth priority. Therefore no additional concepts or functions need be added to GCS-3D to deal with hidden surfaces.

3. Polygons

A fundamental concept for raster graphics is the polygon that is filled in with color.* A polygon is specified by two types of information. First, the way in which the polygon will be colored is described, then the location of the polygon's vertices in virtual world coordinates (in either 2D or 3D) must be given.

In the raster extension to the CORE system there are three different types of colored polygon, namely: plain, painted, and shaded. The plain type is simply a polygon filled with a constant color or gray scale intensity over its entire surface. The color/intensity of the polygon is the current color/intensity attribute in effect when the polygon primitive is defined.

A painted polygon is filled with an image made up of differently colored pixels. The polygon's image is defined by the user as a large rectangular array containing color/intensity values. The user then specifies the exact portion of the image's pixel array to be placed in the polygon (see the CORE reference manual for the specifics of the process). Once a painted polygon is described it can be viewed from any location in virtual world space and the image on its surface will appear properly transformed. This is analogous to viewing a picture on the wall from different angles and having it still look correct.

A shaded polygon may have a different color specified at each of its vertices. It is then filled by linearly interpolating

* This is not the outline of a regular polygon, drawn by the GCS subroutine UPLYGN. Such polygons could be built on top of the basic polygon primitive described in this section.

between the colors specified at each vertex (GOURAUD shading). This is quite useful for depicting continuously varying colors or shaded objects.

In keeping with the general method used by GCS-3D for passing arguments in subroutine calls, the three polygon types are new options that belong to the USET subroutines. The set of subroutines added to the CORE system to complete the description of these various polygons must also be added to GCS-3D. The plain polygon needs a comprehensive way of specifying its color (see section 6).

A painted polygon requires the following: a pixel array or file to store its image in. The CORE uses the functions:

```
SET_PIXEL_ARRAY(COLUMNS,ROW,ARRAY)
SET_PIXEL_FILE(COLUMNS,ROWS,FILE_NAME)
```

The format of this file or array must be given, see:

```
SET_PIXEL_ARRAY_FORMAT(BITS_PER_PIXEL,
          BITS_PER_PACKING_UNIT,PIXELS_PER_UNIT)
```

And a way to specify the region of the image that should be used to fill the polygon, see:

```
SET_PIXEL_ARRAY_EXTENT(X_LENGTH,Y_LENGTH)
SET_VERTEX_MAP(VERTEX_INDEX_ARRAY,X_MAP,Y_MAP)
```

A shaded polygon needs a way to define the color/intensity at each vertex: see the SET_VERTEX_COLORS functions.

Finally, the description of a polygon (of any type) is completed by giving its vertex list in either 2D or 3D virtual world coordinates: see the POLYGON fucntions in the CORE raster extensions.

4. Polygon Meshes

Often instead of dealing with a single polygon at a time it is more convenient and efficient to deal with a set of connected polygons called a mesh. Since several polygons in a mesh may share the same vertex, it is convenient to give the entire vertex list of the mesh at one time. Then a polygon element of the mesh is defined by ind xing a subset of these vertices. This saves storage space and gives information about which polygons share common vertices or edges.

To further improve the way polygon meshes are handled, three different types of meshes can be specified: a mesh that does not form a closed polyhydra, a closed convex polyhydra, and a closed concave polyhydra. These are referred to as the shape of the mesh in the CORE system.

If GCS-3D is going to deal with polygon meshes it needs a subroutine call like the POLYGON_MESH functions in the CORE System, to pass the entire vertex list and define the beginning of a mesh. It also should have a subroutine that designates a particular polygonal element in the mesh, such as:

POLYGON_EDGE_LIST(VERTEX_INDEX_ARRAY,N)

The shape attribute of a mesh could easily be included as a new group of USET options.

5. Colors and Intensities

Because of the need to precisely control the color and intensity in raster graphics, most raster systems have a greater range of colors and intensities than do vector systems. Therefore, the ways to specify colors and intensities need to be expanded in GCS.

In the Core System the color attribute (a parallel set of capabilities for the intensity attribute also exists) is specified in two ways: by value and by index. In the first way, the color attribute is specified by Red, Green, and Blue primaries, if the exact (R,G,B) triplet specified is not in the active color set (set of colors the device is capable of displaying simultaneously) values are rounded to the nearest color displayable on the view surface. In the second way (by index), we can sepcify that the I^{th} color/intensity from the ordered set of available colors/intensities, is to be used as the color/intensity attribute.

The concept can be extended to a color lookup table. A lookup table is the set of colors that the display surface is capable of displaying simultaneously, and is a subset of the global color set (all colors that can be displayed on the display surface).

To keep the consistency of passing arguments for sub-routine calls, as used in GCS-3D, some new options need to be introduced.

Setting a color (intensity) by index should be an UPSET option. In order to allow a means of expansion beyond the

8 colors (3 levels of intensity) now available as a USET option, additional USET options should be made available. To set a color or background color by value (red, green, and blue primaries) GCS-3D needs some new UPSET options, or a new set of subroutines to set three new options, one for each primary colors.

Setting the vertex colors or intensities is basically the same as setting the color or intensity, but with a difference of having a list of colors or intensities to be set (for vertices of a shaded polygon). This also can be done by introducing a new set of subroutines to set the UPSET options.

6. Operations on the View Surface

In order to have full control over what is displayed on a raster screen, the application programmer should have direct access to the view surface/refresh buffer (some interaction techniques must directly modify and restore portions of the refresh buffer). This includes reading and writing pixels onto the screen. This can be done either one pixel at a time, or a rectangular portion of the screen could be dealt with as a group of pixels. Also, the user should be able to specify the different ways in which new information can be overlayed on the screen (setting the view surface overlay mode).

The operations on the view surface (see Raster Extensions to the Core System, Section 10) introduces new subroutines for read/write view surface pixel(s) color/intensity (read/write pixel(s) color/intensity of a specified portion of the display surface).

The concept of setting the view surface overlay mode (how a new pixel affects the old pixel on the view surface. See Raster Extensions to the Core System. Section 10.2.5), needs a set of new USET options.

Note: Both Painted Polygons, and direct view surface operations on a group of Pixels require either a current pixel array or current pixel file. To pass the information this array/file must itself be described by the way in which the pixel color information is packed.

7. Inquiry and Redefining

In order that the application programmer have full control over the system, and make the best use of it, a set of inquiry and redefining functions have been introduced into the Core System. They should also be implemented in the GCS-3D system.

The user should be able to inquire: the current color or intensity (by index or value), color or intensities of vertices of a polygon, total global colors or a subset of it (active color set/table lookup), pixel array/file, pixel array format,... (see Raster Extensions to the Core System).

The user should also be able to change the active color table (both by value, and by index). This could be done by REDEFINE function calls (see Raster Extensions to the Core System, Section 4.2.1.3).

A new set of subroutines are needed in GCS-3D for these functions.

8. Control

In standard line drawing systems, output primitives can always be displayed as soon as they are defined. But if we want hidden surfaces to be removed (raster systems), then usually all output primitives must be defined before the hidden surface algorithm can be applied, and then the result can be displayed. Therefore some control function is needed to indicate when all output primitives have been defined, to initiate the hidden surface algorithm. The Core System's Batch of Updates function is for this purpose, and has two capabilities: initiate a NEWFRAME action (update the display surface to contain only the defined, visible, retained segments/frames), or do not initiate a NEWFRAME action.

Hidden Surface Algorithms are expensive (in computer time), when applied to pictures with any complexity. In order to avoid this expense when doing real time work, there should be an alternative display method, and the user should be able to choose which display mode is appropriate. The alternative is fast mode, in which output primitives and boundaries of the polygons are displayed immediately, so the batching has no effect, because the view surface is always up to date. There are two other options, namely, hidden line mode, and hidden surface mode. In either of these two modes nothing is displayed until the end of batch occures (at which time hidden line or surface algorithm is initiated). Both of these depend on Pseudo Display File.

Therefore, for batching of updates a new subroutine should be introduced to optionally initiate a NEWFRAME action

corresponsing to calling UPOST. The NEWFRAME would depend on the current setting for display mode, and on the type of display device.

Also three new USET options should be introduced to correspond to the display modes (fast mode, hidden line mode, and hidden surface mode).

PART II: RASTER GRAPHICS EXTENSIONS TO THE CORE GRAPHICS SYSTEM

1. Introduction

Two trends in computer graphics have motivated the work reported here. The first trend is the development of portable, device-independent graphics subroutine packages for use with various line-drawing graphics devices, both passive and interactive. This trend has most recently focussed on the Core Graphics System, a proposed standard graphics package (GSPC 77). The Core System appears destined to become the basis for ANSI (national) and ISO (international) standards.

The second trend is the rapid emergence of raster graphics hardware out of its traditional preserves of image display and process control into the world of interactive graphics. This has occurred because of rapid change in the economics of large scale integration and because of user demand for the color and shaded areas so easy to provide with raster graphics and so difficult to provide with line-drawing graphics. In many applications, raster graphics is a very natural complement to line-drawing graphics, while in other applications, raster graphics will sooner or later replace line-drawing graphics. Still other applications, never considered practical with line-drawing graphics for reasons of either economics or display aesthetics, suddenly become feasible with raster graphics.

We have been developing standard software for line-drawing graphics, while raster graphics grows at a rapid pace. This is not unexpected, because standards must of necessity lag the technology. It was in recognition of this necessary lag and of the rapid developments and changes in raster graphics that the charter to be Core System developers was to exclude consideration of raster graphics

However, four years after the workshop which chartered and solidified the graphics standards movement, it seems apprcpriate to extend existing line-drawing graphics packages to include raster graphics concepts, so that the new technology can be gracefully integrated with the old.

2. Related Work

Several line-oriented graphics packages with some raster graphics capabilities have been implemented. In all cases, the basic intent is to deal with computer-synthesized images, taking advantage of the raster display's ability to draw (possibly colored) solid areas.

In a 2D package developed by Sproull and Newman (Sproul/Newman 75), a polygonal area is specified by calling the procedure FILLBEGIN, followed by a sequence of LINE or LINETO procedure invocations to define the polygon, followed finally by FILLEND to mark the end of the polygon. Polygons and other output primitives are contained in segments, whose numeric name is used to determine the priority of overlapping primitives. Within a segment, temporal sequence determines visibility.

Fulton and Duquet added a few area-oriented output primitives (circles, triangles, rectangles) to BASIC for use with a plasma panel (Fulton/Duquet 75). Each primitive can be erased, by respecifying the parameters used to generate it. This allows the appropriate cells of the plasma panel to be turned off. If several primitives overlap, temporal sequence determines which cells are turned off by an erasure.

The 2D Interactive Graphics Programming Language (GPL) was first developed by O'Brien and Bown (O'Brien/Bown 75). It is now provided by Norpack (a raster system manufacturer), with raster extensions, under the IGPL name IGPL77 . The system currently allows shaded rectangles to be drawn, and is being extended to include convex and concave polygons. The system

provides a segment-like facility, to group output primitives. Temporal sequence of creation determines visibility of overlapping primitives.

GPAC, a 2D system developed at Toronto by Reeves (Reeves 77), used the FILLBEGIN...FILLEND construct to define polygonal areas. Segments have a depth attribute which is used when output primitives from different segments overlap. Within a segment, temporal order of creation governs, with the most recently created primitive dominating others.

The group at MIT has developed several low-level access packages (AMG75, AMG76, AMG77), which provide an output-oriented capability, without provision for segmentation. The refresh buffer can be manipulated by translating, scaling, and rotating rectangular regions. Output primitives (lines, circles, text, polygons) can be placed into the buffer.

3. Design Objectives

The most fundamental design objective of this work is to extend the Core System for raster graphics in a fashion compatible with the design objectives of the Core System itself, so that the extended system might possess a conceptual integrity in its own right. These objectives, summarized in (van Dam/ Newman 78) and detailed in (GSPC), demand that simplicity, minimality of function (but without substantial loss of capability), and consistency be sought.

Raster graphics implies the need for color and intensity specifications, solid shaded areas (which can in turn hide or obscure other areas), and the display of images created by cameras, X-rays, radars, ultrasound, CAT, etc. While the extensions are thus designed to allow display of solid areas of given color or intensity, the extensions do not include the lighting models which would allow surface shading to be calculated by the extended core. Lighting models (as well as surface texturing models) are considered as higher-level application-oriented capabilities, which can be created using the capabilities of the extended Core Graphics System. Furthermore, these are still active research areas in their own right.

The prototypical raster graphics system for which the extensions are designed is shown in Figure 1.

The image creation system is often able to process lines, text, points, and areas specified in 2D. Its output is placed in the refresh buffer as the pixel values needed to cause the appropriate entities to be displayed. The size of

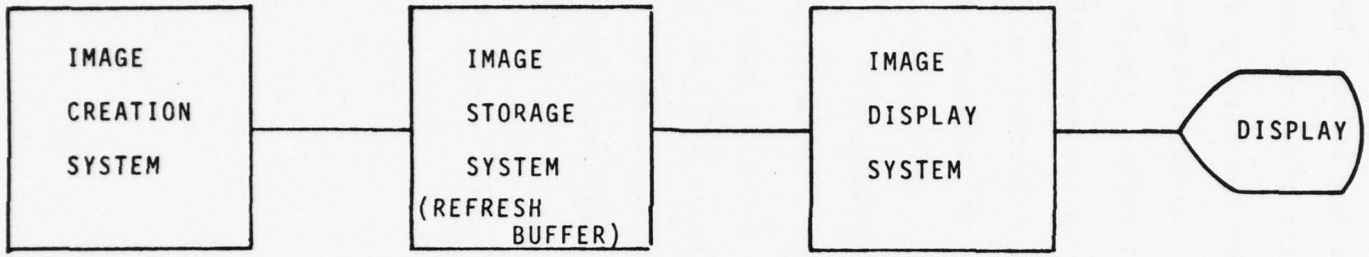


Figure 1
Prototype Raster Graphics Display System

the refresh buffer and number of bits/pixel are variable. The image display system reads the refresh buffer in synchronism with a TV raster scan, using the pixel values to control the color or intensity of points on the display screen. An optional look-up table in this system can dynamically assign intensities or colors to pixel values. There is a fixed 1-1 correspondence between pixels in the refresh buffer and points on the display screen (some contemporary raster systems do not impose this restriction). The interaction system allows devices such as keyboards and tablets to be used, with appropriate feedback displayed on the screen.

The look-up table is treated as an option, because it is not a standard raster display system capability. On the other hand, it is quite useful for image analysis and interactive. graphics. Thus another design objective is to incorporate the look-up table into the extensions, but making it transparent to applications which don't use it.

4. The Core Graphics System

A very brief review of the current capabilities of the Core System is appropriate, as it sets the stage for the following discussion of extensions to the Core System.

4.1 Graphics Output Primitives

Objects in 2D and 3D world (application) Coordinates are described to the Core System as combinations of moves, lines, line sequences (called polylines), markers (a generalization of points), and text. A current position is used as the starting point for lines and text strings. The output primitives' positions can be specified as absolute coordinates, or relative to the current position.

4.2 Output Primitive Attributes

The current value of these attributes determine the detailed appearance of output primitives when they are created. Once an output primitive is created, its attribute values can be changed only by deleting the primitive (using segments, described below), and then respecifying the primitive to the Core System after first establishing new attribute values. Primitive attributes are: linestyle (dot, dash, solid, etc.), linewidth, intensity, color, character font, character size, character spacing, character orientation, and character quality.

4.3 Grouping, Naming, and Modification

Each output primitive is placed into one and only one segment. Two different types of segments can be created: with one, the segment is displayed once and no record is kept of the segment or its contents; with the other, the segment is retained until explicitly deleted. Such retained segments,

which are named, are the unit of picture modifications for interaction. Segments can be deleted, renamed, and have their attributes modified. In addition to the grouping provided by segments, an output primitive within a named segment may be placed in a named group. Segments cannot contain references to other segments; thus picture hierarchies and copies of segments cannot be created. The segment and group names are used for pick (light pen) identification. When a segment is created, the current position is set to the origin, and all output primitive attributes are reset to their defaults.

4.4 Segment Attributes

Segments have dynamic attributes, which can be changed either while a segment is being constructed or at a later time. These attributes are visibility (whether the segment is actually displayed), detectability (whether the segment can be "seen" by a light pen or other pick device), highlight (calling the operator's attention to the segment by blinking or intensifying it), and an image transformation (applied, after the viewing transformation has been performed, to those output primitives surviving the clipping process).

4.5 Viewing Transformations

Viewing of 2D objects is specified with the traditional window and viewport except that the window may be inclined with respect to the principal axes. The object can be clipped against the window. In 3D, a viewer is imagined to be at an arbitrary point in three-space, looking toward some other point. The field of view is determined by specifying a rectangle in a projection plane, with the rectangle then mapped into the

viewport. If the viewport is infinitely far away, an orthographic projection results; otherwise, a perspective projection results. The 3D object can be clipped against a 3D solid volume: a pyramid for perspective projections, and a solid rectangle for parallel projections. The image of a viewed object, be it 2D or 3D, can be translated, scaled, and rotated after the image has been created. 2D viewing is a proper subset of 3D viewing, with all 2D objects in the Z = 0 plane.

4.6 Interaction

Five classes of devices for operator input are supported. Picks provide the segment and group names of output primitives. Locators provide a position, while valuators provide a scalar value. Text input devices provide character strings, and choice devices provide a selection from several alternatives. The common physical prototypes for picks are light pens; for locators, joysticks and tablets; for valuator, dials; for text, alphanumeric keyboards; and for choice, programmed function keyboards. Each device class has a system-defined feedback, which can be turned on or off. Picks, text, and choice, when enabled, can place event reports on an input queue, which can be interrogated by the application program. Locators and valuators, when enabled, can be sampled by the application program, and can also have their input information placed on the input queue in conjunction with the occurrence of an event.

5. The Raster Graphics Extensions

In this section the raster graphics extensions to the Core System are described. The extensions naturally fall into several areas, each of which is treated in a separate subsection. There is usually a discussion of design alternatives, followed by a more detailed discussion of the selected alternatives. A reference manual defining the details of each subroutine added to the Core System is an appendix to this paper.

5.1 Hidden Surface Removal

The Core System currently has no capabilities for defining surfaces, so the hidden surface and hidden edge removal have never been considered. However, this capability is central to effectively using raster graphics. The key question here is how the application programmer tells the graphics package which surfaces obscure other surfaces. There are several possible solutions.

For strictly 2D systems, temporal priority can be used: the most recently displayed segment or output primitive has priority over other segments and output primitives. This is particularly easy to implement. Alternatively, segments (or perhaps output primitives) can explicitly be assigned priorities, while within a segment, temporal sequence is used. Implementation of this latter case is more difficult than the former.

The assigned priorities can be either static (permanently fixed when the segment is created) or dynamic (changeable after the segment is created). In either case, this approach is sometimes called a 2½D system. The priorities are akin to the depths of segments along a Z-axis, and each segment lies

in a plane normal to the Z-axis. The segment with priority 1 (the highest priority) is in the Z=0 plane. The collection of segments is viewed from infinity on the negative Z-axis.

For 3D, the basic answer is that the X,Y,Z coordinates of the surfaces plus the viewing specification (where the viewer is located) provide all the information needed by hidden surface algorithms. However, there is more to the problem. In the Core System, many segments can simultaneously be displayed on the view surface, and each segment can have a different viewing specification. Hence the above "basic" answer is true only for individual segments.

The synthetic camera analogy for the Core System is useful in motivating the difficulty. Each segment and its viewing specification defines a "photograph", which is displayed within a viewport on the view surface. When several viewports overlap or are in fact identical (frequently the case), the photographs overlap. For line-drawings, this is no problem, since each photograph is just a collection of lines. Thus one photograph does not obscure another photograph.

For raster graphics, a photograph can include solid areas, which can obscure lines and other solid areas. One might attempt to resolve this by assigning priorities to each photograph (i.e., segment). But what of a view of a single object composed of multiple segments? The determination of the visibility of surfaces in each segment must be done in three dimensions, before the photograph is actually created, considering all the segments defining the object. One way to permit this is to define a second level of structuring in the

Core System - a collection of segments. Hidden surface removal in 3D would be done on each such collection of segments. This approach is unacceptable, because it would introduce a second level of structuring to the Core System. Furthermore, the priority of the photographs produced by each such collection of segments must still be resolved.

There is another, more attractive way to deal with hidden surface removal. In level 4 of the Core System, the viewing transformation specifies not a projection from 3D world coordinates into a rectangular 2D normalized device coordinate viewport on the view surface, but rather a transformation from 3D world coordinates into a 3D solid rectangular viewport in 3D normalized device coordinates. The display on the view surface is then a parallel projection of the entire contents of 3D NDC space (the unit cube) onto the front plane of the unit cube. Figure 2 illustrates this. Two segments with identical 3D viewports then have their output primitives in the same part of 3D NDC space. The hidden surface algorithm operates on all of 3D NDC space, producing a single unified image or photograph on the view surface. Thus there is no need to be concerned about the priority of several overlapping photographs on the view surface. If two output primitives have identical coordinates in 3D NDC space, then temporal priority can still be used to determine visibility. The most recently created output primitive would take priority over others.

The transformation of a 3D view volume (be it the truncated pyramid of a perspective view or the solid rectangle of a

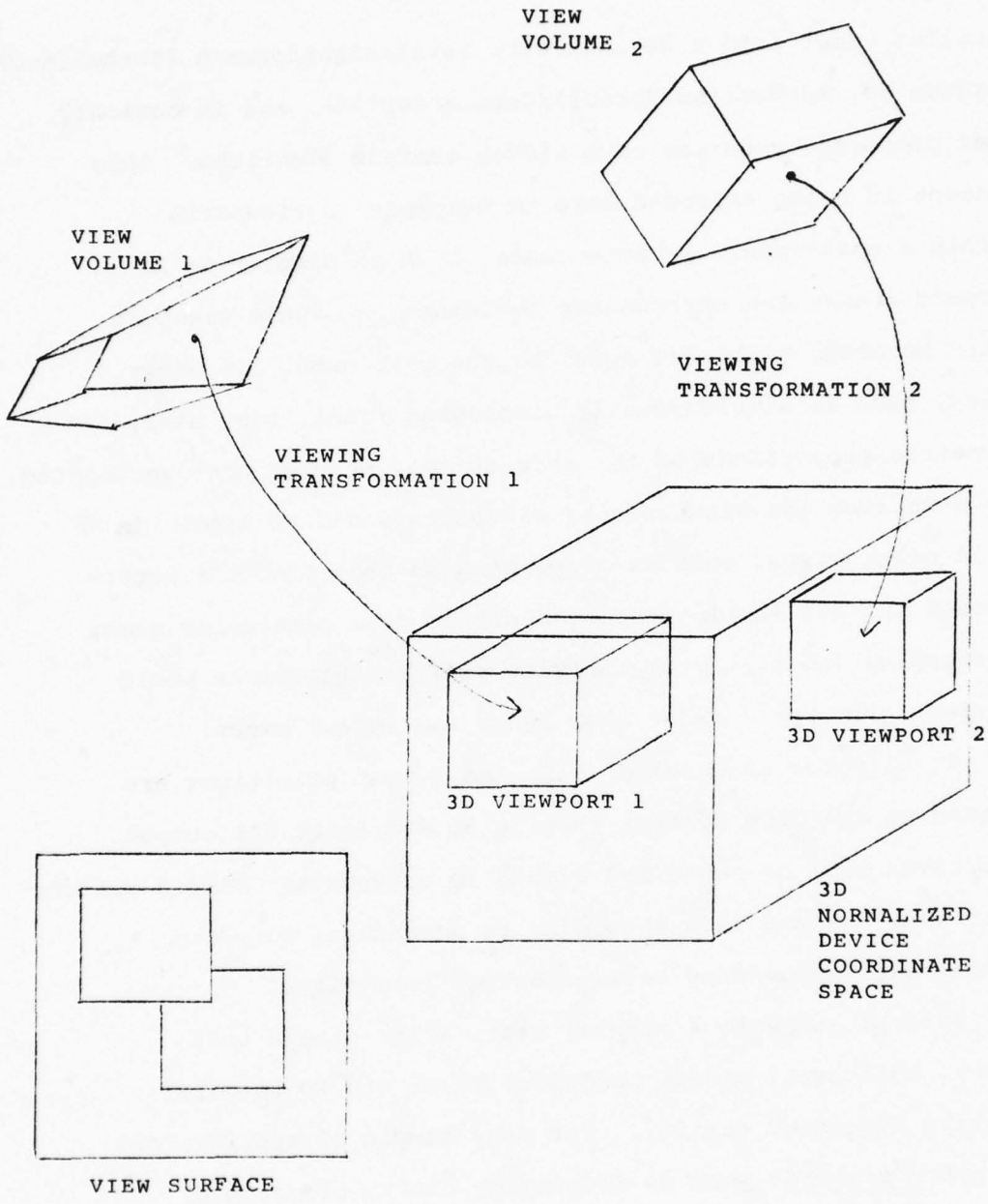


FIGURE 2

CREATION OF A DISPLAY FROM SEVERAL VIEW VOLUMES

parallel view) into a 3D unit cube is straightforward (Sutherland/Hodgman 74, Sutherland/Sproull/Schumacker 74), and is commonly used preparatory to use of a hidden surface algorithm. This concept is being extended here to multiple 3D viewports within a unit cube. In some cases, such as displaying an airport runway and surrounding buildings, a single viewport would be used, with size equal to the unit cube. In other cases, such as simultaneously displaying front, top, side, and isometric projections of the same object, several non-overlapping (as seen from the viewsurface) viewports would be used. In still other cases, such as displaying an object with a superimposed cut-out showing enlarged detail of a particular area, overlapping (as seen from the view surface) viewports would be used. Figures 3 and 4 show these two latter cases.

If clipping is enabled while the output primitives are defined to the Core System, then in 3D NDC space all output primitives will be contained within 3D viewports. Hidden surface algorithms can use this knowledge to advantage, to avoid unnecessary comparisons between output primitives.

Several attractive special cases arise within this general conceptual model. Consider first the 2D temporal priority discussed earlier. For many simple 2D applications of raster graphics such as displaying charts and graphs, 2D temporal priority is an attractive and useful concept for a programmer to use. This concept is in fact the default for the Core System, because all 2D viewports (which is the

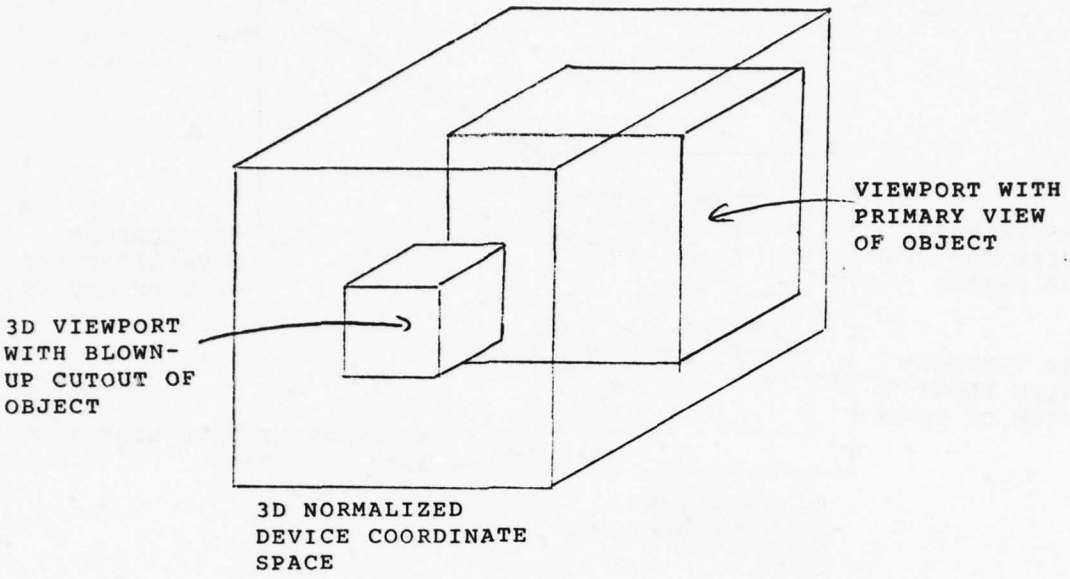


FIGURE 3

ARRANGEMENT OF TWO 3D VIEWPORTS SO THAT RESULTING DISPLAY APPEARS TO HAVE THE FORWARD 3D VIEWPORT SUPERIMPOSED IN FRONT OF THE REARWARD 3D VIEWPORT

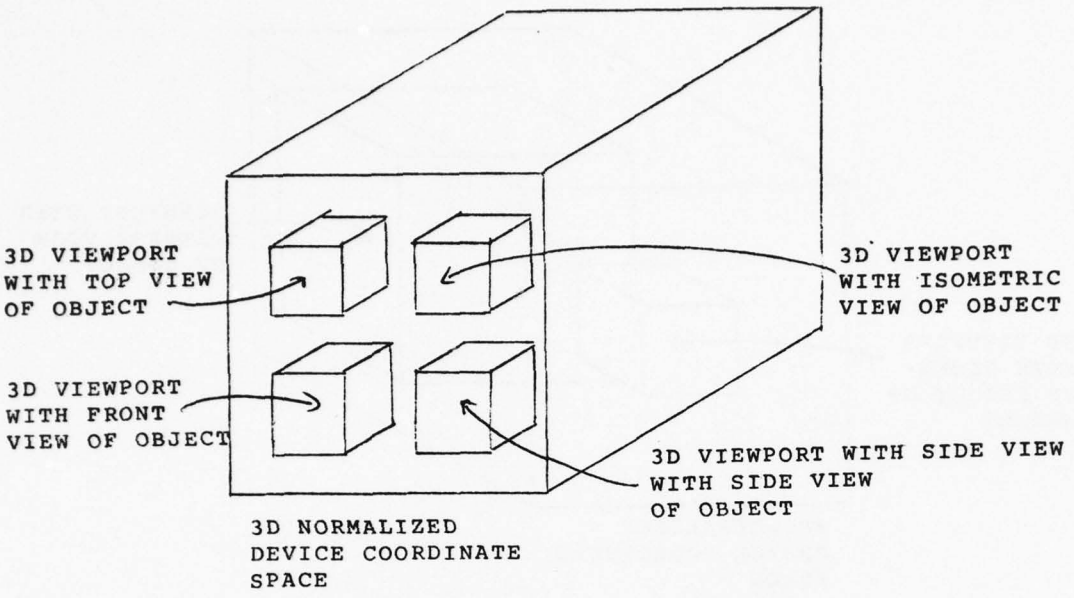


FIGURE 4

ARRANGEMENT OF FOUR 3D VIEWPORTS SO THAT THE RESULTING DISPLAY HAS FOUR SEPARATE VIEWS ON VIEW SURFACE

typically-used default) lie in the front face of 3D NDC space (the Z=0 plane). Hence output primitives in NDC space have no depth, so the hidden surface algorithms have no basis for deciding which output primitives are visible. It is in just this case that temporal priority is used.

As a second case, consider the 2½D case, wherein each segment has an explicit priority for visibility. All output primitives, however are 2D.

The effect of 2½D can be achieved by using segment priorities in the range 0-1. A segment with priority P and 2D viewport defined by corner points (X_{min},Y_{min}), (X_{max},Y_{max}) would in fact have a 3D viewport defined by corner points $(X_{min},Y_{min},P),(X_{max},Y_{max},P)$. That is, the 3D viewport is in the Z=P plane. Hence the effect of the depth priority for a 2D segment is achieved by putting the image of the segment on the appropriate constant Z plane in 3D NDC space , as in Figure 5.

If the depth priorities are dynamic, changing a priority can be effected by applying an image transformation to the segment: a translation along the Z-axis of NDC space would be used to reposition the image in the appropriate plane of constant Z.

Thus the facilities for either a static or dynamic 2½ D priority capability are available. A novice programmer might have trouble using the facilities in the context of 3D viewports and NDC space, but a very simple set of additional

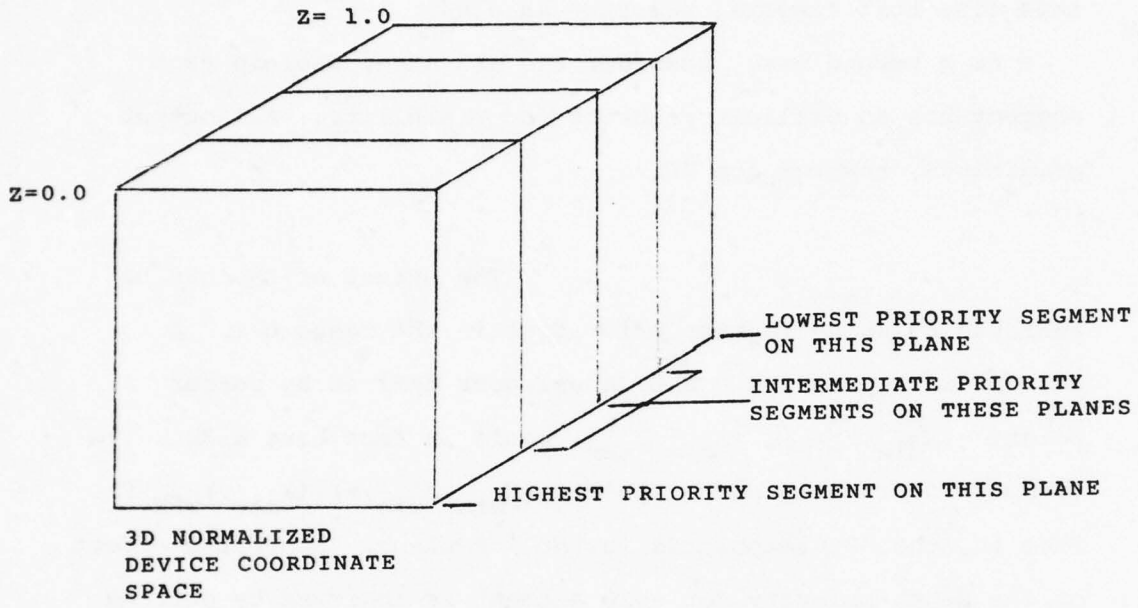


FIGURE 5
USE OF VIEWPORTS ON PARALLEL PLANES IN 3D NDC SPACE
TO IMPLEMENT EXPLICIT (2½D) DEPTH PRIORITY

subroutines could be built on top of the extended Core System to make the facilities available in a more tractable form.

It so happens, then, that the Core System already contains the key concept needed to provide hidden surface removal in a general and flexible way - 3D viewports and 3D NDC space. Furthermore, the concept can be used to provide two other hidden surface concepts commonly used in 2D - temporal priority and 2½D priority.

5.2 Polygons

The basis for shaded (or colored) areas and hidden surface removal is the specification of bounded surfaces, which then can obscure either other bounded surfaces or the other primitives: lines, markers, and text. The Core System extensions deal only with planar bounded surfaces which can be defined by polygons. Bounded surfaces defined by conic sections, etc. are not treated, nor are non-planar surfaces.

Polygons can be specified in several ways. Several simple raster packages use a sequence such as:

```
BEGIN_FILL
     MOVE
     LINE
     LINE
     LINE
      .
      .
      .
     LINE
END_FILL
```

The bracketed MOVE and LINE calls define a sequence of lines, which is automatically closed by the END_FILL, forming a polygon. Several packages use this approach, which does, however, have

the disadvantage of using the same output primitives for two purposes: drawing lines, and defining polygons. This can hinder program debugging, because the meaning of an isolated call to LINE depends on whether a BEGIN_FILL is currently active. Furthermore, many subroutine calls are needed to define a polygon.

This first objection can be removed by introducing an EDGE output primitive, to be used in place of LINE within the BEGIN_FILL...END_FILL pairs:

```
BEGIN_FILL
    EDGE(X,Y)
    EDGE (X,Y)
     :
     :
END_FILL.
```

The second objection remains, however, and can be removed by using a single polygon subroutine. This has been done in the extensions, with the four procedures:

```
POLYGON_ABS_2(X_ARRAY,Y_ARRAY,N)
POLYGON_REL_2(X_ARRAY,Y_ARRAY,N)
POLYGON_ABS_3(X_ARRAY,Y_ARRAY,Z_ARRAY,N)
POLYGON_REL_3(X_ARRAY,Y_ARRAY,Z_ARRAY,N)
```

Many objects displayed graphically consist of numerous interconnected polygonal faces. If each face were specified by a separate polygon primitive, coordinates of shared vertexes would be passed to the Core System several times, and connectivity/shared edge information would have to be deduced from the equality of vertex coordinates. To avoid these problems, a second way of defining polygons is provided: the polygon mesh. Polygons in a polygon mesh are defined in a 2-step process:

- A list of vertices, numbered from 1 to N, is specified.
- Each polygon is defined by giving a list of vertex numbers. The vertices are connected in the order given, with the last vertex being connected to the first for closure.

The vertex list is specified with

```
SET_POLYGON_MESH_ABS_2(X_ARRAY,Y_ARRAY,N)
      "          REL_2 (X_ARRAY,Y_ARRAY,N)
      "       ABS REL_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)
      "           REL_3
```

A polygon in the mesh is specified with

```
POLYGON_EDGE_MESH_ABS_2(VERTEX_LIST,N)
```

A polygon mesh exists entirely within a segment, and is terminated either by another SET_POLYGON_MESH or by closing the segment.

Polygon meshes have several attributes. A polygon may be used to define a polyhedron. If so, the polyhedron attribute is so set. This allows the hidden-surface algorithm to remove back-facing polygons. The default attribute value is for the general case, which is non-polyhedral.

If a polygon mesh does define a polyhedron, it may be convex or non-convex. Again, hidden-surface algorithms can take advantage of this knowledge. The default is the general case, non-convex. The shading of polygons can be controlled in one of three ways: by the color or intensity attribute discussed in section 5.3, or by the pixel array attribute discussed in section 5.4, or by interpolated shading. The polygon type attribute is used to control which of these three types of shading is used. The attribute is set by the procedure

```
SET_CURRENT_POLYGON_TYPE(TYPE).
```

If the type is color/intensity (as opposed to pixel array), then the polygons in a mesh can each have different colors and intensities. Normally, as a default, each visible polygon is displayed with the specified value. The interpolation value can be used to request Gouraud (continuous) shading to be applied to all polygons in a mesh.

5.3 Color and Intensity

In raster graphics, precise control of color and intensity are central to many applications. The Core System has a color attribute and an intensity attribute. However, there is no way to precisely determine or specify colors and intensities. Futhermore, intensity is in fact one part of color specification, so there is a distinct non-orthogonality between the color and intensity attributes. That is, a large range of colors, including all grey levels, can be specified using additive red, green, and blue primaries, or using hue, saturation, and intensity. Given a color attribute, intensity is redundant. In the extensions, a view surface is initialized for either color or intensity use: the attributes cannot be mixed. Intensity is retained only as a convenience for grey-level applications, and to maintain upward-compatibility.

Because the specific ranges of colors and intensities can vary greatly from one raster display to another and are so important to many applications, only one view surface may be selected at a time. This removes from the Core System the impossible task of trying to second-guess the programmer in mapping a color attribute value to two dissimilar raster displays.

The discussion here centers on the color attribute and functions for its control. A parallel set of capabilities for the intensity attribute also exists, and is described in the appended Reference Manual. Conceptually, the extensions define an ordered set of colors specified by (R,G,B) triples: $\{(R_i,G_i,B_i)\}$. This set is called the active color set. The size of the set varies among view surfaces, and can be determined with the inquiry function INQUIRE_NUMBER_COLOR(M). Another inquiry function, INQUIRE_COLOR(I,R,G,B), can be used to find the values of the R,G, and B components in the I^{th} triple.

The color attribute for output primitives is specified in one of two ways: by value, and by index. The function SET_CURRENT_COLOR(R,G,B) specifies the current value of the color attribute. If the exact (R,G,B) triple specified is not included in the active color set, then a best fit to an available triple is performed. Alternatively, the function SET_CURRENT_COLOR_INDEX(I) specifies that the I^{th} triple from the ordered set is to be used as the color attribute.

This concept can be extended, as an option, for use with color look-up tables. With a look-up-table, the previously-described active color set is a subset of the <u>global color set</u>. The size of the active color set is just the number of entries in the look-up table. The size of the global color set is the size of the universe of colors which can be placed in the look-up table, and is usually two raised to the number of bits per entry in the table. INQUIRE_NUMBER_GLOBAL_COLORS(N) can be used to find the size of the global color set. The

(R,G,B) triple for global color J is found with INQUIRE_GLOBAL_COLOR(J,R,G,B). An active color (an entry in the look-up table) can be changed in two ways: by value and by index, similarly to setting the value of the color attribute. REDEFINE_COLORINDEX(J,R,G,B) changes the J^{th} active color to (R,G,B). If (R,G,B) is not in the set of global colors, a best-fit is performed. Alternatively, an "associative" change can be specified with REDEFINE_COLOR(R1,G1,B1,R2,G2,B2). The active color (R1,G1,B1) is changed to (R2,G2,B2). If needed, best fits are performed on both colors. The current color attribute must of course belong to the set of active colors. If the current color attribute has index I in the active color set and the meaning of index I is changed to some new color from the global color set, then the current color attribute is also changed.

This conceptual model for working with the color attribute is usable with a wide range of systems, from the inexpensive two to eight color systems (one to three bits per pixel) without look-up table, through the typical eight bit per pixel system with a 256 entry look-up table of twelve bits per pixel, up to the 24 to 36 bit per pixel systems.

The extended Core System need not always explicitly store either the active or global color set, but need only be able to generate triples in the set. For instance, the device driver for a 24 bit per pixel system could use the 24 bit color index as the color specification, by simply dividing the 24 bits into three groups of eight bits each. On

the other hand, the device driver for an eight color system can trivially store an eight-entry table.

The look-up table is typically used to partition planes of the refresh buffer into subsets, to store several pictures. The concepts described here permit such manipulations.

5.4 Pixel Arrays

Pixel Arrays are two-dimensional arrays of intensity or color values. They occur in several environments. In traditional image processing, the pixel arrays represent real objects, and come from devices such as cameras, X-rays, ultrasound scanners, and CT systems. In some cases these devices produce direct digital information, and in other cases film must be scanned and digitized.

Pixel arrays representing synthetic or modelled objects are regularly created by computer programs, sometimes using quite sophisticated modelling, lighting, and surface texturing. Artists often use interactive painting programs to create aesthetically pleasing scenes defined by pixel arrays. Part of the objective of extending the Core System for raster graphics is to allow such pixel arrays to be displayed easily, along with lines and shaded polygons. Pixel arrays could easily be treated as a rather special type of output primitive, being placed directly onto the front plane of 3D NDC space. Then, however, other constructs in 3D NDC space might be obscured. Furthermore, using NDC space is unattractive, because then the mixing of pixel arrays and other output primitives is greatly complicated due to the use of two different coordinate systems. Yet applications such as interactive image analysis and

interpretation, as well as interactive graphics arts applications, demand such mixing.

An attractive alternative, used in the extensions, is to place the pixel arrays into world coordinates, again as a special type of output primitive. Where would the pixel array be placed? Presumably on some plane, oriented arbitrarily in 3D world coordinates.

One might wish to display not the entire rectangular pixel array, but rather some portion thereof: this would occur if a photo were cropped around an irregular contour. Indicating the portion of the pixel array to be ignored can be done in several ways. A special pixel value meaning "ignore me" might be used, or as is done in the extensions, a polygon can be used to circumscribe that part of the pixel array to be displayed.

How can the plane on which the pixel array will be displayed and the polygon circumscribing some or all of the pixel array be best specified? A special construct for this would work, but is quite unnecessary, because the extensions already include polygons, as described in an earlier section. All that is needed is some mechanism for mapping a pixel array onto a polygon, and for indicating that a polygon is to be shaded not with the current color intensity attribute, but rather with a pixel array. This latter matter is handled by the POLYGON_TYPE attribute, which has three values: plain, interpolated, and painted. A plain polygon is displayed using the current color or intensity, while a painted polygon is displayed using the current pixel array and pixel array to

polygon mapping. The default polygon type is plain. An interpolated polygon is shaded by linear interpolation of vertex colors or intensities so that Gourand shading is possible as a special case.

If a polygon is to be painted, a mapping is needed as shown in Figure 6 . A polygonal part of the pixel array is mapped into a similar polygonal part of the 3D world. Appropriate averaging or convolutions are used to find the correct pixel values for that part of the view surface surrounded by a visible painted polygon.

The mapping from a pixel array to a 3D world coordinate polygon is specified in several steps. The pixel array, for convenience, can be thought of as having an arbitrary size, defined by SET_PIXEL_ARRAY_EXTENT. The two parameters are the width and height of the pixel array, in arbitrary units (some applications may conveniently think of these as world coordinate units). Figure 6 shows the pixel array and its length and height. The (0,0) origin is always at the lower left of the pixel array, in the center of the pixel. Similarly the point (X_LENGTH,Y_LENGTH) is in the center of the upper right pixel. Using this pixel array coordinate system, three points in the pixel array (not necessarily centers of pixels) are defined and associated with three vertices of the world coordinate polygon. The three points in the pixel array and three points on the world coordinate polygon define a mapping which transforms the pixel array coordinate space onto the plane of the three polygon points. Whatever part of the pixel array falls within the polygon is visible, as in Figure 6 .

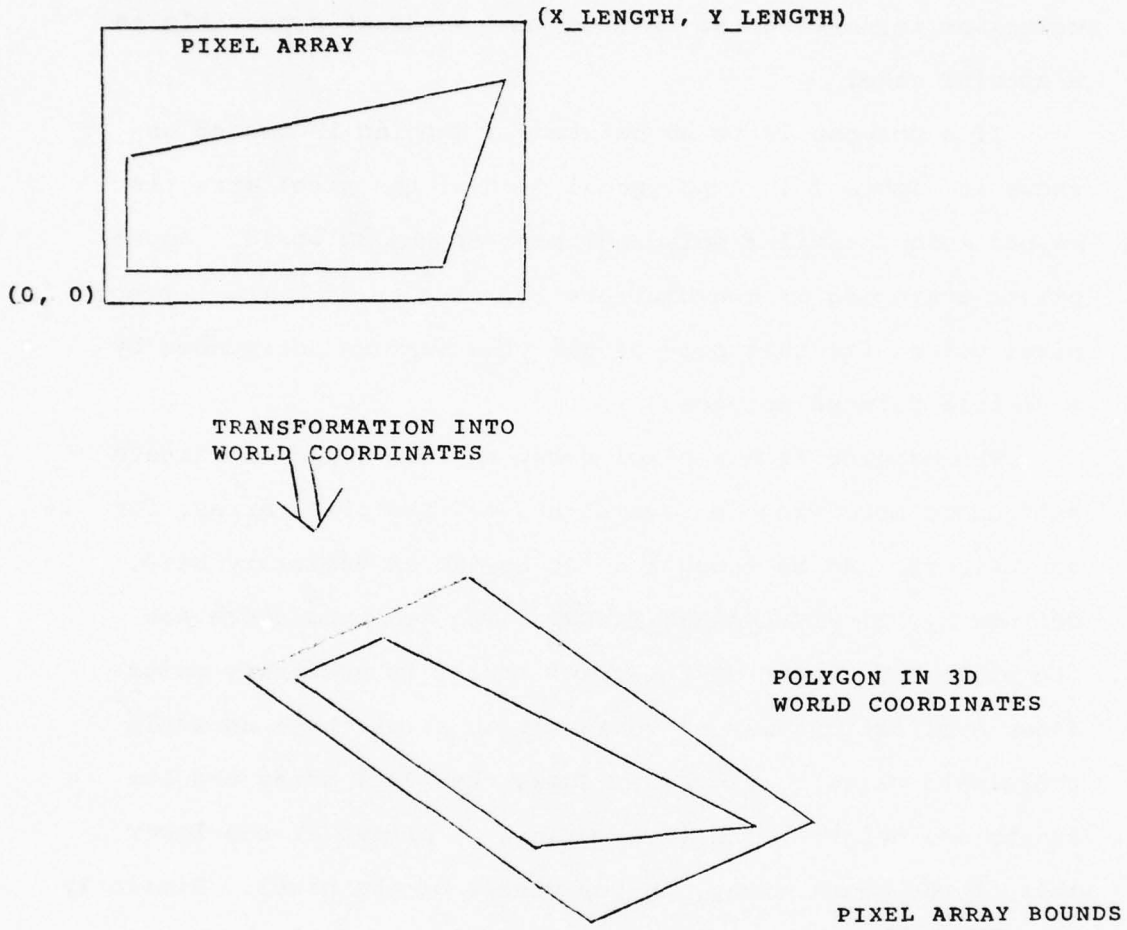


FIGURE 6
TRANSFORMATION OF PIXEL ARRAY INTO WORLD COORDINATES. ONLY PORTION OF ARRAY WITHIN POLYGON IS DISPLAYED.

(Naturally the three points cannot be colinear). This mapping is defined with the function SET_VERTEX_MAP.

This approach is rather powerful. It can be used for texture mapping, placing pictures on walls of rooms being modelled and displayed, but it can also easily be used to display rectangular images in a rectangular polygon on the view surface. This notion of mapping images onto polygons and thence the polygons to the view surface is at the heart of how the extensions to the Core System deal with images.

5.5 Operations on the View Surface

For raster graphics systems the view surface and refresh buffer are one and the same. The refresh buffer is made directly available to the application programmer, to read or write individual pixels or arrays of pixels. Pixels are of course addressed using the intermediary of normalized device coordinates. This is perhaps the most and only substantial area in which the extensions depart significantly from the Core System philosophy: for vector systems, the Core System explicitly does not make a real or pseudo display file directly available to the application program for manipulation and modification.

This has been done for numerous reasons. Some interaction techniques must directly modify and restore portions of the refresh buffer. Tremendous amounts of computer time might go into calculating pixel values for a single picture. That time would be lost if the view surface pixel array could not be saved. (A similar cost does not exist with line-drawing display files.) Powerful visual effects can be achieved by some view surface (pixel array) manipulations, or by combining on a screen the outputs of numerous different programs which creat images in different ways.

On the other hand, no geometrical transformations on the view surface have been defined. If the view surface pixel array can be read and written, then a multitude of geometrical (rotate, scale, translate, etc.) and image-oriented (average, filter, etc.) operations can be implemented.

5.6 Control

In a standard line-drawing graphics package, output primitives can always be displayed as soon as they are defined. If hidden surfaces (or edges) are to be removed, however, then numerous factors suggest that all output primitives must usually be defined before any output primitives can be displayed. Some control function is needed to indicate when all output primitives have been defined, so that a hidden surface algorithm can be initiated. Without such a function, the Core System would be forced to display each visible output primitive as it is defined, possibly also removing from the display output primitives (or portions thereof) obscured by the new output primitive.

The Core System's batch of updates function is just right for this purpose. The effect of the END_BATCH_OF_UPDATES function depends on which one of three display modes has been set: "fast", "hidden edge", and "hidden surface". In fast mode, output primitives are displayed immediately, so the batching has no effect beyond what it normally does: guarantee that the view surface is up to date. If either of the other modes has been set, nothing is displayed until the end of batch occurs, at which time the (possibly slow) hidden edge or surface algorithm is initiated. If a line-drawing view surface has been selected, then "hidden surface" display mode, "fast", exists to give the application program explicit control over how often the hidden edge or surface algorithm is used, and to give compatibility with the existing Core System.

Another change to the Core System's control is a restriction to one active view surface, as discussed in the earlier section on color and intensity attributes.

6. Summary

In general, the Core System proved to be a receptive host for the raster graphics extensions. The 3D NDC space was useful, and the Core System's emphasis on consistency and minimality challenged us to examine numerous alternatives before a final design.

PART III: DRAFT REFERENCE MANUAL: RASTER GRAPHICS EXTENSIONS TO THE CORE SYSTEM

2 OUTPUT PRIMITIVES

2.2 FUNCTIONAL CAPABILITIES

2.2.6 POLYGON PRIMITIVES

2.2.6.1 POLYGON

POLYGON_ABS_2 (X_ARRAY,Y_ARRRAY,N)
POLYGON_ABS_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)
POLYGON_REL_2 (X_ARRAY,Y_ARRAY,N)
POLYGON_REL_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)

This function is used to define a polygon with a vertex list in world coordinates. The polygon is the region enclosed by the connected sequence of lines that starts at the first vertex in the array, joins all the other vertices in order, then returns to the first vertex to close the defined area.Because closure is implied the CP is set to the coordinates of the first vertex at the completion of the primitive.

The color pattern of the interior of the polygon will depend on the current polygon type and the corresponding set of primitive attributes (see SET_POLYGON_TYPE).

General Errors:
1. There is no selected view surface.
2. No segment is open.
3. N is <= 2.
4. Some vertices are not coplanar (beyond an as yet undefined epsilon).

Errors for Painted Polygons:
5. There is no pixel array format specified.
6. A vertex index given in SET_PIXEL_MAP call is > N.
7. A vertex defined in pixel map coordinates falls outside the pixel map.
8. The triangle defined on the pixel map is not similar to the one defined in world coordinates (the length of all 3 sides of both triangles must be proportional to be similar triangles).

Errors for Shaded Polygons:
9. The number of vertices specified in the current vertex color list (see SETTING_VERTEX_COLORS) is not equal to N.

2.2.6.2 SET_POLYGON_MESH

SET_POLYGON_MESH_ABS_2 (X_ARRAY,Y_ARRAY,N)
SET_POLYGON_MESH_ABS_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)
SET_POLYGON_MESH_REL_2 (X_ARRAY,Y_ARRAY,N)
SET_POLYGON_MESH_REL_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)

This function is used to begin the definition of a new polygon mesh. It specifies the entire vertex list for the mesh of polygons defined in world coordinates. This list will be referenced by the POLYGON_EDGE_LIST function. The CP is not used or updated by this function.

The current polygon mesh shape in effect (see SET_CURRENT_POLYGON_MESH_SHAPE) when the SET_POLYGON_MESH function is invoked determines the type of the mesh.

A mesh is made up of a series of polygon elements defined using the POLYGON_EDGE_LIST command.

A particular mesh is completed (closed) when another SET_POLYGON_MESH function is invoked or a Segment is closed.

Default -- N equals zero.

Errors:
1. There is no view surface selected.
2. No segment is open.
3. N <= 2.

2.2.6.3 INQUIRE_POLYGON_MESH

INQUIRE_POLYGON_MESH_ABS_2 (X_ARRAY,Y_ARRAY,N)
INQUIRE_POLYGON_MESH_ABS_3 (X_ARRAY,Y_ARRAY,Z_ARRAY,N)

These functions return the current polygon mesh vertex lists in absolute world coordinates.

Errors: None.

2.2.6.4 POLYGON_EDGE_LIST (VERTEX_INDEX_ARRAY,N)

This function is used to define a single polygon in a polygon mesh, by specifying which vertices from the vertex list (see SET_POLYGON_MESH) are to be used. The polygon is the region enclosed by the connected sequence of lines that start with the first vertex specified by the VERTEX_INDEX_ARRAY, joins all the other vertices in order, then returns to the first vertex to close the defined area. Because closure is implied the CP is set to the first vertex at the completion of the primitive.

The color pattern of the interior of the polygon will depend on the current polygon type and the corresponding set of primitive attributes (see SET_POLYGON_TYPE).

General Errors:
1. There is no selected view surface.
2. No segment is open.
3. N is <= 2.
4. Some vertices are not coplanar (beyond an as yet undefined epsilon).
5. A VERTEX_INDEX is out of range of the mesh vertex list.
6. There is no vertex list in existence.

Errors for Painted Polygons:
7. There is no pixel array format specified.
8. A vertex index given in the SET_PIXEL_MAP call is > N.
9. A vertex defined in pixel map coordinates falls outside the pixel map.
10. The triangle defined on the pixel map is not similar to the one defined in world coordinates (the length of all 3 sides of both triangles must be proportional to be similar triangles).

Errors for Shaded Polygons:
11. The number of vertices specified in the current vertex color list (see SETTING_VERTEX_COLORS) is not equal to N.

4.2 FUNCTIONAL CAPABILITIES

4.2.1.1.1 SET_CURRENT_POLYGON_TYPE (TYPE)

This function sets the current system-maintained attribute value of the polygon type to either plain, painted or shaded. A plain polygon is filled with a single color specified by the current color (or current intensity) attribute. A painted polygon is filled with an image made up of differently colored pixels. The current pixel array, set by other primitive attribute routines, defines the pixel values. A shaded polygon is filled by linearly interpolating between the color specified at each vertex (Couraud shading). The color (or intensity) value at each vertex are set by other primitive attribute functions.

Default -- Currrent Polygon Type is Plain.

Errors:
1. No segment is open.
2. Invalid attribute value.

4.2.1.1.2 SET_CURRENT_COLOR

SET_CURRENT_COLOR (R,G,B)

This is the currently defined procedure for setting the color attribute. R,G,B are in the range of 0,1.

Errors:
1. No view surface is selected.
2. Segment is not open.
3. Invalid attribute value.
4. Type of the selected view surface is gray level rather than color.

SET_CURRENT_COLOR_INDEX (I) 1 <= I <= N

The current color is set to the Ith color of the currently selected view surface. This is the color which would be returned by a call to INQUIRE _COLOR_VALUE (I,R,G,B), with the same view surface selected.

Errors:
1. No view surface is selected.
2. Segment is not open.
3. Invalid color index.
4. Type of the selected view surface is gray level rather than color.

SET_CURRENT_INTENSITY (INTENSITY)

This is the currently defined procedure for setting the intensity attribute. INTENSITY is in the range of 0,1.

Errors:
1. No view surface is selected.
2. Segment is not open.
3. Invalid attribute value.
4. Type of the selected view surface is color rather than gray level.

SET_CURRENT_INTENSITY_INDEX (I) 1 <= I <= N

The current intensity is set to the Ith INTENSITY of the currently selected view surface. This is the intensity which would be returned by a call to INQUIRE_INTENSITY_VALUE (I, INTENSITY) with the same view surface selected.

Errors:
1. No view surface is selected.
2. Segment is not open.
3. Invalid intensity index.
4. Type of the selected view surface is color rather than gray level.

4.2.1.1.3 SET_VERTEX_COLORS

SET_VERTEX_COLORS (R_ARRAY, G_ARRAY, B_ARRAY, N)
SET_VERTEX_INTENSITIES (INTENSITY_ARRAY, N)

This function sets the arrays containing colors (intensities) of different vertices of a shaded polygon. These vertices correspond to the list of vertices given either by a POLYGON or a POLYGON_EDGE_LIST commands. These R,G,B's (intensities) are rounded by the core system to the nearest color (intensity) displayable on the selected view surface. R,G,B's (intensities) are in the range of 0,1.

Defaults -- N = 0.

Errors:
1. No view surface is selected.
2. No segment is open.
3. N < =2 .
4. Invalid attribute value.
5. Type of selected view surface is gray level rather than color, or vice versa.

SET_VERTEX_COLORS_BY_INDEX (INDEX_ARRAY, N)
SET_VERTEX_INTENSITIES_BY_INDEX (INDEX_ARRAY, N)

This function sets the array containing the indices of colors (intensities) of different vertices of a shaded polygon. These vertices correspond to the list of vertices given either by a POLYGON or a POLYGON_EDGE_LIST command.

Defaults -- N = 0.

Errors:
1. No view surface is selected.
2. No segment is open.
3. N < =2 .
4. Invalid index.
5. Type of selected view surface is gray level rather than color, or vice versa.

4.2.1.1.4 SET_PIXEL_ARRAY (COLUMNS,ROWS,ARRAY)

This function is used to pass an image's pixel array and to give its dimensions. The pixel array contains either intensity or color indices, depending on the type of view surface selected (see SELECT_VIEW _SURFACE). The method used here of passing an array is only feasible for small image arrays.

The alternate method for designating a pixel array is with the SET_PIXEL_FILE function. The most recently used of these two functions defines the pixel array to be used when a painted polygon is specified.

This function also sets the defaults for the pixel array extent and the vertex map.

Defaults -- A 2 by 2 array contains the index of the default intensity or color attribute.

Errors:
1. No view surface selected.
2. No segment is open.
3. COLUMNS <= 0 or too large.
4. ROWS <= 0 or too large.

4.2.1.1.5 SET_PIXEL_FILE (COLUMNS,ROWS,FILE_NAME)

This function is used to specify the name of a file that holds pixel array and to give its dimensions.

The alternate method for designating a pixel array is with the SET_PIXEL_ARRAY function. The most recently used of these two functions defines the pixel array to be used when a

painted polygon is specified.

This function also sets the defaults for the pixel array extent and the vertex map, as described by SET_PIXEL-ARRAY_EXTENT and SET_VERTEX_MAP.

Default -- There is no default file, only a default array (see SET_PIXEL_ARRAY).

Errors:
1. No view surface selected.
2. No segment open.
3. Columns <= 0.
4. Rows <= 0.
5. Invalid file name.

4.2.1.1.6 SET_PIXEL_ARRAY_EXTENT (X_LENGTH,Y_LENGTH)

This function specifies the size, in world coordinate units of the pixel array. X_LENGTH is the distance between element (1,1) and (COLUMNS,1) of the array, and Y_LENGTH is the distance between element (1,1) and (1,ROWS) of the array. The extent defines the proportions of the rectangular boundary for the entire image and it gives a convenient coordinate system for vertex mapping (see (SET_VERTEX_MAP).

The following defines the correspondence between the pixel map coordinate system and the column-row index of a pixel.

X_MAP = 0,0	COLUMN = 1
X_MAP = X_LENGTH	COLUMN = MAX_COLUMN
Y_MAP = 0,0	ROW = 1
Y_MAP = Y_LENGTH	ROW = MAX_ROW

see the figure on the next page.

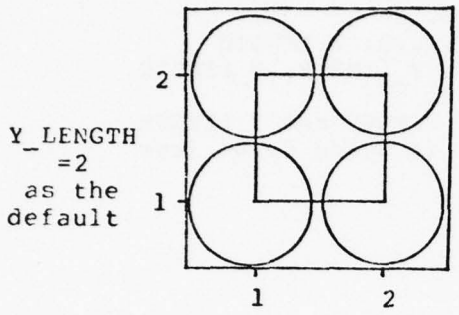
The pixel extent can also be viewed as a way of changing the aspect ratio of the pixel's of an image. It thereby allows the user to avoid distortion of an image that was originally generated with non-square pixels.

Default -- To facilitate working with an image made up of square pixels and to address each pixel directly the following defaults are used:

X_LENGTH = COLUMNS - 1.
Y_LENGTH = ROWS - 1.

That is, each pixel is assumed to cover a unit square in world coordinates.

The column and row values are set when a pixel array or file is defined or by the default pixel array.



Y_LENGTH =2 as the default

X_LENGTH / COLUMNS = horizontal pixel size

Y_LENGTH / ROWS = vertical pixel size

X_LENGTH = 2 as the default

Errors:
1. No view surface selected.
2. No segment is open.
3. X_LENGTH <= 0.0
4. Y_LENGTH <= 0.0

4.2.1.1.7 SET_VERTEX_MAP (VERTEX_INDEX_ARRAY,X_MAP_ARRAY, Y_MAP_ARRAY)

This function specifies the mapping between the image's pixel extent and a polygon specified in world coordinates (using the polygon or polygon mesh functions). This is accomplished by defining three vertices in world coordinates and three corresponding points on the image, in the coordinate system defined by the extent. Once the mapping is given, then it is possible to determine the transformation which makes the desired part of the image fit (and be displayed) within the polygon. The vertices in world coordinates are specified using an index into either: the X,Y,Z arrays of a following polygon, or the vertex index array of a POLYGON_EDGE_LIST (in this latter case two levels of indexing are used to specify a vertex). The three points on the image's pixel array extent are given by X and Y coordinates in pixel map space.

Default -- To facilitate working with rectangular painted polygons the following defaults are given:

```
VERTEX_INDEX_ARRAY -- 1, 2, 3
X_MAP_ARRAY        -- 0.0, 0.0, X_LENGTH
Y_MAP_ARRAY        -- 0.0, Y_LENGTH, Y_LENGTH
```

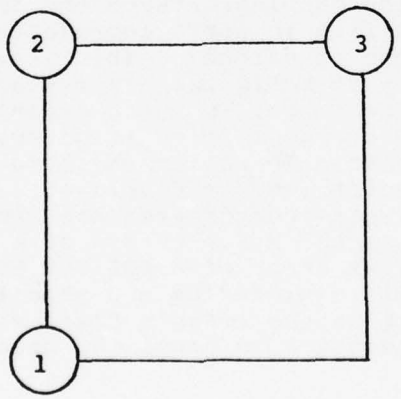
These defaults always exist, because X_LENGTH and Y_LENGTH always have either a default or a user assigned value (see SET_PIXEL_ARRAY_EXTENT).

Errors:
1. No view surface selected.
2. No segment open.
3. A VERTEX_INDEX <= 0.
4. A X_MAP value is < 0.0
5. A Y_MAP value is < 0.0

4.2.1.1.8 SET_CURRENT_POLYGON_MESH_SHAPE (SHAPE)

This function is used to specify the current shape of a polygon mesh being dealt with so that it can be handled efficiently.

There are three shapes of mesh each corresponding to a class of polyhedra. The first mesh shape is the nonclosed polyhedra or open mesh. The second shape is the closed concave polyhedral mesh. The third shape is the closed convex polyhedra.



The current polygon mesh shape in effect when the SET_POLYGON_ MESH command is invoked determines the shape attribute of that mesh.

The current polygon mesh shape in effect when the SET_POLYGON_MESH command is encountered, determines the shape attribute of that mesh.

Default -- The default shape is a nonclosed polyhedra, an open mesh.

Errors:
1. No view surface selected.
2. No segment open.
3. Invalid attribute value.

4.2.1.2 INTERROGATING CURRENT ATTRIBUTE VALUES FOR STATIC PRIMITIVE ATTRIBUTES

4.2.1.2.1 INQUIRE_CURRENT_POLYGON_TYPE (TYPE)

This function returns the current polygon type attribute.

Errors:
None.

4.2.1.2.2 INQUIRE_CURRENT_COLOR

INQUIRE_CURRENT_COLOR (R,G,B)

Returns the current color attribute in R,G,B. R,G,B are in the range of 0,1.

Errors:
1. No view surface is selected.
2. Type of the selected view surface is gray level rather than color.

INQUIRE_CURRENT_INTENSITY (INTENSITY)

Returns the current intensity attribute. INTENSITY range is between 0,1.

Errors:
1. No view surface is selected.
2. Type of the selected view surface is color rather than gray level.

INQUIRE_CURRENT_COLOR_INDEX (I)

Returns the index of the current color.

Errors:
1. No view surface is selected.
2. Type of the selected view surface
   is gray level rather than color.

INQUIRE_CURRENT_INTENSITY_INDEX (I)

Returns the index of the current intensity.

Errors:
1. No view surface is selected.
2. Type of the selected view surface
   is color rather than gray level.

4.2.1.2.3 INQUIRE_VERTEX_COLORS

INQUIRE_VERTEX_COLORS (R_ARRAY, G_ARRAY, B_ARRAY, N)
INQUIRE_VERTEX_INTENSITIES (INTENSITY_ARRAY, N)

This function returns the colors (intensities) of the vertices of the current shaded polygon set by the SET_VERTEX command. R,G,B's (intensities) are in the range of 0,1.

Errors:
1. No view surface is selected.
2. N < =2 .
3. Type of selected view surface is gray level rather than color, or vice versa.

INQUIRE_VERTEX_COLORS_BY_INDEX (INDEX_ARRAY, N)
INQUIRE_VERTEX_INTENSITIES_BY_INDEX (INDEX_ARRAY, N)

This function returns the indices of the colors (intensities) of the vertices of the current shaded polygon set by the SET_VERTEX command.

Errors:
1. No view surface is selected.
2. N < =2 .
3. Type of selected view surface is gray level rather than color, or vice versa.

4.2.1.2.4 INQUIRE_PIXEL_ARRAY (ACTIVE_FLAG,COLUMNS,ROWS,ARRAY)

This function returns the logical variable ACTIVE_FLAG. If it is set to true then the current pixel array and its dimensions are returned. If it is false then the pixel array will be stored in a file (see INQUIRE_PIXEL_FILE).

Errors:
None.

4.2.1.2.5 INQUIRE_PIXEL_FILE (ACTIVE_FLAG,COLUMNS,ROWS, FILE_NAME)

This function returns the logical variable ACTIVE_FLAG. If it is set to true then current file name and its dimensions are returned. If it is false then the pixel array will be stored in an array (see INQUIRE_PIXEL_ARRAY).

Errors:
None.

4.2.1.2.6 INQUIRE_PIXEL_ARRAY_EXTENT (X_LENGTH,Y_LENGTH)

This function returns the current pixel array extent information.

Errors:
None.

4.2.1.2.7 INQUIRE_VERTEX_MAP (VERTEX_INDEX_ARRAY,X_MAP,Y_MAP)

This function returns the current vertex map information.

Errors:
None.

4.2.1.2.8 INQUIRE_CURRENT_POLYGON_MESH_SHAPE (SHAPE)

This function returns the current polygon mesh shape attribute.

Errors:
None.

4.2.1.3 REDEFINING ATTRIBUTE VALUES FOR STATIC PRIMITIVE ATTRIBUTES

REDEFINE_COLOR (R,G,B , R',G',B')

All output primitives displayed on the selected view surface with color of R,G,B will be changed to have the color of R',G',B'. Both R,G,B and R',G',B' are rounded by the core system to the nearest color displayable on the currently selected view surface. If the current color attribute value is R,G,B, it is also changed to R',G',B'.

Errors:
1. No view surface is selected.
2. Invalid attribute value.
3. Type of the selected view surface is gray level rather than color.

REDEFINE_COLOR_INDEX (I,R,G,B) 1 <= I <= N

This function changes the color value whose index is I to R,G,B. The R,G,B is rounded by the core system to the nearest color displayable on the currently selected view surface. If the value of the current color attribute has the index of I, then the current color attribute is also changed.

Errors:
1. No view surface is selected.
2. Invalid color index.
3. Invalid color attribute value.
4. Type of the selected view surface is gray level rather than color.

REDEFINE_INTENSITY (INTENSITY, NEW_INTENSITY)

All output primitives displayed on the selected view surface with intensity of INTENSITY will be changed to have the an intensity of NEW_INTENSITY. Both INTENSITY and NEW_INTENSITY are rounded by the core system to the nearest intensity displayable on the currently selected view surface. If the current intensity attribute has a value of INTENSITY, it is also changed.

Errors:
1. No view surface is selected.
2. Invalid attribute value.
3. Type of the selected view surface is color rather than gray level.

REDEFINE_INTENSITY_INDEX (I, INTENSITY)

This function changes the intensity value whose index is I to INTENSITY. The INTENSITY is rounded by the core system to the nearest intensity displayable on the currently selected view surface. If the value of the current intensity attribute has the index of I, then the current intensity attribute is also changed.

Errors:
1. No view surface is selected.
2. Invalid intensity index.
3. Invalid intensity attribute value.
4. Type of the selected view surface is color rather than gray level.

7 CONTROL

7.2 FUNCTIONAL CAPABILITY

7.2.2 DEVICE SELECTION

7.2.2.2 SELECTING THE VIEW SURFACE

7.2.2.2.1 SELECT_VIEW_SURFACE (SURFACE_NAME,COLOR)

This function selects the logical view surface SURFACE_NAME for all subsequent graphic output, until the view surface is deselected with a DESELECT_VIEW_SURFACE functional call. The COLOR parameter specifies whether the view surface is used to display intensities (Black and White),or color (Red,Green ,Blue).

Errors:
1. The specified surface has not been initialized.
2. The specified surface is already selected.
3. The specified surface was not selected (because of resource limitations,for example).
4. A segment is open.
5. View surface is not capable of displaying color.
6. Invalid COLOR parameter

7.2.7 BATCHING OF UPDATES

7.2.7.2 ENDING A BATCH OF UPDATES

7.2.7.2.1 BATCHING OF UPDATES

END_BATCH_OF_UPDATE (ACTION)

This function denotes the end of a batch of changes to the picture. All changes to the picture that are specified prior to this function call are guaranteed to have taken place before control returns from this function. This function can be invoked at any time after a BEGIN_BATCH function call. ACTION is a logical variable indicating if a newframe action should be performed.

Errors:
1. There is no BEGIN_BATCH.

7.2.9 BACKGROUND COLOR OR INTENSITY

7.2.9.1 SETTING THE BACKGROUND

SET_BACKGROUND_COLOR (R,G,B)
SET_BACKGROUND_INTENSITY (INTENSITY)

The global background color (intensity) is set to R,G,B (INTENSITY). This action does not take effect until a newframe action occures.

Errors:
1. Attribute value out of range.

SET_BACKGROUND_COLOR_INDEX (INDEX)
SET_BACKGROUND_INTENSITY_INDEX (INDEX)

The global background color (intensity) is set to the color (intensity) with the index of INDEX. This action does not take effect until a newframe occures.

Errors:
1. Attribute index out of range.

7.2.9.2 INQUIRING WHAT THE BACKGROUND IS

INQUIRE_BACKGROUND_COLOR (R,G,B)
INQUIRE_BACKGROUND_INTENSITY (INTENSITY)

Returns the value of the global background color (intensity).

Errors:
None.

INQUIRE_BACKGROUND_COLOR_INDEX (INDEX)
INQUIRE_BACKGROUND_INTENSITY_INDEX (INDEX)

Returns the index of the global background color (intensity).

Errors:
None.

7.2.10 INQUIRING COLORS AND INTENSITIES

7.2.10.1 INQUIRING THE NUMBER OF COLORS OR INTENSITIES

INQUIRE_NUMBER_OF_COLORS (N)

Reports the number of colors which can be displayed simultaneously on the currently selected view surface..

Errors:
1. No view surface is selected.
2. Type of the selected view surface is gray level rather than color.

INQUIRE_NUMBER_OF_INTENSITIES (N)

Reports the number of intensities which can be displayed simultaneously on the currently selected view surface.

Errors:
1. No view surface is selected
2. Type of the selected view surface is color rather than gray level.

7.2.10.2 INQUIRING THE VALUE OF COLOR OR INTENSITY

INQUIRE_COLOR_VALUE (I,R,G,B) 1 <= I <= N

Reports the value of color I (in R,G,B) of the currently selected view surface. R,G,B are in the range of 0,1.

Errors:
1. No view surface is selected.
2. I is out of range (Less than 1 or greater than N).
3. Type of the selected view surface is gray level rather than color.

INQUIRE_INTENSITY_VALUE (I, INTENSITY) 1 <= I <= N

Reports the value of intensity I of the currently selected view surface. INTENSITY is in the range of 0,1.

Errors:
1. No view surface is selected.
2. I is out of range (Less than 1 or greater than N).
3. Type of the selected view surface is color rather than gray level.

7.2.11 INQUIRING GLOBAL COLORS AND INTENSITIES

7.2.11.1 INQUIRING THE NUMBER OF GLOBAL COLORS AND INTENSITIES

INQUIRE_NUMBER_OF_GLOBAL_COLORS (M) M >= N

This function returns the total number of colors which can be displayed on the currently selected view surface. It is not necessary that all the colors be simultaneously displayable.

Errors:
1. No view surface is selected.
2. Type of the selected view surface is gray level rather than color.

INQUIRE_NUMBER_OF_GLOBAL_INTENSITIES (M) M >= N

This function returns the total number of intensities wich can be displayed on the currently selected view surface. It is not necessary that all of the intensities be simultaneously displayable.

Errors:
1. No view surface is selected.
2. Type of selected view surface is color rather than gray level.

7.2.11.2 INQUIRING THE VALUE OF GLOBAL COLOR AND INTENSITY

INQUIRE_GLOBAL_COLOR (I,R,G,B) 1 <= I <= M

Returns the value of color I (in R,G,B) from the set of all colors (Global) for the currently selected view surface. R,G,B are in the range of 0,1.

Errors:
1. No view surface is selected.
2. I is out of range (Less than 1 or greater than M).
3. Type of the selected view surface is gray level rather than color.

INQUIRE_GLOBAL_INTENSITY (I, INTENSITY) 1 <= I <= M

Returns the value of intensity I from the set of all intensities (Global), for the currently selected view surface. INTENSITY is in the range of 0,1.

Errors:
1. No view surface is selected.
2. I is out of range (Less than 1 or

greater than M).
3. Type of the selected view surface is color rather than gray level.

7.2.12 DISPLAY MODE

7.2.12.1 SET_DISPLAY_MODE (MODE)

This function sets the display mode to: (1) Fast mode: Draw only the contour of the polygons or surrounding boundary of the painted polygon. No hidden surface or lines are removed. (2) Hidden line mode: Hidden lines are removed. Processing is done only when END_BATCH occures. (3) Hidden surface mode: Hidden surfaces are removed. Processing is done only when END_BATCH occures.

Errors:
1. System is not initialized.
2. No view surface is selected.
3. Invalid mode value.

7.2.12.2 INQUIRE_DISPLAY_MODE (MODE)

This function returns the display mode. (see SET_DISPLAY_MODE (MODE)).

Errors:
None.

7.2.13 PIXEL ARRAY FORMATTING

7.2.13.1 SET_PIXEL_ARRAY_FORMAT (BITS_PER_PIXEL,BITS_PER_PACKING_UNIT, PIXELS_PER_UNIT)

This function is used to specify the storage format in which a pixel array (or file) will be passed. The BITS_PER_PIXEL value tells how many bits are designated to represent each pixel. The BITS_PER_PACKING_UNIT gives the basic word length that the information will be placed in (it will usually be 8,12,or 16 bits). The PIXELS_PER_UNIT value tells how many pixel descriptions will be placed in each packing unit. The packing is assumed to be right justified.

Defaults -- There are none.

Errors:
1. Invalid attribute value.

7.2.12.2 INQUIRE_PIXEL_ARRAY_FORMAT (SET_FLAG,BITS_PER_PIXEL,
BITS_PER_PACKING_UNIT,PIXELS_PER_UNIT)

This function returns the logical variable SET_FLAG that tells if the pixel array format has been set. If it is set to true then the current pixel array format information is returned.

Errors:
None.

10.2.1 WRITING A VIEW SURFACE PIXEL

WRITE_VIEW_SURFACE_PIXEL_INTENSITY (X, Y, INTENSITY)
WRITE_VIEW_SURFACE_PIXEL_COLOR (X,Y,R,G,B)

The intensity or color of the pixel on the currently selected view surface addressed by X,Y (in normalized device coordinates) overwrites the INTENSITY or R,G,B. The INTENSITY or R,G,B is rounded by the core system to the nearest intensity or color displayable on the currently selected view surface. INTENSITY and R,G,B are each between 0,1.

Errors:
1. No view surface is selected.
2. No segment is open.
3. X,Y out of range.
4. Intensity or color attribute out of range.
5. Type of the selected view surface is gray level rather than color, or vice versa.

WRITE_VIEW_SURFACE_PIXEL_INTENSITY_BY_INDEX (X,Y, INDEX)
WRITE_VIEW_SURFACE_PIXEL_COLOR_BY_INDEX (X,Y, INDEX)

The intensity or color of the pixel on the view surface addressed by X,Y (in normalized device coordinate) overwrites the intensity or color referenced by index INDEX.

Errors:
1. No view surface is selected.
2. No segment is open.
3. X or Y out of range.
4. INDEX out of range.
5. Type of selected view surface is gray level rather than color, or vice versa.

10.2.2 READING A VIEW SURFACE PIXEL

READ_VIEW_SURFACE_PIXEL_INTENSITY (X,Y,INTENSITY)
READ_VIEW_SURFACE_PIXEL_COLOR (X,Y,R,G,B)

Reports the intensity or color in Red, Green, and Blue components, of the Pixel addressed by X,Y (in normalized device coordinates) on the currently selected view surface.

Errors:
1. No view surface is selected.
2. X or Y out of range.
3. Type of the selected view surface is gray level rather than color, or vice versa.

READ_VIEW_SURFACE_PIXEL_INTENSITY_BY_INDEX (X,Y,INDEX)
READ_VIEW_SURFACE_PIXEL_COLOR_BY_INDEX (X,Y,INDEX)

Reports the intensity or color index of the pixel addressed by X,Y (in normalized device coordinate) on the currently selected view surface.

Errors:
1. No view surface is selected.
2. X or Y out of range.
3. Type of the selected view surface is gray level rather than color, or vice versa.

10.2.3 WRITING A GROUP OF VIEW SURFACE PIXELS

WRITE_VIEW_SURFACE_PIXELS_INTENSITY_BY_INDEX
(XMIN,XMAX,YMIN,YMAX)
WRITE_VIEW_SURFACE_PIXELS_COLOR_BY_INDEX
(XMIN,XMAX,YMIN,YMAX)

Writes the intensity or color of pixels from the current pixel array into the specified region of the selected view surface,using the pixel array packing format. If the array is a file, then the file is overwritten into the specified portion of the view surface. The size of the current pixel array (xmax-xmin and ymax-ymin) must be the same as the size (xmax-xmin and ymax-ymin) of the specified portion of the view surface. Current pixel array can be set by calling the SET_ CURRENT_PIXEL_ARRAY function. Intensities or colors are specified by their indices.

Errors:
1. No view surface is selected.
2. Xmin >= Xmax.
3. Ymin >= Ymax.
4. Xmin,Xmax,Ymin or Ymax out of range.
5. The current pixel array is too large or too small.
6. Type of the selected view surface is gray level rather than color, or vice versa.

10.2.4 READING A GROUP OF VIEW SURFACE PIXELS

READ_VIEW_SURFACE_PIXELS_INTENSITY_BY_INDEX
(XMIN,XMAX,YMIN,YMAX)
READ_VIEW_SURFACE_PIXELS_COLOR_BY_INDEX
(XMIN,XMAX,YMIN,YMAX)

Reads the intensity or color of the specified region of the selected view surface, into the current pixel array, using the pixel array packing format specified by SET_PIXEL_ARRAY_FORMAT. If the array is a file then the specified portion of the view surface is written into the file. The size of the current pixel array (xmax-xmin and ymax-ymin) must be the same size (xmax-xmin and ymax-ymin) of the part of the view surface being interrogated. This array is available by calling the INQUIRE_CURRENT_PIXEL_ARRAY. Intensities or colors are specified by their indices.

Errors:
1. No view surface is selected.
2. Xmin >= Xmax.
3. Ymin >= Ymax.
4. Xmin,Xmax,Ymin or Ymax out of range.
5. The current pixel array is too large or too small.
6. Type of the selected view surface is gray level rather than color, or vice versa.

10.2.5 SET_VIEW_SURFACE_OVERLAY_MODE (MODE)

This function specifies how a new pixel will affect the old pixel on the view surface. A new pixel is one which is sent by either a SET_VIEW_SURFACE_PIXEL or in a SET_VIEW_SUFFACE_PIXELS function. An old pixel is one which already exists on the view surface.

The particular way the new pixel combines with the old is controlled by the MODE. A partial list of modes is: REPLACE, AND, OR, EXCLUSIVE OR, ADD, SUBTRACT, ...

Default -- The default mode is REPLACE.

Errors:
1. No view surface is selected.
2. No segment is open.
3. Nonraster view surface is selected.
4. Invalid mode value.

10.2.6 INQUIRE_VIEW_SURFACE_OVERLAY_MODE (MODE)

This function returns the current vaue of the overlay mode, which either has the initial default value or has been set by SET_VIEW_SURFACE_OVERLAY_MODE.

Errors:
None.

PART IV: MODIFICATIONS TO EXTEND A VECTOR GRAPHICS SYSTEM TO A RASTER GRAPHICS SYSTEM

What follows is a list of modifications and additions that we are making to the original line drawing Core Graphics System to allow it to handle raster graphics. An explanation of the concepts and functions needed is in the paper "Raster Graphics Extensions to the Core Graphic System", and in the reference manual for those extensions.

The list given here outlines the software needed to support this expansion. The Core System at George Washington University is being expanded in this way, so many features have already been implemented. Those features that have not yet been implemented are marked by an astrisk.

We provide this list with the hope that it will be useful to others who undertake similar efforts either with the Core System or with other graphics packages.

1. Device Driver Extensions
    1. A software scan line converter, if no hardware scan converter is available.
    2. Add color and intensity levels
    3. A way of specifying if color or gray scale is being used
    4. Allow the Device Driver to accept 3D normalized device coordinates.
    5. Allow the Device Driver to deal with a modularized Pseudo Display File (see section 3)
    6. A polygon scan converter
    *7. A more direct way to output a pixel array to the screen.

2. General Extensions

   1. Add a way to set the background color or intensity
   2. Add a new set of display modes:
      a. Fast Mode
      b. Hidden
      c. Hidden Line Removal Mode
   3. Create a general subroutine to control where information should be sent (Device Driver or Pseudo Display File). This was added to support the three display modes and to further structure the code.
   *4. Either a simple 2½D depth priority sort or a complete Hidden Surface Removal Algorithm.
   *5. Add direct pixel (view surface) operations

3. Pseudo Display File (PDF)

   1. Add a Z component to each vertex coordinate in the PDF. This is required for the Hidden Surface Removal Algorithm (HSA) in Normalized Device Coordinate (NDC) space.
   2. Add opcodes for the Polygon and its attributes
   3. Modularize the PDF. This means that all the coordinate information for an output primitive (Line, Text, Polygon) is grouped together, when stored in PDF. For example, a line definition would consist of the following in the PDF: the opcode and the coordinate of both vertex coordinates. Before the PDF was made modular, a line definition would only contain one raster coordinate, the other vertex would be found in either a move, mark, or line definition made before the lines endpoint was given. This made the task of any Hidden Line/Surface Removal difficult, because a search had to be made to find the starting point of the line.
   *4. The data structure of the PDF could be further developed to support Hidden Surface Removal Algorithm. The specific way of structuring the PDF will depend on the type of Hidden Surface Algorithm that is being used.
   *5. Allow the PDF to reference pixel array files. Complex images require very large pixel arrays (512X512), therefore it is important to avoid storing them inside of the PDF. The pixel array should instead be stored in a separate file, and the file reference information placed in the PDF.

4. Viewing Primitives

   1. Pass the Z components along with the X,Y coordinates for output primitives (line,text,etc.) to the PDF.
   2. Make 3D windows (truncated pyramids for perspective views) map into rectangular boxes in 3D Normalized Device Coordinates. This must be done so that a line in the world coordinates remains a straight line in NDC space.

3. Send all the vertex coordinates that describe a primitive (line, text, etc.) in a group to allow a modularized PDF to be constructed.
4. Add the basic polygon primitives and the required support:
   a. Add the user output primitive calls.
   b. Apply the viewing transformations for the vertices of a polygon.
   c. Calculate additional information like the minimum Z coordinate and the equation of the plane containing the polygon. This is passed to the PDF to make the hidden surface algorithm more efficient.

*5. Add a polygon clipping algorithm.
6. Painted Polygon extensions:
   a. A new set of polygon primitive attributes.
   *b. A method for mapping a section of an image onto a polygon in the world.
   *c. The translation, scaling, and rotation of images
   *d. An averaging or convolution algorithm used to maintain the appearance of an image (minimize the distortion caused by the above transformations)
7. Shaded Polygon extensions:
   *a. The SET_VERTEX_COLORS attributes
   *b. The color interpolation algorithm for GOURAND shading

4. Mesh Extensions

   *a. The mesh vertex list - primitive
   *b. The POLYGON_EDGE_LIST subroutine.

5. Attributes

   1. Extensions of the color and intensity specifications
      a. Color/Intensity defined by index
      *b. Color/Intensity defined by value (Red, Green, Blue)
      *c. A method for finding the best color fit for a particular device from an R,G,B specification.
      *d. A set of functions for redefining colors referenced by index
      *e. Extend the color/intensity inquiry capabilities.
   2. Add the Polygon type attribute
   3. The Mesh shape attribute
   4. A way of handling pixel arrays and files
   *5. Painted Polygon attributes
   *6. Shaded Polygon attributes

REFERENCES

AMG75 TV Programmer's Manual, the Architecture Machine Group, MIT, 1975.

AMG76 85 Micro Calls, The Architecture Machine Group, MIT, 1976.

AMG77 An Extention to the Magic Graphics Software, The Architecture Machine Group, MIT, 1977.

FULT75 Fulton, D.L. and Duquet, R.T. "List Processing Primitives for BASIC," Computers and Graphics 1 (2/3), September 1975, pp. 203-210.

GSPC77 Status Report of the Graphic Standards Planning Committee of ACM/SIGGRAPH. Published as Computer Graphics 11 (3), Fall 1977.

OBRI75 O'Brien, C.D. and Brown, H.H. "Image: a language for the Interactive Manipulation of a Graphics Environment," Computer Graphics 9 (1), Spring 1975, pp. 53-60. (Proceedings of SIGGRAPH '75 Annual Conference).

REEV77 Reeves, V. GAPC User's Manual, University of Toronto, January 1977.

SPRO75 Sproull, R. and Newman, W. "The Design of Gray-Scale Graphics Software," Conference on Computer Graphics, Pattern Recognition and Data Structures, IEEE, 1975, 18-20.

SUTH74a Sutherland, I., Sproull, R., and Schumacker, R. "A Characterization of Ten Hidden-Surface Algorithms," Computing Surveys 6, 1 (March 1974), 1-56.

SUTH74b Sutherland, I. and Hodgman, G. "Reentrant Polygon Clipping," CACM 17, 1 (January 1974), 32-38.

VAND78 van Dam, A. and Newman, W., "The Development of a Graphics Standard," Computing Surveys 10, 4 (December 1978), uvw-xyz.

In accordance with letter from DAEN-RDC, DAEN-ASI dated 22 July 1977, Subject: Facsimile Catalog Cards for Laboratory Technical Publications, a facsimile catalog card in Library of Congress MARC format is reproduced below.

Foley, James D
Raster graphics extensions to the Graphics Compatibility System (GCS) / by James D. Foley, James Templeton, Dara Dastylar, Department of Electrical Engineering and Computer Science, George Washington University, Washington, D. C. Vicksburg, Miss. : U. S. Waterways Experiment Station ; Springfield, Va. : available from National Technical Information Service, 1979.
76 p. : ill. ; 27 cm. (Miscellaneous paper - U. S. Army Engineer Waterways Experiment Station ; O-79-3)
Prepared for Office, Chief of Engineers, U. S. Army, Washington, D. C., under Contract No. DACA39-78-M-0073.
References: p. 76.

1. Computer graphics. 2. Graphic arts. 3. Graphics Compatibility System. 4. Raster graphics. I. Templeton, James, joint author. II. Dastylar, Dara, joint author. III. George Washington University, Washington, D. C. Dept. of Electrical Engineering and Computer Science. IV. United States. Army. Corps of Engineers. V. Series: United States. Waterways Experiment Station, Vicksburg, Miss. Miscellaneous paper ; O-79-3.
TA7.W34m no.O-79-3

WATERWAYS EXPERIMENT STATION REPORTS PUBLISHED ON THE GRAPHICS COMPATIBILITY SYSTEM

	Title	Date
Miscellaneous Paper O-79-1	Host-Computer Implementation Guidelines for the Three-Dimensional Graphics Compatibility System (GCS)	Mar 1979
Miscellaneous Paper O-79-2	Device Implementation Guidelines for the Three-Dimensional Graphics Compatibility System (GCS)	Mar 1979
Miscellaneous Paper O-79-3	Raster Graphics Extensions to the Graphics Compatibility System (GCS)	Mar 1979