# Advanced Graphics Package

# User Guide


HEWLETT PACKARD

HEWLETT-PACKARD COMPANY
Engineering Productivity Division
11000 Wolfe Road
Cupertino, California 95014

MANUAL PART NO. 97085-90000
Printed in U.S.A. June 1983
E0683

# Printing History

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates. However, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

First Edition June 1983

# PREFACE

*Who needs to use this manual?*

The AGP User Guide is intended for users of Hewlett-Packard's Advanced Graphics Package (AGP). Graphics application designers will appreciate it as a general overview of AGP's capabilities. Application programmers will use it as they write their first graphics programs, enjoying its conceptual descriptions and programming examples. Operators of graphics applications who want to better understand the system they use may read the first two chapters of this manual.

*What does it cover?*

The AGP User Guide provides a tutorial introduction to AGP. It teaches the new user by introducing basic graphics concepts and AGP's implementation of them. This manual does not attempt to describe all details of the AGP system, as this would impair the learning process.

Three other manuals, the *AGP Reference Manual* ,the *Device Handlers Manual*, and the *AGP Supplement for HP-UX Systems* will be of interest to AGP users. The *AGP Reference Manual* provides additional product documentation. As more sophisticated applications are addressed, it will be relied on for complete descriptions beyond the introductory level of the User's Guide. The *Device Handlers Manual* has a different purpose. It documents details of the interaction between AGP and the graphics devices being used. For information on AGP calls which are operating system dependent, refer to the *AGP Supplement for HP-UX Systems*.

*What does it assume?*

Very little is assumed by the AGP User Guide. An overall understanding of basic concepts of computers (e.g., programs, peripherals, etc.) is necessary. Some familiarity with computer graphics is helpful, but a thorough reading and understanding of Chapter 2 will provide it. Please refer to the demo for examples of how to program using the AGP Subroutines. Also, knowledge of the FORTRAN programming language is desirable, since that is the language in which examples are written. Experience with additional host languages, such as Pascal, would complement the FORTRAN knowledge and allow those languages to be used for graphics production. (See the *AGP System Supplement*.)

# HP Computer Museum
# www.hpmuseum.net

**For research and education purposes only.**

*How is it organized?*

This manual is organized as a tutorial. To benefit from the tutorial approach, the new user should read the chapters in the order in which they are presented.

The chapters are:

Chapter 1 An overview of the AGP product. Graphics production, features, and host language considerations are covered in this chapter.

Chapter 2 General concepts of computer graphics. An overview of AGP's six functional areas is presented. The remaining chapters treat these areas in detail.

Chapter 3 Output Primitives. The use and attributes of the fundamental elements of a graphics picture is the subject of this chapter.

Chapter 4 Viewing Transformations. The process of producing graphics images from two or three-dimensional objects is explained.

Chapter 5 Segments. The ability to subdivide complex pictures into separate logical entities is discussed.

Chapter 6 Input. The steps for receiving graphics input from different types of devices are presented.

Chapter 7 Control. The remaining functions, necessary to control the graphics environment, are the subject of the final chapter.

On the cover: "Wright Brothers Flyer, Standard Wright Model A 1908", Kuohsiang Chen and Charles L. Owen, Design Processes Laboratory, Institute of Design, Illinois Institute of Technology.

# Table of Contents

**Chapter 4 Viewing Transformations**



**Chapter 5 Segments and Primitives Outside of Segments**

# List Of Figures

# List of Tables

# Chapter 1
# Overview

## GENERAL

A computer graphics package is a tool to help computer users utilize graphics devices. Graphics devices are hardware peripherals that draw lines and characters in response to commands from an application program, producing pictures based on the information they are sent. A computer graphics tool package is a software layer between the application program and the graphics device that makes the application program easier to write and maintain.

The Advanced Graphics Package (AGP) is a computer graphics package. AGP provides the application program with simplified access to the devices necessary to produce computer graphics, assuming much of the overhead associated with interactive graphics creation.

It relieves the program of the burden of using commands and units specific to a device. This idea, known as *device independence*, means that the internal logic of the application program may remain unaffected by the type of graphics device being used. Device independence is desirable for several reasons. It makes the application program easier to write initially, easier to maintain over time, easier to enhance through the addition of new peripherals, and easier to transport from one system to another. With these features, AGP makes the advantages of computer graphics available while simplifying its utilization.

## DIVISION OF THE GRAPHICS TASK

With AGP, application programs describe *objects* in a graphical world. Objects are specified in the *world coordinate system*, a three-dimensional Cartesian coordinate system. The world coordinate system's units are the units of the application; that is, the object can be described in any units that the application requires.

In general, objects are three-dimensional, though the nature of some applications may make the use of the third dimension unnecessary. These applications may choose to ignore the depth of objects by implicitly working in the X Y plane.

AGP produces *images* from objects. The image of an object is the flat representation of it on a graphics *display device*. Television produces images on a television set's screen, while AGP produces images on a display device.

Images are produced from objects in two steps, using the *virtual coordinate system*. Virtual coordinates are two-dimensional and range between 0.0 and 1.0. They allow an object to be translated into units that are neither application-dependent nor device-dependent.

As shown in Figure 1-1, the two steps in the production of images from objects are:

1. Application data represented in the world coordinate system is translated to virtual coordinates by projection from three dimensions to two dimensions.

2. Virtual coordinates are translated to the units and commands needed by individual display devices.

AGP is broken into two separate programs to reflect this division of processes. Each program is loaded individually. Together they provide a high-level interface to the graphics devices being used.

## User Program

Generating computer graphics with AGP requires writing a *User Program* (UP). The user program is an application program containing calls to AGP routines. The parameters to these routines control the graphics production process, allowing information to be passed both to and from AGP. Routines referenced by the user program are appended to it by searching the library UPLIB (User Program LIBrary) when the user program is loaded.

Data in the user program is represented in terms of the world coordinate system. The AGP routines appended to the user program generate virtual coordinate data from the user's world coordinate data. This virtual coordinate data is then transmitted to a second program, called the work station program via interprocess communication.

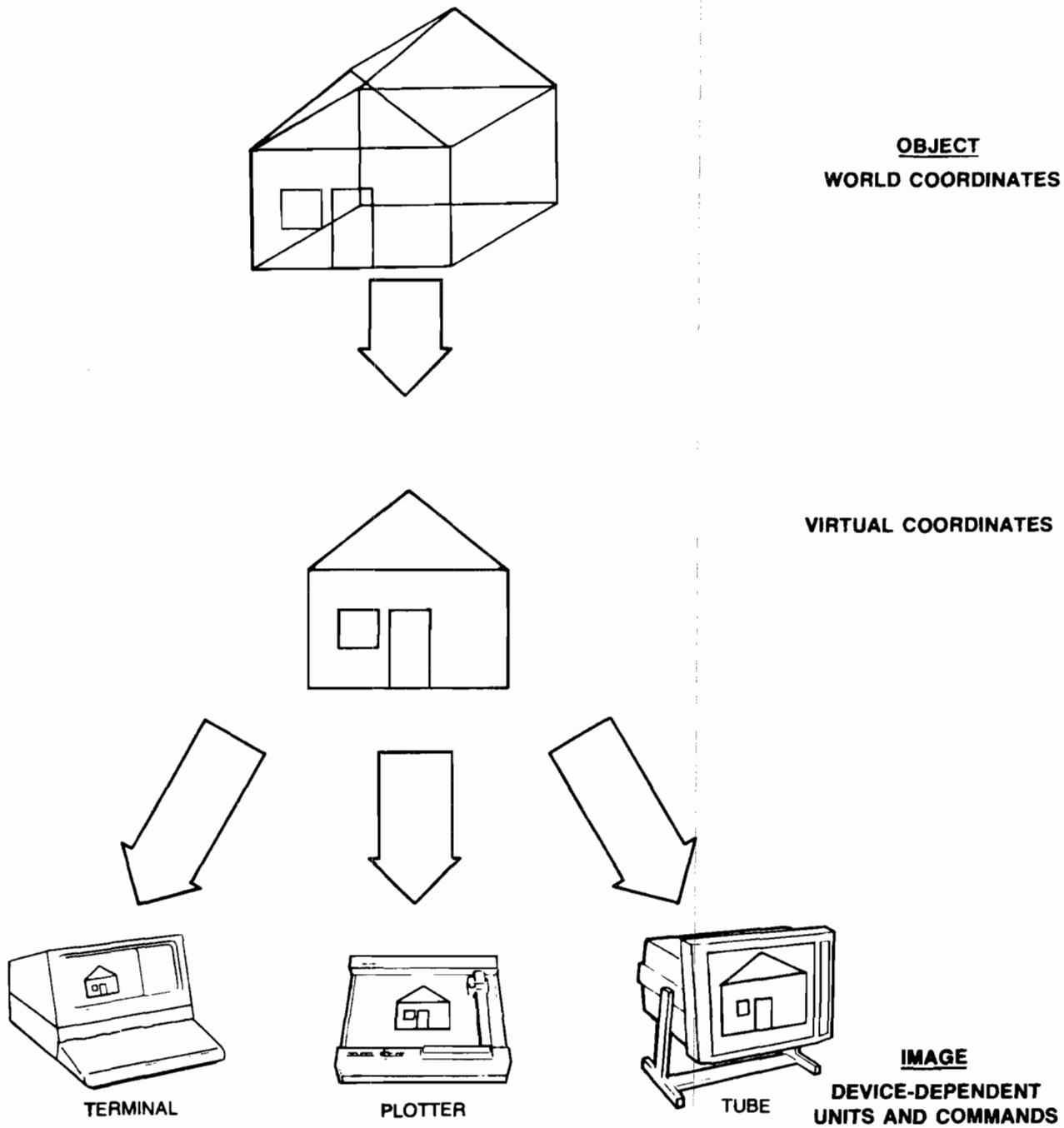


**Figure 1-1. Images are Produced From Objects**

## Work Station Program

Each *Work Station Program* (WSP) controls a graphics *work station*. A work station is a collection of devices that are treated as an identifiable unit. Every work station has its own work station program. Unlike the user program, the work station program is not written by the user, but is configured and loaded with standard parts provided in AGP.

Each work station program knows how to translate device-independent virtual coordinate data from a user program into device-dependent commands for a particular device. A work station may include the following logical devices (a logical device is the abstraction of a physical device):

1. graphics display
2. alphanumeric display
3. button input
4. keyboard input
5. locator input
6. valuator input
7. pick input

Two types of output devices are available: graphics and alphanumeric. Of the input devices, a button device returns an integer, a keyboard device returns alphanumeric text, a locator returns a real coordinate pair, a valuator returns a single real value and a pick device identifies a displayed image. A work station may have one device of each logical type.

The portion of a work station program that converts virtual coordinate data to device-dependent commands is known as a device handler. A device handler controls a single device on a work station; for example, one is used for the HP 2623A Graphics Terminal's keyboard. A second device handler would be necessary for its screen. The device handlers in the work station program are part of DGL (Device-independent Graphics Library). AGP relies on DGL for this service. See Figure 1-2 for the relationship between AGP and DGL.

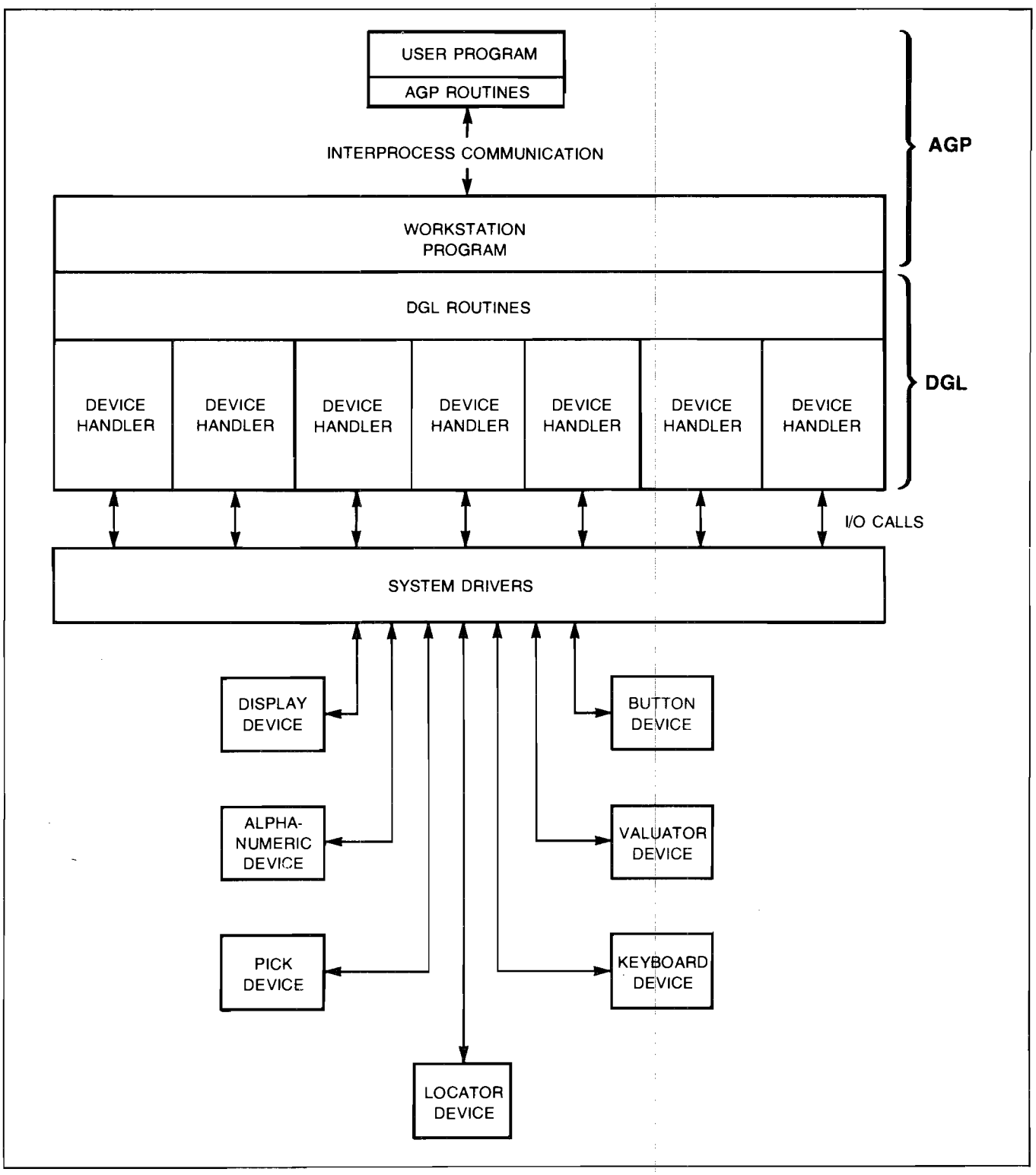


**Figure 1-2. Relationship Between AGP and DGL**

Throughout this manual, certain features will be described as "device-dependent", meaning they are a function of the device handlers provided by DGL and of capabilities of the particular devices being used. The *Device Handlers Manual* should be referred to in these cases. For example, the devices supported by DGL are exactly those devices that AGP may use. The *Device Handlers Manual* provides a list of these devices, as well as device-dependent information affecting AGP.

## Advantages of the Division

The graphics system is divided into two separate programs for a number of reasons. First, the user program is device-independent. Once written, it can control any supported graphics device by choosing the appropriate work station program when the user program is executed.

A second advantage is that the user program can control more than one work station concurrently. The same output is directed to each without having to duplicate portions of user program code. For example, both a terminal version of a graphics image and a hard copy on a plotter could be produced simultaneously. Two work stations would be initialized before the two images are output, depending on the operating system. As many as eight could be chosen by initializing the appropriate work station programs.

Since much of the AGP code necessary to perform the graphics task is in the work station program, less of the user program's address space is needed for the AGP support routines. Therefore, another advantage of the two program approach is that more logical address space is available to the user.

Finally, loading procedures are simplified and the time to load a graphics program is reduced by dividing the graphics task. Standard work station programs need only be loaded once when AGP is installed in the system. User programs are loaded more frequently, but since they contain only a part of the code necessary to produce graphics, the loader will have less work to do and so will finish more quickly.

# Chapter 2
# Concepts of Computer Graphics

## INTRODUCTION

Computer graphics is an important step toward using computers as a communications tool, rather than just a computation device. Through graphics, information is transmitted in picture form, instead of as alphanumeric symbols whose meaning may not be immediately apparent. The graphical picture can be understood at a glance, saving the viewer from tediously analyzing tabular data. In this way, ideas rather than letters and numbers can be communicated quickly and efficiently, increasing the productivity of the communication process.

The following sections describe six major functional areas of AGP, which would be present in one form or another in almost any high level graphics product. The concepts and examples introduced here will be developed in detail in the following chapters.

## GRAPHICS OUTPUT PRIMITIVES

*Graphics output primitives* are the building blocks of objects. Just as an algorithm must be broken down into the simplest possible instructions to create a computer program, a graphically created object must be reduced to output primitives. AGP uses six types of output primitives: *moves*, *lines*, *polylines*, *polygon sets*, *text*, and *markers*. Lines and text are familiar concepts. A move is just that, a move from one location to another. A polyline is a sequence of connected lines; markers are data symbols, such as diamonds or asterisks. A polygon set is a group of associated polygons, each of which is a closed plane figure bounded by straight lines. In Figure 2-1, markers indicate data points on a graph, while a series of line primitives are used to connect the data points.

The location of each output primitive is specified in the world coordinate system, with a point denoted by its X, Y, and Z coordinates. The graphics software "remembers" the most recently specified point, known as the *current position*. A line can then be created by specifying another point, which AGP connects to the first. This is known as a "draw." The end of that line then becomes the current position, and another line can begin there. The current position can also be moved to another point without drawing a line; this is known as a "move." A picture can be drawn by successive moves and draws.

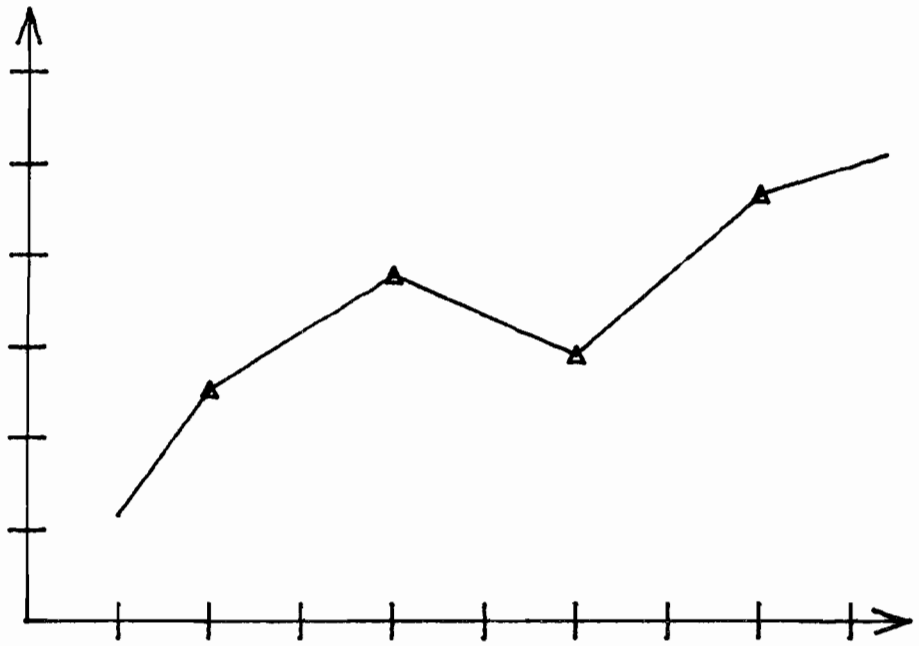

**Figure 2-1. Output Primitive Examples**

The polyline primitive makes it possible to draw a series of connected lines all at one time, instead of making a series of draws. This is done by specifying the number of points to be drawn and the X, Y, and Z arrays of the coordinates to be connected.

The polygon-set primitive allows the user to draw a series of connected lines to form closed figures (polygons) in the world coordinate system (see Figure 2-2). Like polylines, the user specifies the number of points to be drawn and the X, Y, and sometimes Z arrays of the coordinates to be connected. Unlike polylines, the polygon set primitive is planar and allows the user to specify the style and color of the fill pattern for the interior of these bounded regions.

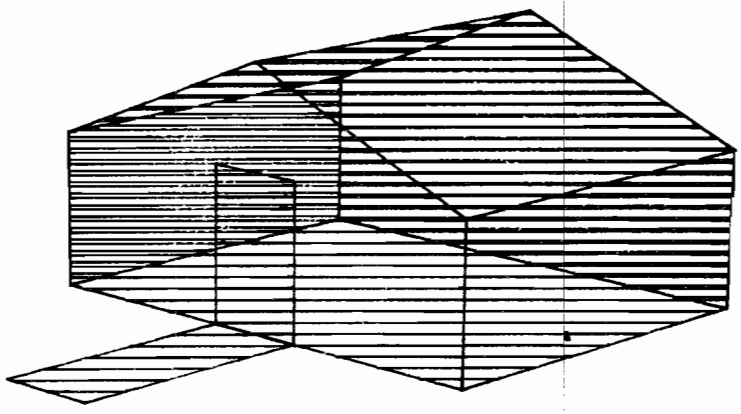

**Figure 2-2. A House Created With Polygon Set Calls**

Text can be of high, medium, or low quality. High quality text is software-generated from a series of line primitives and can appear in a variety of different fonts. Medium and low quality text are usually generated in hardware. Since display devices typically cannot produce all sizes and orientations of text, the appearance of medium and low quality text conforms less fully to the requirements of an application program than high quality text does. Each character in a medium quality text string is placed individually, while a low quality string is placed as a unit. Thus only the starting point of a low quality text string is under the application program's control (see Figure 2-3).

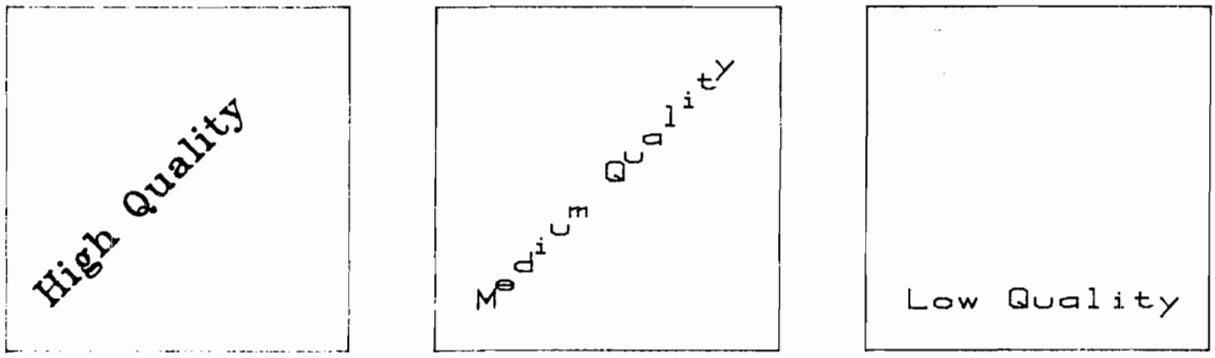


**Figure 2-3. High, Medium, and Low Quality Text**

With the output primitives of line, polyline, polygon set, text, and marker, and the ability to move the current position, complex graphical pictures can be constructed. Figure 2-4 shows the Gossamer Condor, a particularly interesting graphics figure consisting of nothing more than line primitives.

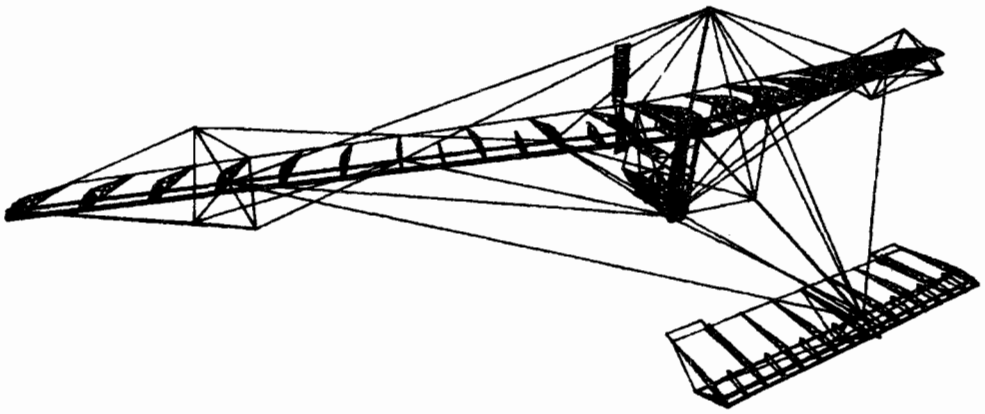

**Figure 2-4. Complex Figures Can Be Drawn**

(Courtesy of Dr. Paul MacCready, President of AeroVironment, Inc., designer of the Gossamer Condor. Data created by Paul Lionikis and Prof. Charles L. Owen, Institute of Design, Illinois Institute of Technology.)

# ATTRIBUTES

*Attributes* are the general characteristics of graphical pictures. Output primitive attributes affect the appearance of individual primitives. For example, the attributes of a line primitive are color, linewidth, linestyle, and pick-ID. Different values of the linestyle attribute might produce the various styles shown in Figure 2-5.

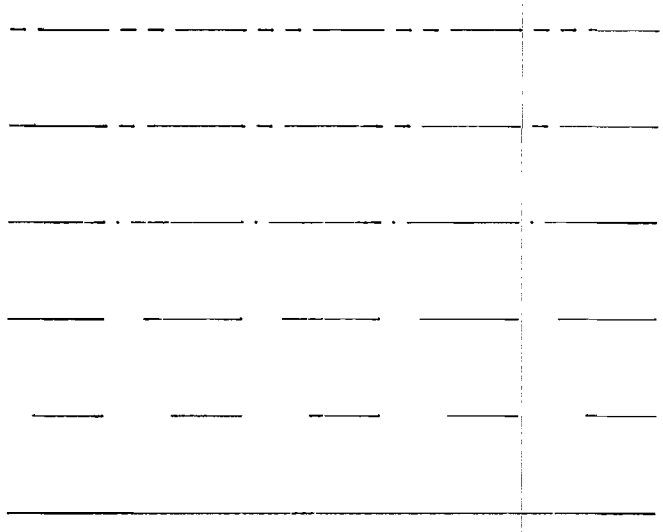

**Figure 2-5. An Output Primitive Attribute: Linestyle**

Polygon set attributes include polygon style, interior color, and interior linestyle. *Polygon style* is composed of four factors: interior density, interior orientation, edge display/non-display, and device independence/dependence. *Interior color* defines the color of the polygon set's interior fill and *interior linestyle* specifies the form (solid, dashed, dotted, etc.) of interior fill lines.

Text appearance can also be controlled with primitive attributes. Low quality text has the attributes of size, justification, color, linewidth, and linestyle. Medium quality text has these attributes as well as gap and orientation. High quality text has all of these, as well as font and slant. Figure 2-6 shows examples of Eurostyle, Simplex Roman, Triplex Roman, Script, Mathematical, and Gothic text fonts.


**Figure 2-6. Six Fonts of High Quality Text**

## VIEWING TRANSFORMATIONS

As objects are created, they are transformed for viewing. A *viewing transformation* converts two or three-dimensional objects into two-dimensional images. The direction and distance from which an object is seen, the portion of it to be displayed, and the location of the object's image on the display device are all chosen by the user.

Renaissance artists practiced this act of transforming objects for viewing, calling it "perspective". Though television is now commonplace, the first television viewers were astounded to see their world displayed on a flat screen. Today, a camera accomplishes in a moment what took an artist years to master: portraying three dimensions in two.

AGP is capable of both two and three-dimensional viewing transformations. Two-dimensional viewing is a subset of three-dimensional viewing, where the Z axis of the world coordinate system is ignored. Since the two-dimensional transformation is the simpler case, it is discussed first.

## Two Dimensions

In defining the two-dimensional viewing transformation, three questions need to be addressed:

1. What portion of the world coordinate system will be displayed (where is the *window*)?
2. Where will the window be mapped in the virtual coordinate system (where is the *viewport*)?
3. Where will the virtual coordinate system be placed on the display device (what are the *logical display limits*)?

Setting the window can be thought of as taking a large sheet of paper and cutting a rectangular hole in it, then placing the piece of paper over the object of interest. Only what is inside the hole, or window, will be visible. In AGP, the window is specified as a rectangular area of the world coordinate system. Any portion of an output primitive lying inside the window is displayable; any portion outside is not. In some cases, the window may be large enough to see an entire object; in others, it may be more desirable to see part of the object while not showing the remainder. Figure 2-7 shows the effect of two different windows on the same object.

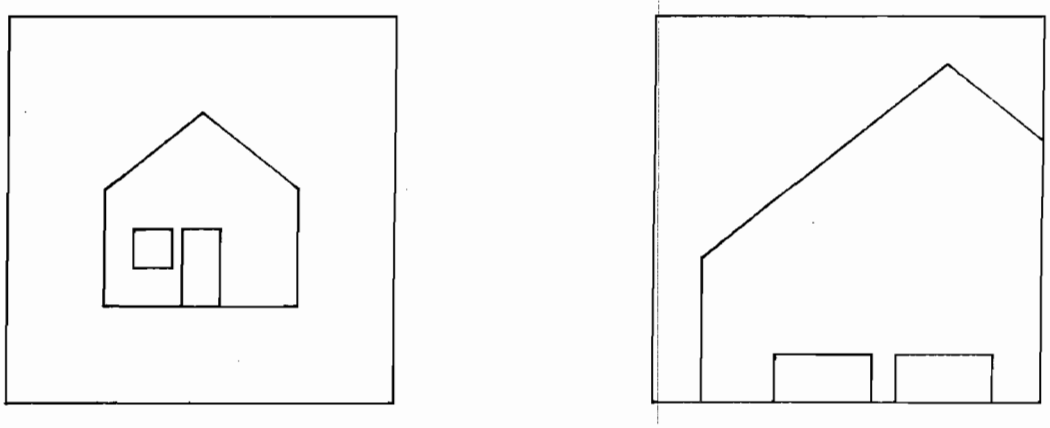

**Figure 2-7. Two Windows On the Same Object**

The area on a view surface where images appear is called a viewport. The dimensions of the viewport are set in the virtual coordinate system, which is the two-dimensional system whose units range from 0.0 to 1.0. Frequently, the entire view surface will be the viewport, or multiple viewports can be defined sequentially so that several images can be displayed together. Figure 2-8 shows such an example. On an HP Graphics Terminal, a left and right viewport have been set, allowing one image to be displayed on the left and another on the right.

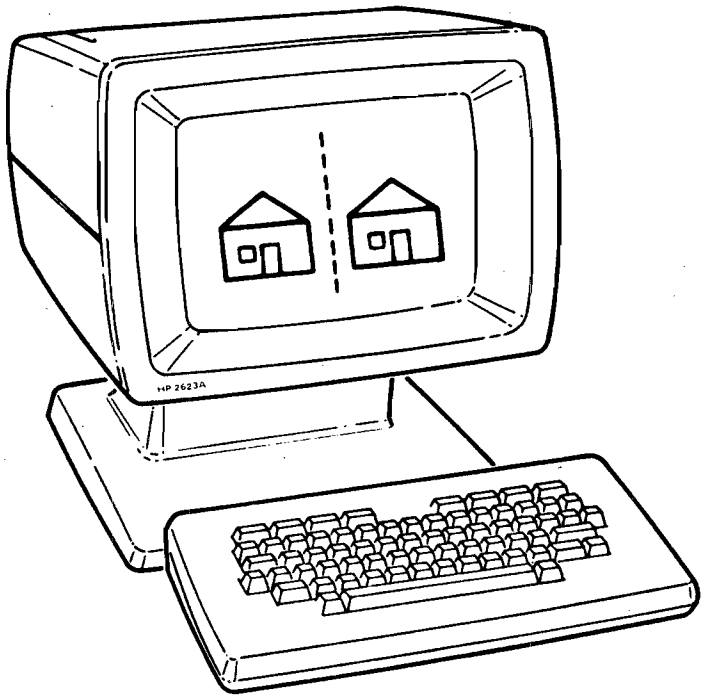


**Figure 2-8. Two Viewports, One Object**

Once the window and viewport have been defined, output primitives within the window are mapped into the viewport so that each point in the window corresponds to a point in the viewport.

To offer full flexibility in accessing display devices of different sizes and shapes, AGP lets the application program control the mapping of the virtual coordinate system to the display device. The logical display limits can be manipulated to alter the absolute size of the picture that is produced. These limits bound the *logical display surface*, perhaps making it the entire display device, or often defining some subset of it.

The adjective "logical" distinguishes this surface from the surface of the device itself, sometimes called the physical surface.

The steps in the mapping of objects in the window to images on a display device's logical display surface are somewhat complex. They are described more fully in Chapter 4, where the AGP routines that control this mapping are presented.

## Three Dimensions

Three-dimensional viewing is more complex than two-dimensional viewing because it can be done from more than one direction. Three-dimensional viewing transformations are very powerful in nature. With relatively little application code, displayed images can be given a variety of perspectives and appearances.

AGP mathematically determines the appearance of three-dimensional objects by transforming the application program's data. A representation of this process is shown in Figure 2-9, a medieval woodcut by Albrecht Durer (from The Metropolitan Museum of Art, Harris Brisbane Dick Fund, 1941). Using a mathematical version of Durer's mechanical approach, AGP can create various views of an object. By altering the viewing transformation, the best view for a particular application can be achieved.

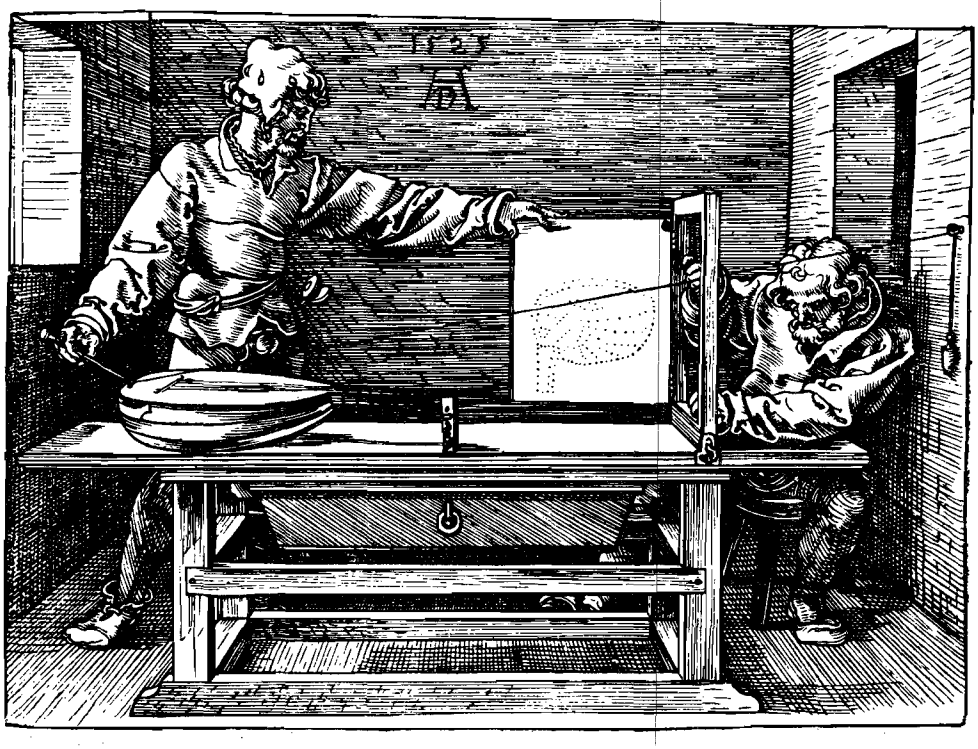

**Figure 2-9. Albrecht Durer, Artist Drawing a Lute**

Durer's artist is using string to mechanically project a lute (object) through a wooden frame to a hook on the wall. The pattern formed by the strings where they intersect the plane of the frame is the transformed view (image) of the lute. This transformation of an object into an image corresponds nicely to the constructs in AGP that accomplish the same thing.

AGP uses a *viewplane*, a plane on which a three-dimensional object is reduced to a two-dimensional image by passing lines called *projectors*, one through each point of the object, to a single point called the *center of projection*. In the Durer woodcut, the wooden frame contains the viewplane, the string is a projector, and the hook on the wall is the center of projection. When the center of projection is a finite distance from the viewplane, a *perspective projection* is obtained. When it is at infinity, that is, when the projectors are all parallel, a *parallel projection* is obtained (see Figure 2-10). Since the hook on the wall is not more than a few feet from the wooden frame, Durer is portraying a perspective projection.

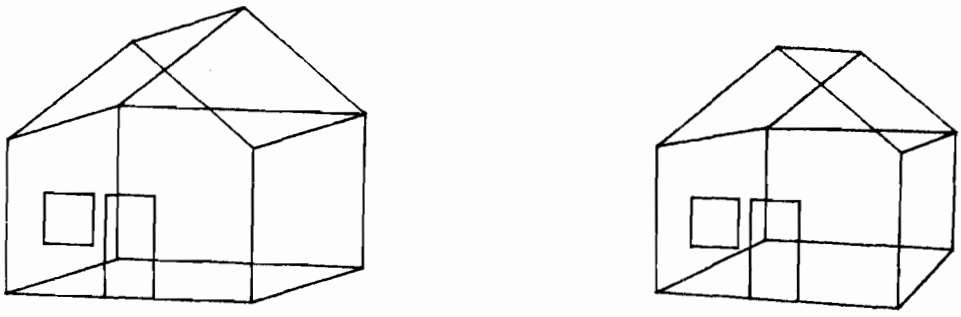

**Figure 2-10. Parallel (left) and Perspective (right) Projections**

In AGP, further control of an image's appearance is provided by depth clipping. Depth clipping controls how much of the object will be projected onto the viewplane. Objects in front of the *hither plane* and beyond the *yon plane* will not be projected. In Figure 2-11, the depth of a house has been clipped.

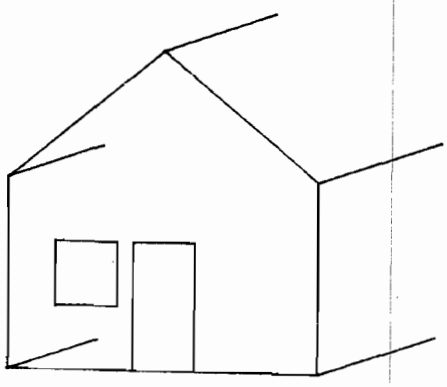

**Figure 2-11. Depth Clipping of an Object**

Once the portion of the object between the depth clipping planes is projected onto the viewplane, its image has only two dimensions, so the remaining steps are similar to those of the two-dimensional viewing transformation: the projected primitives are clipped to the window, the window is mapped to the viewport, and the viewport is displayed on the display device.

## SEGMENTS

A *segment* is a collection of output primitives that define an image. This image can be compared to a photograph, consisting of a specific object looked at from a particular viewpoint. In other words, a segment is a collection of output primitives to which a viewing transformation has been applied. One or more segments may appear on a display device simultaneously.

Segment attributes affect the appearance of segments. The segment attributes are *visibility*, *highlighting*, and *detectability*.

A fundamental difference between the attributes of segments and those of output primitives is their susceptibility to modification. Output primitive attributes are static, while segment attributes are not. The attributes of a primitive cannot be changed after a primitive is created. To modify a primitive's image, first you must either clear the display device or, if the primitive is contained in a segment, purge the segment. Then the primitive must be created again with the attributes in question modified.

A graphical picture can be altered easily by changing the attributes of one or more of its segments or by purging some of its segments. Using multiple segments in a picture allows changes to be made to one image without affecting those remaining, meaning a picture need not be erased and entirely reconstructed by the user in order to modify its appearance.

Output primitives are grouped into segments so the computer has an internal representation of the currently displayed graphical picture. This is useful for a number of reasons:

1. Primitives of a displayed image may be picked (chosen interactively) by an operator. Since segments are named, the picked image is identified by its segment name.

2. When a change is made to a picture, the display device may be updated by AGP without the application program recreating all the primitives.

3. Through segments, selected parts of a picture may be erased.

4. Segments may be highlighted (to attract the viewer's attention), made invisible (to temporarily remove them from view), or made detectable (so they may be picked).

The following example shows how, in Figure 2-12, the frame of a house, its door and window, and its floor are grouped into segments so that their visibility can be manipulated independently.

```
open segment 1
  draw frame of house
close segment 1

open segment 2
  draw door and window
close segment 2

open segment 3
  draw floor
close segment 3

make segment 1 visible
make segment 2 visibile
make segment 2 invisible
make segment 3 visible
make segment 2 visible
```

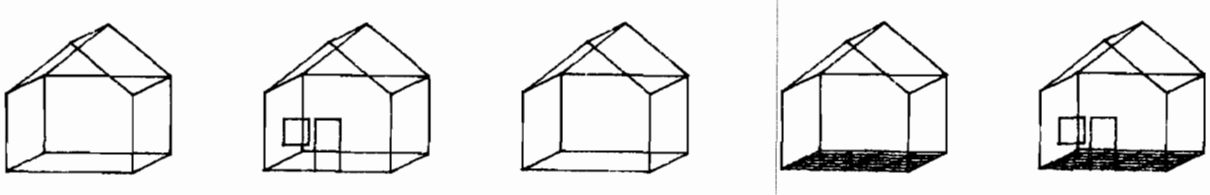

**Figure 2-12. A House Made Up of Segments**

Output primitives do not have to be grouped into segments. However, the computer has no internal representation of primitives outside of segments. They are therefore temporary in nature, in that when the display is updated, those primitives will not be carried over to the next version. The removal of all primitives not contained in visible segments is called a *new-frame-action*. Some applications do not require all primitives to be saved, so they need not use segments.

## INPUT

*Input devices* allow an operator to return information to an application program. There are five available input devices: button, keyboard, locator, valuator, and pick. A button device returns an integer value, good for applications which require only simple choices by the operator (rather than complex strings of characters such as those entered with a keyboard). A button device is analogous to a start/stop/select device, such as a button on a slide projector.

A keyboard device allows the operator to enter a string of characters, typically from an alphanumeric keyboard. The keyboard function returns to the calling program the string that was entered and the number of characters in the string. This device is useful for complex responses by operators.

A locator device provides a real coordinate pair depicting a point on the view surface. Typical locators are graphics tablets and cross-hair cursors. A possible use of this device would be designating the longitude and latitude of a city on a map.

A valuator device allows the operator to send a scalar value between 0.0 and 1.0 to the application program. The device can be a control dial or

multiposition switch, analogous to a dimmer switch on a light. The dimmer can be rotated to any position, which sets the light at the desired intensity.

A pick device lets the operator select output primitives using a graphics tablet or by positioning a cursor on the desired primitives' image. The *pick-ID* of the primitives (assigned through AGP when they were created) and the name of the segment containing the primitive(s) are returned to the application program. Using a pick device can be compared to a child pointing at the candy bar he wants in a candy store. He doesn't need to know the name of the candy. He simply sees it and identifies it by pointing. The person behind the counter, like AGP, identifies the candy's name for the child.

AGP obtains information from the input devices by either *requesting* or *sampling* the input. Before requested information can be returned to the application program, the operator must enter a termination command (such as pressing the return key). Sampled input is returned immediately, with no operator interaction required. Information from all input devices can be obtained by the request method, while only the locator and valuator devices may be sampled.

## CONTROL

Control functions are used to manage various aspects of a graphics application. Some of these are: initialization and modification of the graphics environment, inquiry of the features currently in effect, management of error routines, and sending device-dependent escape functions. Information concerning these topics is presented in Chapter 7 of this manual.

# Chapter 3
# Output Primitives and Attributes

## GENERAL

The application programmer describes objects in a three-dimensional graphical world to AGP for display. These objects are composed of six types of output primitives: move, line, polyline, polygon sets, marker, and text.

The appearance of the output primitives result from:

1. the parameters passed to the output primitive routine (for example, the coordinates of a line, the characters of a text string, or the type of marker),

2. the current values of the primitive attributes, and

3. the viewing transformation in effect.

When output primitives are created, these aspects are fixed for these primitives.

All output primitives are described in the world coordinate system using the current position as a reference point. Initially, the current position is the world coordinate origin (0.0, 0.0, 0.0). Each output primitive call results in a new current position being established. Certain operations, such as specifying a new viewing transformation or opening or closing a segment, reset the current position back to the origin.

Coordinate positions can be specified as either absolute or relative with respect to the current position. Relative coordinates are provided as a notational convenience for the programmer; AGP converts them to absolute coordinates when the call is executed.

Some output primitive calls are available in either two-dimensional or three-dimensional forms. Two-dimensional calls are abbreviations for the three-dimensional calls, meaning the Z coordinate of the current position is used and left unchanged when the two-dimensional form is used. If no three-dimensional calls are made, the Z coordinate of the current position always remains 0.0.

Attributes are the characteristics of graphics figures. Color, linestyle, linewidth, and pick-ID are the general attributes of output primitives. Polygon style, interior color and interior linestyle are attributes specific to polygon sets. Text size, gap, font, slant, orientation, and justification are attributes specific to text. Each has a current value

under the application program's control. The current values of applicable attributes determine the appearance of output primitives as they are generated.

When a house is drawn, its line primitives can be given a particular linestyle, such as "dashed" which is linestyle 2 on some devices, by setting the current value of linestyle to 2 before the house is created. What if the display device does not have linestyle 2? That is, what if the device cannot perform a requested attribute exactly? In these cases, AGP maps the requested attribute value to a supported attribute value; for example, if linestyle 2 were not supported, the house would be drawn in linestyle 1. The *Device Handlers Manual* details device specific functionality including supported linestyles.

In the following sections, general output primitives, general output primitive attributes, polygon-set attributes, text, and text attributes are described.

# GENERAL OUTPUT PRIMITIVES

## Move

| | |
|---|---|
| J2MOV (X,Y) | Move the current position (2D, absolute) |
| J3MOV (X,Y,Z) | Move the current position (3D, absolute) |
| JR2MV (DX,DY) | Move the current position (2D, relative) |
| JR3MV (DX,DY,DZ) | Move the current position (3D, relative) |

The move routines are used to set the current position to a point in the world coordinate system. They do not produce visual output on the display device. Instead, they affect the position at which subsequent primitives will be located.

Coordinate positions can be specified as either absolute or relative with respect to the current position. Also, the two-dimensional routines are abbreviations for the three-dimensional routines where the Z coordinate of the current position is left unchanged.

## Line

| | |
|---|---|
| J2DRW (X,Y) | Draw a line (2D, absolute) |
| J3DRW (X,Y,Z) | Draw a line (3D, absolute) |
| JR2DR (DX,DY) | Draw a line (2D, relative) |
| JR3DR (DX,DY,DZ) | Draw a line (3D, relative) |

The line routines draw a line from the current position to a point specified in the parameters. For absolute forms, this point is the world coordinate point specified: either (X,Y) or (X,Y,Z). For relative forms, the end of the line is found by adding the given offsets to the current position. After the line is output, the endpoint becomes the new current position. The following example draws the square in Figure 3-1. It can be assumed that the window and viewport are set at the default values. Therefore, the world coordinates are such that the X and Y values range from -1.0 to 1.0.

```
CALL J2MOV (-0.5,-0.5)      * move to the lower left corner
CALL J2DRW ( 0.5,-0.5)      * draw a square: bottom,
CALL J2DRW ( 0.5, 0.5)      *   right side,
CALL J2DRW (-0.5, 0.5)      *   top,
CALL J2DRW (-0.5,-0.5)      *   left side
```

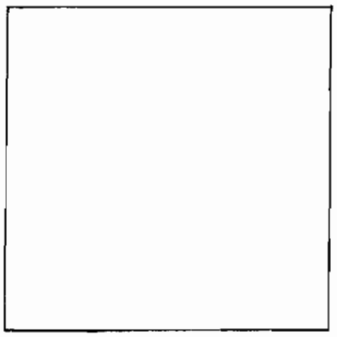

**Figure 3-1. Square Created with Moves & Draws**

The following example draws the outline of a house, as in Figure 3-2. The window and viewport are set at the default values. Therefore, the world coordinates are such that the X and Y values range from -1.0 to 1.0.

```
CALL J3MOV (-0.5,-0.5, 0.0)        * move to the lower left corner
CALL JR3DR ( 1.0, 0.0, 0.0)        * draw a house: bottom,
CALL JR3DR ( 0.0, 0.6, 0.0)        *   right wall,
CALL JR3DR (-0.5, 0.2, 0.0)        *   right side of roof,
CALL JR3DR (-0.5,-0.2, 0.0)        *   left side of roof,
CALL JR3DR ( 0.0,-0.6, 0.0)        *   left wall.
```

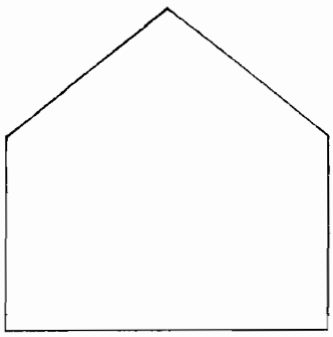

**Figure 3-2. House Created with Moves and Lines**

## Polyline

| | | |
|---|---|---|
| J2PLY | (NPOINT,XVEC,YVEC) | Draw a polyline (2D, absolute) |
| J3PLY | (NPOINT,XVEC,YVEC,ZVEC) | Draw a polyline (3D, absolute) |
| JR2PL | (NPOINT,DXVEC,DYVEC) | Draw a polyline (2D, relative) |
| JR3PL | (NPOINT,DXVEC,DYVEC,DZVEC) | Draw a polyline (3D, relative) |

The polyline routines are used to draw a connected line sequence. The number of points in the line sequence is found in NPOINT, while the points (or, for relative calls, the point offsets) themselves are passed as arrays in the remaining parameters. Initially, a move is made to the first point. The connected line sequence begins there, continuing to the second point, the third point, and so on, until the NPOINT point is reached.

The current position is left at the NPOINT element of the array. If NPOINT equals 1, these calls set the current position to the first element of the array and produce no lines; that is, a move is produced.

As in the move and line routines, coordinate positions can be specified as either absolute or relative to the current position. Note that the current position is updated after each vector is output, so that the relative coordinates of the next vector are calculated from this new position. Also, two-dimensional calls are shorthand for three-dimensional calls; they leave the current Z coordinate unchanged.

Polylines are a notational convenience for the application programmer. One polyline call may replace a number of line calls, producing a shorter but functionally equivalent application program. The following example draws the house in Figure 3-3 using only one polyline.

```
REAL X(5), Y(5), Z(5)
DATA X / 1.0, 0.0,-0.5,-0.5, 0.0/
DATA Y / 0.0, 0.6, 0.2,-0.2,-0.6/
      :
      :
CALL J2MOV (-0.5,-0.5)              * establish current position for
                                      relative polyline call
CALL JR2PL (5,X,Y)                  * draw house
```

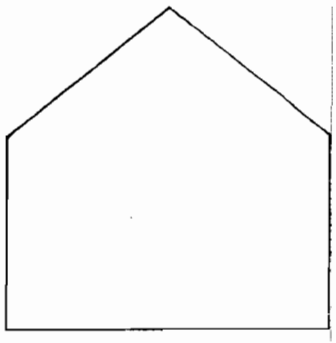

**Figure 3-3. A House Created with Polyline**

## Marker

| | |
|---|---|
| J2MRK (X,Y,MARKNO) | Mark a point (2D, absolute) |
| J3MRK (X,Y,Z,MARKNO) | Mark a point (3D, absolute) |
| JR2MK (DX,DY,MARKNO) | Mark a point (2D, relative) |
| JR3MK (DX,DY,DZ,MARKNO) | Mark a point (3D, relative) |

Markers identify individual points in the world coordinate system. Depending on a particular display device's capabilities, AGP utilizes either software or hardware to generate the markers. When one of these functions is invoked, the current position is updated to the point specified by the parameters, then the desired marker symbol is output centered at that point.

A variety of markers are available, selected via the MARKNO parameter. The size and orientation of markers is fixed for a device. Unlike other output primitives, they are not affected by the viewing transformation in effect when they are generated. AGP supports these 19 markers on all devices:

| | | |
|---|---|---|
| 1 - '.' | 7 - rectangle | 13 - '3' |
| 2 - '+' | 8 - diamond | 14 - '4' |
| 3 - '*' | 9 - rectangle with cross | 15 - '5' |
| 4 - 'O' | 10 - '0' | 16 - '6' |
| 5 - 'X' | 11 - '1' | 17 - '7' |
| 6 - triangle | 12 - '2' | 18 - '8' |
| | | 19 - '9' |

In addition to these, a device may support additional markers of its own. Refer to the *Device Handlers Manual* for more detailed information on markers.

In the following example the bottom and center of the house are marked. The house has been created previously, with several line primitive calls, a polyline call, or a polygon set call, with the appropriate data. From now on in this manual, the subroutine HOUSE is used to draw this series of line primitives. The window and viewport values are defaulted. Therefore, the world coordinates are such that X and Y values range from -1.0 to 1.0. The house is shown in Figure 3-5.

```
CALL HOUSE                          * draw a house
CALL J3MRK (0.0, 0.0, 0.0, 8)       * mark it with a diamond
```

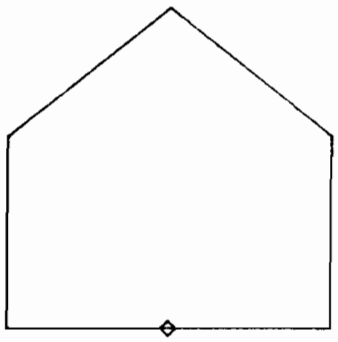

**Figure 3-5. Marking the Threshold of a House**

## GENERAL OUTPUT PRIMITIVE ATTRIBUTES

A primitive can have several different modifiable attributes, or characteristics, all of which are defined individually. A line, for example, can have four attributes: color, linestyle, linewidth, and pick-ID, while a marker can have two: color and pick-ID. Some attributes are general, that is, they can be applied to more than one primitive type, while some are specific to a single primitive type. The color attribute, for example, can be applied to all primitive types, while the font attribute can be applied to text only. Several attributes apply only to polygon sets. This section describes attributes that apply to all output primitives. A later section describes polygon-specific attributes.

AGP maintains a set of current attribute values for all primitive attributes. They are set to defaults at system initialization. A primitive is assigned system-maintained attribute values when it is created. All general attribute values except pick-ID are defined as indices in device-dependent tables. An attribute value can be changed from the default index to another index with a call to the appropriate subroutine.

Except for polygon style and color tables, the device-dependent attribute tables are not modifiable. Depending on the device, the linestyle table may have additional entries that the user may define (see the *Device Handlers Manual* for additional information regarding the linestyle table). Depending on the operating system, the size of some tables, such as the color table, may be altered.

In the following section, the general attributes of color, linestyle, linewidth, and pick-ID are discussed in addition to the specific attributes of polygons and text. Table 3-1 summarizes the general attributes which apply to the primitives line, polyline, marker, text, polygon edge and polygon interior.

**Table 3-1. General Primitive Attributes And Their Range of Applications.**

| | Output Primitives | | | | | |
|---|---|---|---|---|---|---|
| Attribute | LINE | POLYLINE | MARKER | TEXT | POLYGON EDGE | POLYGON INTERIOR |
| Color | X | X | X | X | X | |
| Linestyle | X | X | | X | X | |
| Linewidth | X | X | | X | X | |
| Pick-ID | X | X | X | X | X | X |

## Color

JCOLM (MODEL) — Choose the color model
JCOLR (COLOR) — Specify color
JDCOL (ID,COLOR,COLP1,COLP2,COLP3) — Redefine color table entry

(*NOTE: Polygon interior colors are specified by JPICL (COLOR)*; see "Polygon-specific Attributes")

Every graphics display device supported by AGP has its own color table defined in the device handler which lists the colors available on the device. Space for new entries can be added to a color table, or existing entries deleted, by changing the source code.

Choose the color of output primitives by setting the color table index (COLOR) in JCOLR. (This must be set before the output primitive is created.) This color does not affect the interior of polygons. To set the color of polygon set interiors, use JPICL.

To change a color table entry, use JDCOL to specify the table index (COLOR) and a set of color parameters (COLP1,COLP2 and COLP3). AGP offers two models to interpret these parameters:

- o The RGB (Red-Green-Blue) model (default case)
- o The HSL (Hue-Saturation-Luminosity) model

The RGB model, shown in Figure 3-6, is a color cube that has as its axes the primary additive colors: red, green and blue. Each point within the cube has a red intensity value (X coordinate), a green intensity value (Y coordinate) and a blue intensity value (Z coordinate). Each axis or intensity ranges from zero to one, and each point defines a unique color.

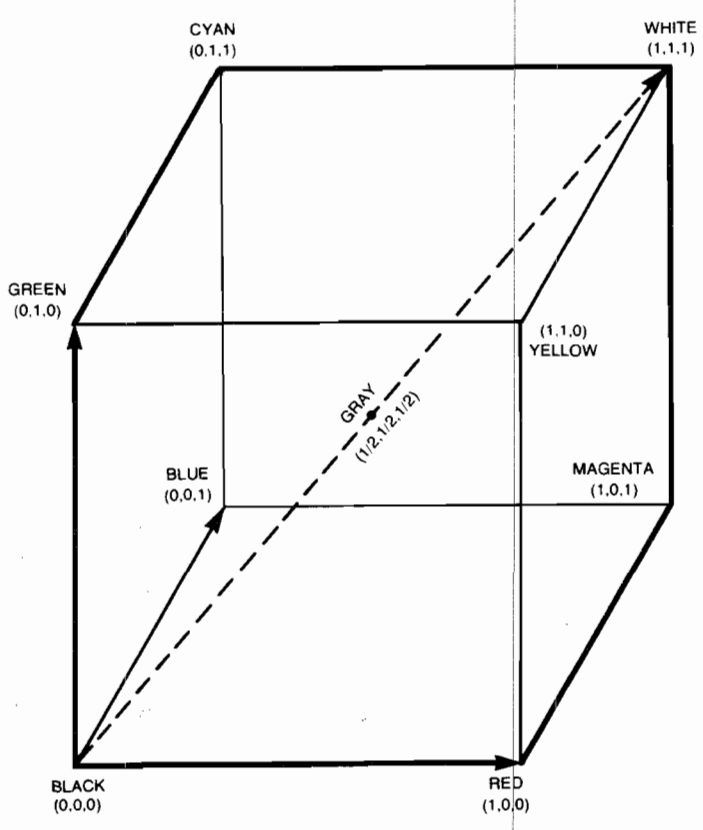


**Figure 3-6. The RGB Model**

The HSL model, shown in Figure 3-7, is a color cylinder in which:

- The angle about the axis of the cylinder, in fractions of a circle, is the hue.
- The radius is the saturation.
- The height is the luminosity (the intensity or brightness per unit area).

Angle, radius and height all range from zero to one, and each point within the cylinder defines a unique color with a hue value (angle), a saturation value (radius), and a luminosity value (height).

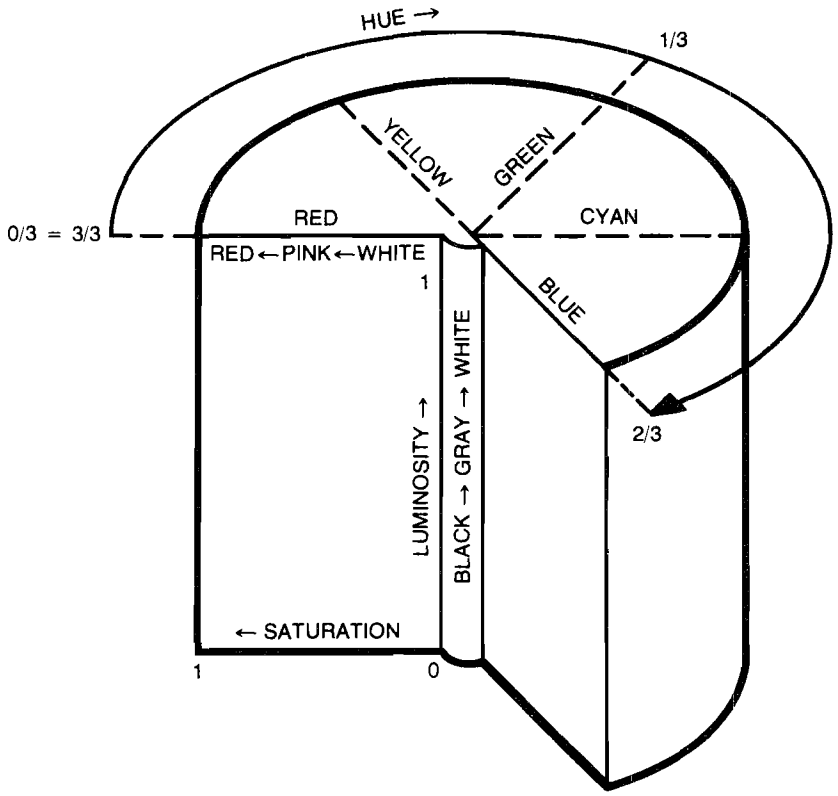


**Figure 3-7. The HSL Model**

AGP ignores calls to JDCOL if the device does not allow its color entries to be redefined.

To choose the specific color model, call JCOLM:

- Model = 1 is the RGB Model (default case)
- Model = 2 is the HSL Model

The default values for the color table are defined in the *Device Handlers Manual*. Every default color table has at least an entry at index 1 and display devices which have a background color have an entry at index 0. Color 1 is the default color for all primitives. The highest color index allowed on the system is 32767, but most devices allow fewer colors in their color tables. If JCOLR or JPICL specifies an index too large for the current graphics output device, color 1 is used instead.

The ability to assign colors to primitives and the actual effects produced (including the accuracy with which the desired color is expressed), depend on the current graphics output device. The *Device Handlers Manual* describes the features of all devices supported by AGP.

## Linestyle

JLSTL (LSTYLE) — Set linestyle index

JLSTL sets an index into a device-dependent table. With it the pattern used to draw lines, polylines, polygon set edges, and text can be specified.

A maximum of 255 linestyles is available, but the actual number supported by a particular graphics device is device-dependent.

Linestyles use various patterns of long line segments, short line segments, and blank spaces to produce a variety of line primitive appearances. Some examples are shown in Figure 3-8.

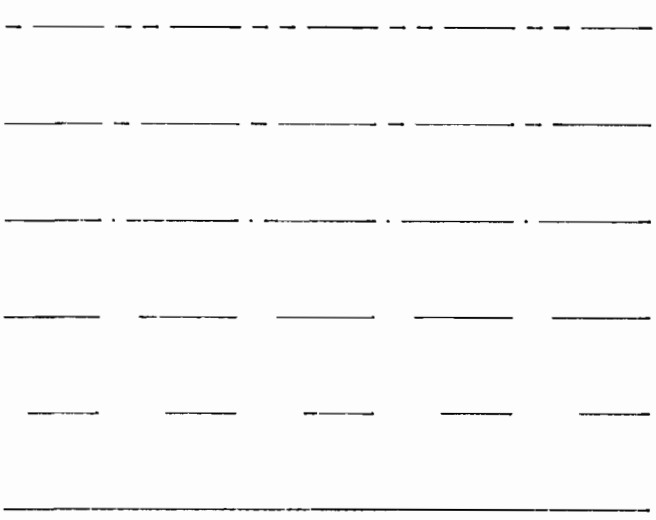

**Figure 3-8. Various Linestyles**

Care should be taken when linestyles that are not entirely solid are used. For instance, if a linestyle contains a number of blank spaces, a short line primitive might produce nothing visible, since the primitive's end could be reached before the non-blank portion is encountered.

Even when longer line primitives are drawn, a linestyle may not necessarily produce the desired appearance. Some linestyles have readily identifiable line primitive endpoints, while others do not. Those that do, called vector-adjusted linestyles, are good for objects whose endpoints must be seen, such as rectangles.

Of the remaining linestyles, a distinction is made between those that begin each line primitive in the same way (start-adjusted), and those that continue a new line primitive as if the previous primitive had never ended (continuous). Continuous linestyles are good for objects whose endpoints are *not* to be emphasized, such as curves. (Curves are drawn by drawing many small line primitives, one after another.)

The effect of linestyle on graphics text is device-dependent, as some devices allow text to be displayed in a variety of linestyles.

At AGP initialization, the linestyle index is 1.

## Linewidth

JLWID (LWIDTH) — Set linewidth index

JLWID sets an index into a device-dependent table. With it, the linewidth of lines, polylines, polygon set edges, and text can be defined. While all devices support at least one linewidth, the maximum number of supported linewidths is device-dependent. A value of 1 specifies the thinnest width possible. When multiple widths are supported, the width of the line increases as the value of LWIDTH does. If LWIDTH is greater than the number of linewidths supported, or if linewidth is less than 1, the linewidth is set to the thinnest available width.

Text is affected by linewidth only if the device is capable of applying it to text. Markers are not affected by this call as they are defined to be output with the thinnest linewidth supported by the device.

At initialization, the default linewidth is 1.

## Pick-ID

JPKID (PICKID) — Set pick identifier

JPKID sets the pick-ID attribute. It is used to give different names to different parts of a segment. In this way, a segment can be defined with several identifiable components.

When a primitive is created, the current pick-ID is bound to it. Its pick-ID and the name of the segment to which it belongs are returned to the application program whenever the primitive is selected by the operator using the pick input device. Pick-ID applies only to primitives within a segment. Primitives outside a segment do not have a pick-ID associated with them and are not detectable by the pick input function. The initial pick-ID attribute is 1.

Figure 3-9 represents a menu of candies. If the menu were identified as a segment, and each type of candy were given a separate pick-ID, each type of candy could be selected separately from the others.

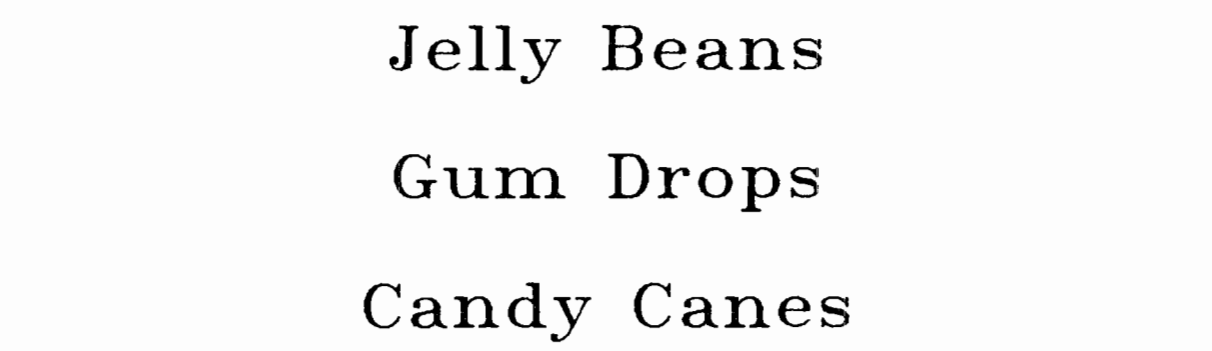


**Figure 3-9. A Menu of Candies**

## POLYGON-SPECIFIC ATTRIBUTES

| | |
|---|---|
| JPSTL(PINDEX) | Choose polygon style |
| JDPST(ID,PINDEX,DENSTY,ORIENT,EDGE,DIDDF) | Redefine polygon style |
| JPICL(COLOR) | Choose polygon interior color |
| JPILS(LSTYLE) | Choose polygon interior |

JPSTL selects the polygon style by specifying an index (PINDEX) into a work station's polygon style table. Each index points to an entry in the style table that specifies four descriptors of polygon style:

- o Interior density (DENSTY)
- o Interior orientation (ORIENT)
- o Edge display or non-display (EDGE)
- o Device independence or dependence (DIDDF)

These descriptors characterize the interior pattern and edge display of a polygon. They are described in detail in a following subsection.

JDPST modifies the contents of this style table. The work station ID is specified with the ID parameter and the index is specified with the PINDEX parameter. Some devices reflect the change made with JDPST retroactively, but for most devices the change appears only in later images. The effect of redefinition on previously output polygons is device dependent. If any of the parameters are out of range for that work station, the call is ignored.

JPICL selects the polygon interior color by specifying an index (COLOR) into a device-dependent color table, the same table used by JCOLR.

JPILS selects the polygon interior linestyle by specifying an index (LSTYLE) into a device-dependent linestyle table, the same table used by JLSTL.

Table 3-2 shows the attributes which apply to polygon sets, including those attributes specific to polygon sets and those which are general.

**Table 3-2. Attributes Which Affect Polygon Sets**

| Attribute | Interior | Edge (If Displayed) |
|---|---|---|
| *Polygon-Specific Attributes*: | | |
| Polygon Interior Color | X | |
| Polygon Interior Linestyle | X | |
| Polygon Style - | | |
| Interior Density | X | |
| Interior Orientation | X | |
| Edge Display | | X |
| Device-Independence/ Device-Dependence | X | X |
| *General Attributes* | | |
| Color | | X |
| Linestyle | | X |
| Linewidth | | X |
| Pick-ID | *(Entire Figure)* | |

*NOTE: When used with a polygon set primitive, JPICK (the pick-ID attribute) applies to the complete figure, including the interior and the edge. It is applied to the polygon set in the same way that it is applied to the other output primitives.*

In the following subsections, the polygon-specific attributes are described.

## Polygon Style

| | |
|---|---|
| JPSTL(PINDEX) | Choose polygon style |
| JDPST(ID,PINDEX,DENSTY,ORIENT,EDGE,DIDDF) | Redefine polygon style |

The polygon style table, which is defined separately for each work station, contains a number of entries, each of which specifies the following four descriptors of polygon style:

- o DENSTY (interior density)
- o ORIENT (interior orientation)
- o EDGE (edge display/non-display)
- o DIDDF (device-independence/dependence)

DENSTY (interior density) defines the density of the polygon set's interior fill lines in a range from transparent, or no fill (0.0), to solid, or complete fill (1.0 or -1.0). A positive number requests parallel fill lines; a negative number requests crosshatching with two sets of fill lines at right angles to each other.

ORIENT (interior orientation) defines the angle of orientation of the polygon set's interior fill lines to the horizontal axis of the view surface, and ranges from -90.0 degrees to 90.0 degrees.

Orientation defines the angle of the first set of hatch lines; the second set is always perpendicular to the first set.

EDGE (edge display/non-display) defines whether the boundary of the polygon set is to be drawn (1) or not drawn (0). If the edge is drawn, it has the attributes of color, linestyle and linewidth in effect for general primitives at the time of generation.

DIDDF (device-independence/dependence) specifies whether the polygon is to be generated exactly as specified by the polygon style index (device-independent) or whether it is to be generated using the hardware of the device (device-dependent).

With device-independence (1), hardware is used only when it can produce the range of polygon styles exactly; software is used when a device does not have the necessary hardware filling capability. This ensures portability of the figure from device to device, but may be inefficient on some devices.

With device-dependence (0), only the polygon capability of the device is used to generate the polygon set. Output varies, depending on the device.

While a device with polygon capability would generate the polygon set as close to specification as it can, a device without it would produce only a minimal representation. Device-dependence is useful when speed is essential and accuracy is not.

Figure 3-10 shows the sixteen default polygon styles. These sixteen are available for every device in a device-independent manner; additional styles may be defined for some devices. Note that styles 15 and 16 differ only in presence or absence of an edge; this difference is visible when the polygon edge and interior are different colors.

At initialization, the default polygon style index is 1. Refer to the *Device Handlers Manual* for a description of the default polygon style table for a specific device.

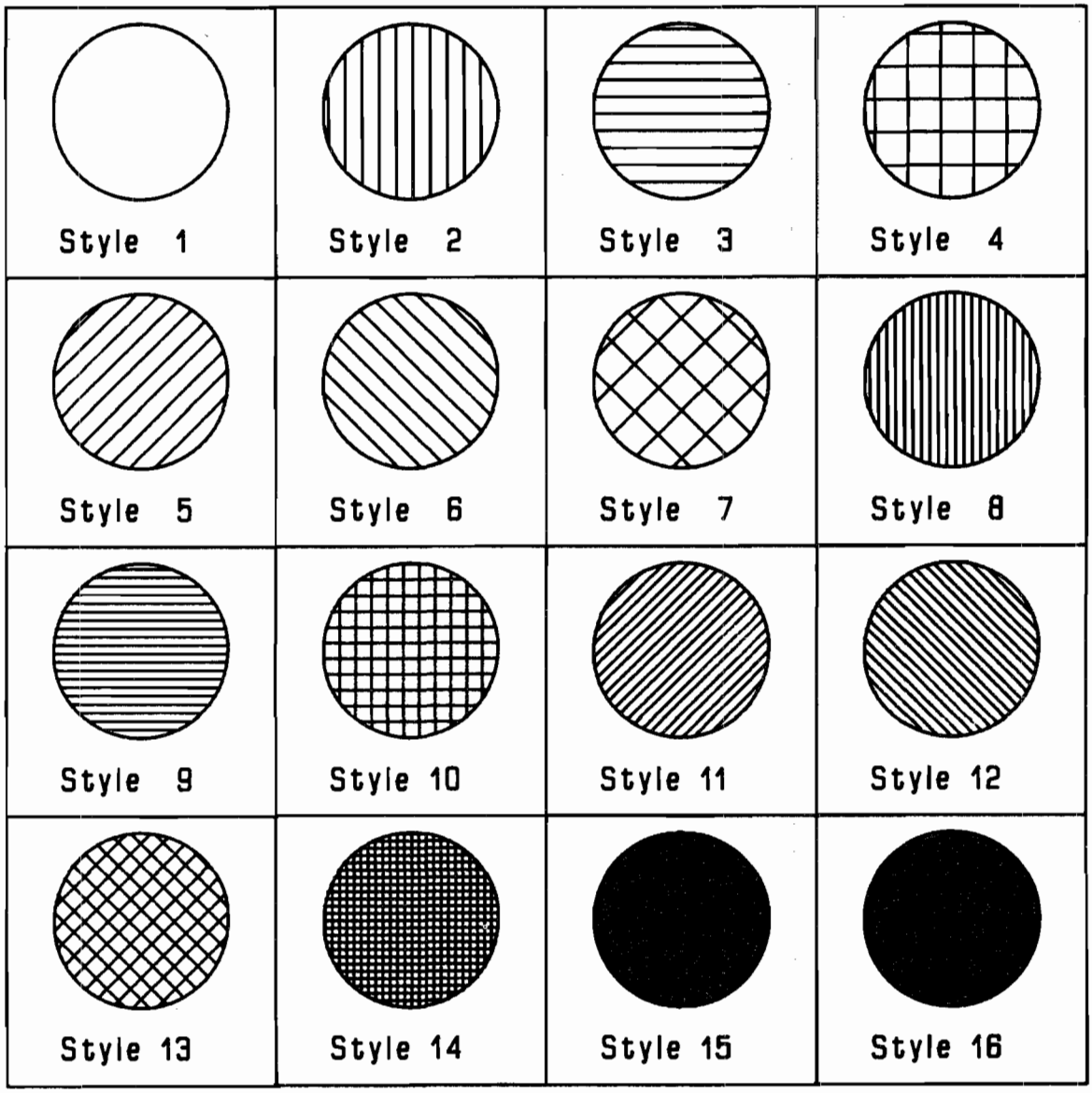


**Figure 3-10. Standard Polygon Styles**

## Polygon Interior Color

JPICL (COLOR) — Choose polygon interior color

JPICL selects the polygon interior color by specifying an index (COLOR) in a device-dependent color table. This is the same color table used by JCOLR, which chooses the color for polygon edges and other primitives.

COLOR 0 is the background color. On a CRT, the background color may be used to erase existing lines.

Although the color table for each device contains indices for 32767 different colors, the actual number of colors supported varies from device to device.

Indices which cannot be assigned a distinct color default to index 1. For example, on an 8-pen plotter indices 1 through 8 are assigned a unique pen number; indices 9 through 32767 are assigned the same pen as index 1.

Call JDCOL to modify the contents of the color table.

Call JICOL to inquire the contents of the color table and JI1IN to inquire the current polygon interior color.

At initialization, the default polygon interior color index is 1.

## Polygon Interior Linestyle

JPILS (LSTYLE) — Choose polygon interior linestyle

JPILS defines the polygon interior linestyle attribute for subsequently generated polygons by indexing a device-dependent linestyle table. This is the same table indexed by JLSTL, which chooses the linestyle for other primitives.

LSTYLE is the linestyle index for polygon interiors. AGP allows 255 distinct linestyles, although most display devices cannot support this many. If the user asks for a linestyle which is out of the device's range, the index defaults to one.

Call JIWS to inquire the number of distinct linestyles on a given display device.

At initialization, the default polygon interior linestyle index is 1.

# TEXT

AGP supports three different levels of graphics text. High quality text strictly adheres to all user-requested text attributes and therefore is slowest to produce. Low quality text ignores or loosely applies certain attributes and therefore is fastest to produce. Medium quality text is between the other two both in terms of accuracy of representation and in the time required to produce it.

The concept of a character cell is needed to discuss text. As shown in Figure 3-11, it is a box that defines the size of an individual text character. It generally contains the entire character, but in some cases, for example "y", part of the character may extend outside the character cell. Text fonts are described as either fixed or variable-width, depending on whether the character cells of different characters have the same or different widths. AGP's high quality text fonts are all variable-width.

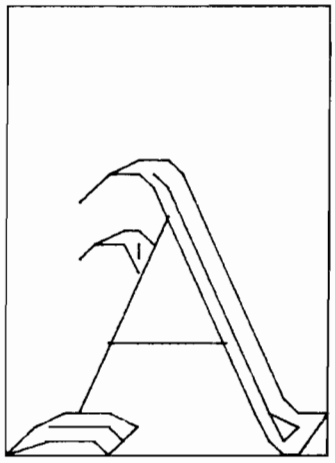

**Figure 3-11. Character Cell**

## High Quality

**JTEXH (NCHARS,STRING)** — **Output high quality text**

Each high quality character is formed using a series of lines, called strokes. The coordinates for these strokes are supplied in a font file. For high quality text, all text attribute specifications are strictly applied. The strokes that make up the characters are subject to the viewing transformation in the same way as line primitives.

After outputting a high quality text string containing NCHAR characters, the current position is updated to the lower left corner of where the next character cell would have been, had there been another character output. Figure 3-12 shows the current position before and after a string is output.

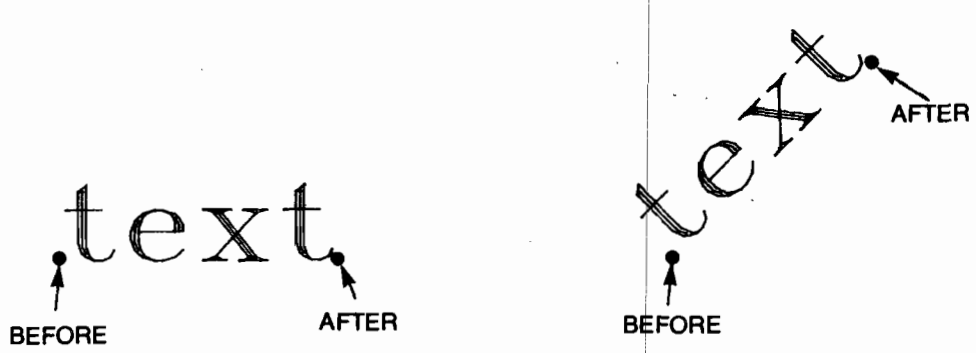


**Figure 3-12. Text and Updating Current Position**

## Medium Quality

JTEXM (NCHARS,STRING) Output medium quality text

Medium quality characters are output using the hardware character generator in the graphics display device. If hardware character capabilities do not exist, software at the device handler level will simulate hardware-generated text.

The orientation of medium quality text is always upright. AGP positions each character individually and uses the hardware character generator or simulator, depending on the device, to output the characters one at a time. Once the entire string has been output, the current position is updated to the same position at which high quality fixed-width text would have left it (refer to Figure 3-12).

Medium quality text is affected by the viewing transformation as follows. The lower left corner of each character cell is placed in the same spot as if high quality text were being output, but the characters are all vertical and their sizes chosen on a "smaller best fit" basis.
Smaller best fit is defined to be:

1. the largest character whose height is less than or equal to the requested cell height and whose width is less than or equal to the requested cell width, or,

2. if even the minimum hardware size does not meet criterion 1, the minimum character size is used.

## Low Quality

JTEXL (NCHARS,STRING) Output low quality text

Low quality characters are output using the hardware character generator if one exists. As with medium quality text, device handler software will simulate hardware character generation if one does not exist. The size of low quality text is also determined using the "smaller best fit" algorithm. AGP then generates the entire string as a unit starting at the current position. After outputting a low quality text string, the current position is updated to the same position at which fixed-width high quality text would have left it (see Figure 3-12).

# TEXT-SPECIFIC ATTRIBUTES

The following section describes attributes specific to the text primitive. As shown in Table 3-3, the number of modifiable text attributes available depends on the level of text used.

**Table 3-3. Text-Specific Attributes**

| Text Attribute | LOW TEXT | MEDIUM TEXT | HIGH TEXT |
|---|---|---|---|
| Size | X | X | X |
| Gap | | X | X |
| Orientation | | X | X |
| Justification | X | X | X |
| Font | | | X |
| Slant | | | X |

## Size and Gap

JCSIZ (WIDTH,HEIGHT,GAP) Define character size and gap

JCSIZ sets the size attribute for subsequently created text. WIDTH and HEIGHT specify the world coordinate size of a character cell. GAP specifies the spacing between character cells.

For low and medium quality text the actual character size is determined on a "smaller best fit" basis. Low quality text size is determined only once for the entire string, while medium quality text size is recomputed for each character so that the viewing transformation is more exactly applied. For high quality text with a fixed-width font, each character maps exactly to the requested size. For high quality text with a variable-width font, the letter "A" maps to the requested size, and the width of the other characters varies according to their widths in the font. This is shown in Figure 3-13.

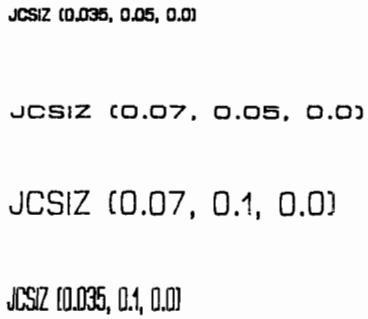


**Figure 3-13. Various Text Widths and Heights**

GAP, which determines the spacing between character cells, is expressed as a fraction of character cell width. A gap of 0.0 produces "normal" spacing. A positive value spreads characters out, while a negative value brings the characters closer together. Figure 3-14 shows the effect of gap on high quality text.

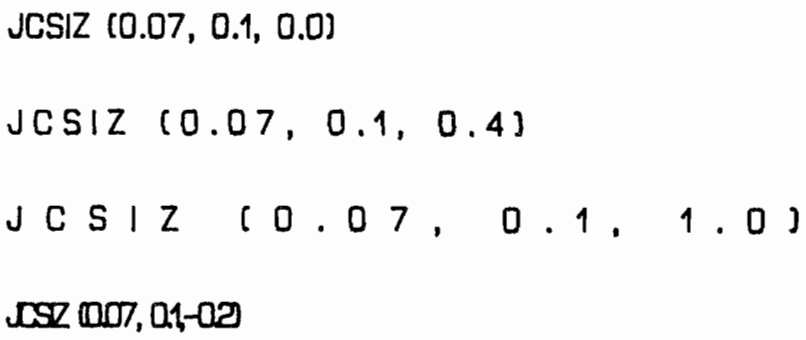


**Figure 3-14. Various Text Gaps**

Gap has no affect on character spacing for low quality text, but it does affect the placement of the current position at the completion of the text call. To understand this, remember that low quality text is sent to the device a string at a time; the device controls the placement of individual characters. After the string is sent, the current position is updated just as if fixed-width high quality text had been used. That is, the current

position is updated exactly according to the requested width and gap independent of the actual results on the device, meaning gap will affect where the next string begins, but has no affect on this string.

Contrast this with medium quality text, which is sent to the device on a character-by-character basis. After the first character is sent, the gap is applied in determining the placement of the current position. Then the next character is placed starting at the new current position, the current position is updated, the third character is placed, and so on. The result here is that gap affects both the position of characters in this string and where the next string will begin.

High quality text, of course, is sent to the device neither on a string nor character basis, but "stroke-by-stroke". Width and gap determine exactly the position of characters in this string and the beginning position of the next string.

The initial values are 0.05 for the height and 0.035 for the width in world coordinate units. Gap is initially 0.0, meaning additional space will neither be added nor subtracted from between character cells.

## Orientation

JCORI(XBAS,YBAS,ZBAS,XPLN,YPLN,ZPLN) Define character orientation

JCORI specifies the character base vector and the character plane vector in world coordinates. The character base vector is (XBAS,YBAS,ZBAS) and the character plane vector is (XPLN,YPLN,ZPLN). Both vectors are relative to the origin.

The character base vector determines the direction of textual output; that is, the bottoms of characters are lined up along this vector. The base of each character cell extends in the same direction as the character base vector; the sides of each character cell lie in the same plane as the character base and character plane vectors.

Another vector is derived from these two character vectors. It is called the character up vector, and is simply that vector perpendicular to the character base vector lying in the same plane as the character plane vector. This vector describes the up direction of characters (see Figure 3-15).

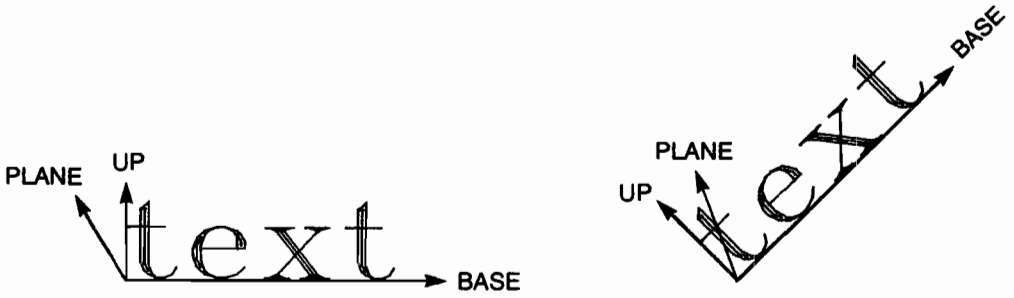


**Figure 3-15. Character Vectors: Base, Plane, and Up**

The effects of these vectors are summarized in Figure 3-16. The plane vector is only relevant to high quality text - the other two ignore it. The base vector is relevant to high quality text, but it also has a limited effect on medium quality text, as follows: the current position is updated character-by-character, but the character bases themselves are not parallel. No character vectors affect low quality text.

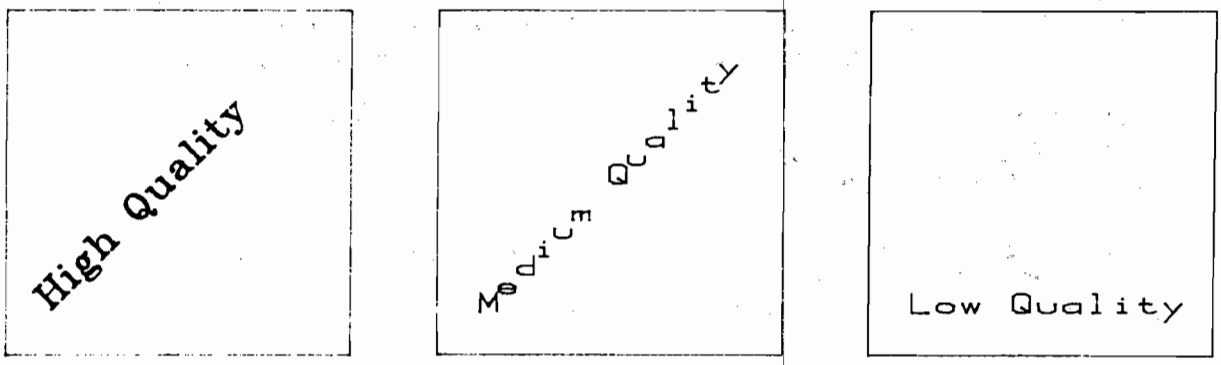


**Figure 3-16. Effect of Character Orientation**

The initial character base is (1.0,0.0,0.0) in world coordinates. The initial character plane is (0.0,1.0,0.0). Characters initially appear parallel to the X Y plane with their bases advancing along the positive X axis.

## Justification

JJUST (BASE,UP) — Set the text justification

JJUST determines the placement of the entire text string relative to the current position. A text string may be justified both horizontally (along the character base vector) and vertically (along the character up vector). Conceptually, justification may be thought of as putting an imaginary box around the entire text string and then moving the box and the text string the required amount.

BASE is expressed as a fraction of the total string width, and UP is expressed as a fraction of the total string height. Their initial values are both 0.0. BASE = 0.0 implies that 0% of the width of the text string is to be placed on the side of the current position opposite the character base vector. UP = 0.0 implies that 0% of the height of the character cell is to be placed on the side of the current position opposite the character up vector. If the character up vector points "up" and the character base vector points "right", then the initial value of (0.0, 0.0) implies that text strings are output with the current position at the lower left corner of the text string.

The example below produces the text in Figure 3-17.

```
CALL JJUST (0.0, 0.0)              * use default justification
CALL J2MRK (0.0, 0.6, 3)           * set and mark current position
CALL JTEXH (12, 12HDefault Just)   * output high quality text
     :
CALL JJUST (0.5, 0.0)              * horizontally justified
CALL J2MRK (0.0, 0.3, 3)           *
CALL JTEXH (12, 12HHorizontally)   *
     :
CALL JJUST (0.0, 0.5)              * vertically justified
CALL J2MRK (0.0, 0.0, 3)           *
CALL JTEXH (10, 10HVertically)     *
     :
CALL JJUST (0.5, 0.5)              * justified both ways
CALL J2MRK (0.0,-0.3, 3)           *
CALL JTEXH (8, 8HCentered)         *
     :
CALL JJUST (1.0, 1.0)              * upper, right corner
CALL J2MRK (0.0,-0.6, 3)           *
CALL JTEXH (11, 11HUpper Right)    *
```

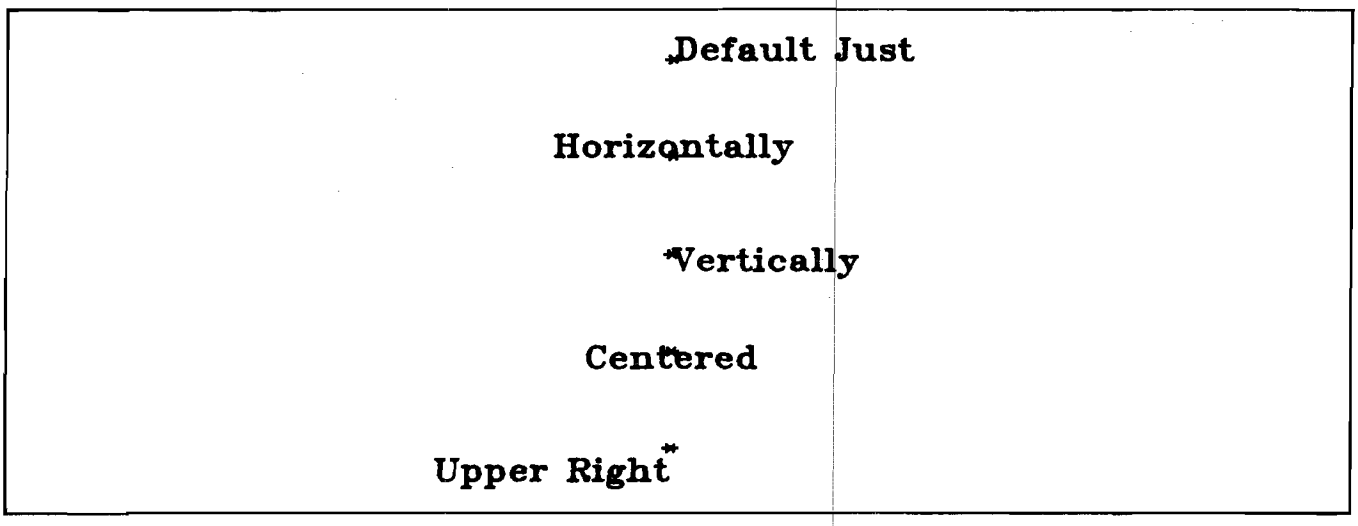


**Figure 3-17. Text Justification**

## Font and Slant

JDFNT(FONT,SLANT,*machine-dependent parameters*) Define a font
JFONT(FONT) Select the current font

The JDFNT call associates a font number with a slant and a font definition for use with high quality text. FONT is the number by which that font will be subsequently identified. The SLANT parameter specifies the angle to slant each character as it is drawn. The *machine-dependent parameters* specify the font definition to be used.

JFONT is used to select the current high quality text font. Initially, the current font = 0, signifying that there is no current font. A value other than 0 will set that font to be the current font for subsequent high quality text calls.

The following example, having *machine-dependent parameters*, produces the text fonts in Figure 3-18.

```
CALL JDFNT (1, 0.0,...) define font #1 = Eurostyle, unslanted
CALL JDFNT (2, 0.0,...) define font #2 = Simplex Roman, unslanted
CALL JDFNT (3, 0.0,...) define font #3 = Triplex Roman, unslanted
CALL JDFNT (4, 0.0,...) define font #4 = Script, unslanted
CALL JDFNT (5, 0.0,...) define font #5 = Mathematical, unslanted
CALL JDFNT (6, 0.0,...) define font #6 = Gothic, unslanted
```

```
      :
      :
CALL J2MOV (-0.9, 0.6)                  * set current position
CALL JFONT (1)                          * current font is font 1
CALL JTEXH (9, 9HEurostyle)             * draw high quality text
      :
CALL J2MOV (-0.9, 0.3)                  * move current position
CALL JFONT (2)                          * current font is font 2
CALL JTEXH (13, 13HSimplex Roman)       *
      :
CALL J2MOV (-0.9, 0.0)                  *
CALL JFONT (3)                          * font 3
CALL JTEXH (13, 13HTriplex Roman)       *
      :
CALL J2MOV (-0.9,-0.3)                  *
CALL JFONT (4)                          * font 4
CALL JTEXH (6, 6HScript)                *
      :
CALL J2MOV (-0.9,-0.6)                  *
CALL JFONT (5)                          * font 5
CALL JTEXH (12, 12HMathematical)        *
      :
CALL J2MOV (-0.9,-0.9)                  *
CALL JFONT (6)                          * font 6
CALL JTEXH (6, 6HGothic)                *
```


**Figure 3-18. Text Fonts**

The SLANT parameter specifies the angle in radians to slant each character. The following example, having *machine-dependent parameters*, produces the various slants in Figure 3-19.

```
CALL JDFNT (1, 0.0,...) define font #1 = Eurostyle, unslanted
CALL JDFNT (2, 0.2,...) define font #2 = Eurostyle, slight slant
CALL JDFNT (3, 1.0,...) define font #3 = Eurostyle, more slant
CALL JDFNT (4,-1.0,...) define font #4 = Eurostyle, back slant
CALL JDFNT (5,-0.2,...) define font #5 = Eurostyle, less back slant

CALL JCSIZ (0.07, 0.1, 0.2)              * set character size and gap
        :
        :
CALL JFONT (1)                           * no slant
CALL J2MOV (-0.5, 0.6)                   * set current position
CALL JTEXH (13, 13HDefault Slant)        * draw high quality text

CALL JFONT (2)                           * slight forward slant
CALL J2MOV (-0.5, 0.3)                   * set current position
CALL JTEXH (10, 10HItalicized)           * draw high quality text

CALL JFONT (3)                           * more forward slant
CALL J2MOV (-0.5, 0.0)                   * set current position
CALL JTEXH (12, 12HFull Italics)         * draw high quality text

CALL JFONT (4)                           * opposite direction
CALL J2MOV (-0.5,-0.3)                   * set current position
CALL JTEXH (14, 14HBackward Slant)       * draw high quality text

CALL JFONT (5)                           * still opposite direction
CALL J2MOV (-0.5,-0.6)                   * set current position
CALL JTEXH (14, 14HLess Backslant)       * draw high quality text
```


**Figure 3-19. Text Slants**

## ALPHANUMERIC DISPLAY

**JALPH (ID,STRING,NCHARS)** **Output to the alphanumeric display**

**JALPH allows the application program to send non-graphics text and data to an alphanumeric display device. This call can be used to send prompts, give status, or print other messages.**

**ID identifies the work station associated with the alphanumeric display device. The text in STRING should be NCHARS long. A carriage-return line-feed (CRLF) is attached to the end of STRING when it is sent to the device, unless the last character in STRING is an underscore. This suppresses the**

CRLF. If the alphanumeric device is the same as the graphics display device, any escape codes which affect the graphics application should not be passed in STRING, since they may place the graphics display device in an unknown state.

JALPH's output is not included in segments. This is because JALPH does not create graphics output primitives, and only graphics output primitives are saved in segments.

# Chapter 4
# Viewing Transformations

## GENERAL

A viewing transformation is the method by which two or three-dimensional objects are turned into viewable images. Viewing transformations modify objects for viewing by reducing them from three dimensions to two dimensions and converting them to device-dependent units as shown in Figure 4-1.

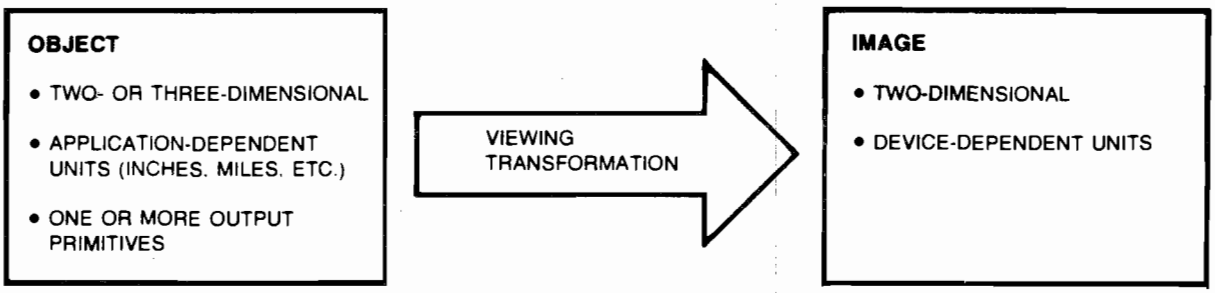


**Figure 4-1. The Viewing Transformation**

AGP treats two-dimensional viewing and three-dimensional viewing in an integrated fashion; the two-dimensional case is a subset of the three-dimensional case. The viewing capabilities are such that two-dimensional users do not need to know about three-dimensional constructs.

# TWO-DIMENSIONAL VIEWING TRANSFORMATION

A number of concepts are needed to understand how AGP turns two-dimensional objects in the world coordinate system into images on a display device. The steps in this transformation are summarized in Figure 4-2.

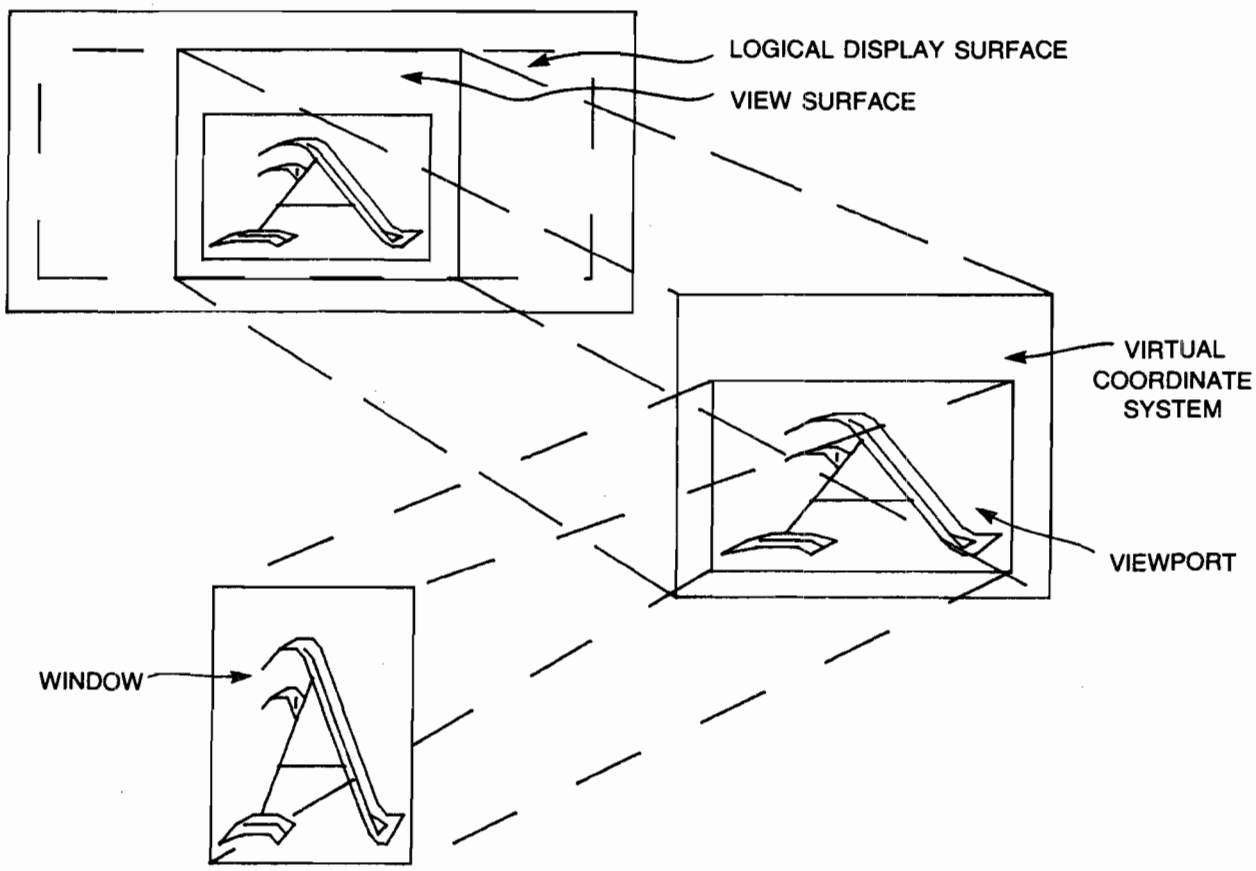


**Figure 4-2. Two-Dimensional Viewing Transformation**

Logical display limits allow a subset of a display device, known as the logical display surface, to be used for output. On a plotter, the bed of the plotter may be 500 millimeters by 1000 millimeters, while the paper in use might be only 250 millimeters square. The logical display limits of the plotter could be set to that size, so that AGP would treat the plotter from that point on as if it were physically 250 millimeters square.

The logical display limits, in turn, determine the size of the *view surface*. The view surface's dimensions are determined by:

1. the aspect ratio of the virtual coordinate system, and

2. the logical display limits of the display device.

Recall that the virtual coordinate system is the two-dimensional system whose units range from 0.0 to 1.0. World coordinate data is translated to virtual coordinates, allowing the user program to remain device-independent. However, not all devices on which graphics is produced have the same aspect ratio (height-to-width ratio). If the virtual coordinate system were always a unit square, a non-square logical display surface (one that is rectangular) would not be entirely available. To make all of a non-square logical display surface available, AGP allows the aspect ratio of the virtual coordinate system to be changed as needed.

The view surface is the largest rectangle within the logical display limits having the aspect ratio of the virtual coordinate system. Once the view surface is determined, a portion of it is designated the viewport. An application might use several viewports, perhaps to show different views of the same object.

The window defines what objects will become images. In Figure 4-2, the window just encompasses the letter "A". Objects whose primitives lie outside the window can be *clipped* (excluded from view) and not mapped to the viewport. When the aspect ratio of the window does not match that of the viewport, this mapping may cause distortion; that is, the images of the primitives will appear too "fat" or "thin". Distortion is deliberately used in Figure 4-2 to differentiate between the window and viewport.

## Display Limits

JDLIM (ID,XMIN,XMAX,YMIN,YMAX) Set logical display limits

JDLIM specifies the area of the display device on which output may be produced by setting limits. Logical display limits are useful for excluding a portion of the display device from receiving images. XMIN, XMAX, YMIN, and YMAX are described in millimeters as offsets from the device's origin (the origin of a device is usually in its lower-left corner). The work station containing the device is specified via the ID parameter. (Refer the appropriate system supplement and to Chapter 7 of the *AGP Reference Manual*.)

JDLIM implicitly redefines the view surface to be the maximum rectangle within the logical display limits that has the same aspect ratio as the virtual coordinate system.

## Aspect Ratio

JASPK (XSIZE,YSIZE) Set aspect ratio of virtual coordinate system

JASPK sets the aspect ratio of the virtual coordinate system (and hence the aspect ratio of the view surface) to be YSIZE divided by XSIZE. A ratio of 1.0 specifies a square virtual coordinate system, a ratio greater than 1.0 specifies it to be higher than it is wide, and a ratio less than 1.0 specifies it to be wider than it is high. Since XSIZE and YSIZE are used to form a ratio, they may be in any convenient units (so long as both are specified in the same units).

Based on the aspect ratio, AGP calculates the range of the X and Y coordinates in the virtual coordinate system. The coordinates of the longer side always range from 0.0 to 1.0, and the coordinates of the shorter side range from 0.0 to a value that achieves the desired aspect ratio. JASPK is said to define the *limits* of the virtual coordinate system, as shown in Table 4-1.

**Table 4-1. Virtual Coordinate System Limits.**

| Aspect Ratio (AR) | X Limits | Y Limits |
|---|---|---|
| AR < 1.0 | (0.0, 1.0) | (0.0, AR) |
| AR = 1.0 | (0.0, 1.0) | (0.0, 1.0) |
| AR > 1.0 | (0.0, 1.0/AR) | (0.0, 1.0) |

When JASPK is called, the viewport is automatically set to map to the entire view surface. For example, the HP 2623A Graphics Terminal has an aspect ratio of 389/511 = 0.7612. To utilize the entire visible screen, the application program may:

```
REAL XSIZE, YSIZE          *
      :                    * width of HP2623A is 511 device units
XSIZE = 511.0              *   and height is 389 device units
YSIZE = 389.0              * set aspect ratio of virtual
CALL JASPK (XSIZE,YSIZE)   *   coordinate system to match
      :
```

The initial aspect ratio of the virtual coordinate system is 1.0, meaning the virtual coordinate system is a unit square. This produces a view surface that is the largest inscribed square within the logical display limits. By changing the aspect ratio, the view surface can become the largest inscribed *rectangle* within the logical display limits.

## Viewport

JVIEW (VPXMIN,VPXMAX,VPYMIN,VPYMAX) Set viewport

JVIEW sets the viewport. VPXMIN, VPXMAX, VPYMIN, and VPYMAX are specified in virtual coordinates and must be within the limits of the virtual coordinate system defined by JASPK. The initial viewport is:

(VPXMIN = 0.0; VPXMAX = 1.0, VPYMIN = 0.0; VPYMAX = 1.0)

The following program section uses two viewports (see Figure 4-3). The HOUSE subroutine has been described previously; it draws a house using line primitives. The output is shown below.

```
CALL JVIEW (0.0, 0.5, 0.0, 0.5)    * set viewport to left half
CALL HOUSE                         *   and draw a house
      :
CALL JVIEW (0.5, 1.0, 0.0, 0.5)    * set viewport to right half
CALL HOUSE                         *   and draw house again
      :
```

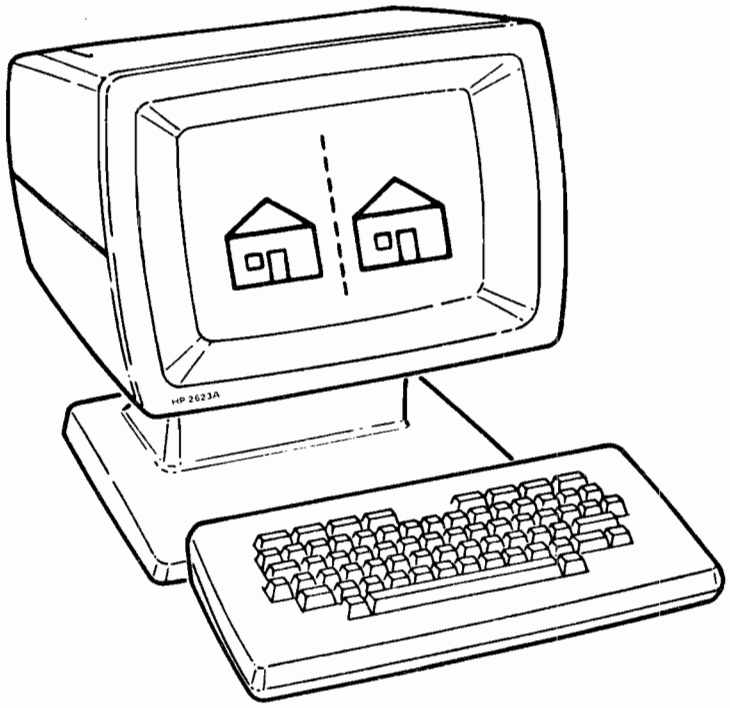


**Figure 4-3. Two Viewports**

## Window

```
JWIND (WXMIN,WXMAX,WYMIN,WYMAX)          Set window
JCLPW (OPCODE)                           Control window clipping
```

The window is set via JWIND. For two-dimensional applications, the window is set in the X Y plane of the world coordinate system. This is not true of three-dimensional applications, where the window is set in the viewplane. (For two-dimensional applications, these are the same, since the window is set by default in the X Y world coordinate plane, which is the viewplane.)

The following example shows the window and viewport being set to avoid distortion.

```
CALL JVIEW (0.0, 1.0, 0.0, 0.2)          * viewport is rectangle
CALL JWIND (100.0, 200.0, 0.0, 20.0)     * window is a rectangle
```

The window's aspect ratio should be the same as the aspect ratio of the viewport to avoid distortion. In general, setting the window by calculating its dimensions as a function of the viewport is a good way to prevent distortion. In this way, no assumption is made about a previously set viewport, helping to ensure that the visual results will be those desired.

JCLPW enables (OPCODE = 1) or disables (OPCODE = 0) the window clipping function. Clipping is desirable when primitives may extend beyond the window. If clipping is suppressed and primitives *do* exceed window boundaries, the resulting image is unpredictable.

The window is initially:

(WXMIN = -1.0; WXMAX = 1.0; WYMIN = -1.0; WYMAX = 1.0)

This produces a window with an aspect ratio of 1.0, which is the same as the aspect ratio of the initial viewport. Window clipping is initially enabled. It is strongly recommended that window clipping be left enabled, at least until a program has been fully debugged and all primitives are known to lie completely within the window. At this point, some small performance gain may be realized by disabling window clipping.

## THREE-DIMENSIONAL VIEWING TRANSFORMATION

Although three-dimensional viewing transformations are conceptually a difficult part of AGP, they are very powerful in nature. With relatively little application code, displayed images can be given a variety of perspectives and appearances.

AGP uses a viewplane, a plane on which a three-dimensional object is projected to a two-dimensional image by passing lines called projectors, one through each point of the object, to a single point called the center of projection (see Figure 4-4). When the center of projection is a finite distance from the viewplane, a perspective projection is obtained. When it is at infinity, that is, when the projectors are all parallel, a parallel projection is obtained.

Once a three-dimensional object is projected onto the viewplane, it then has only two dimensions, so the remaining steps are those of the two-dimensional viewing transformation: the projected primitives are clipped to a window on the viewplane, the window is mapped to the viewport, and the viewport is displayed on the display device.

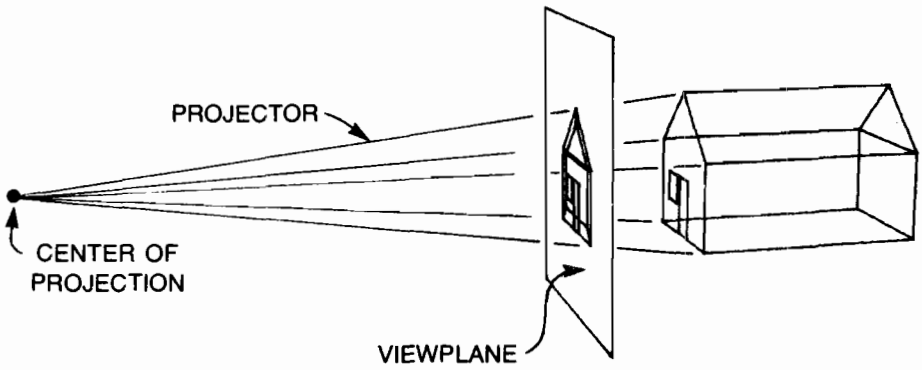


**Figure 4-4. Components of a Projection**

A step has been omitted, however. In addition to window clipping in the viewplane, AGP has depth clipping. Depth clipping clips objects to the region between two planes, the hither plane and the yon plane, which are parallel to the viewplane. Primitives lying in front of the hither plane and beyond the yon plane are eliminated from view and not projected onto the viewplane.

Together with the window, these planes define a *clipping volume*. This is a region of the world coordinate system outside of which primitives will not be seen on the display device. In the Figure 4-5, the clipping volume includes the entire house.

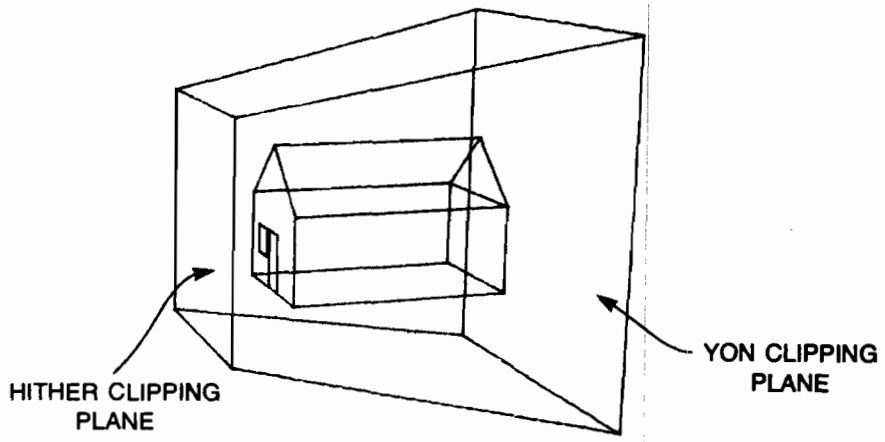


**Figure 4-5. The Clipping Volume**

## Viewplane

| | |
|---|---|
| JVREF (X,Y,Z) | Define the view reference point |
| JVDIS (DIST) | Set viewplane distance |
| JVPLN (XNRM,YNRM,ZNRM,XUP,YUP,ZUP) | Orient the viewplane |

The first step in the three-dimensional viewing transformation is to establish a view reference point with a call to JVREF. The view reference point is a world coordinate point on or near the object being viewed. It can be the center of interest, or near the center of interest of the picture. The viewing transformation is set in relation to the view reference point, facilitating later viewing modifications by simply changing its location.

Once the view reference point is set, the viewplane must be positioned. This requires two steps. First, the viewplane distance is specified with JVDIS: how far is the viewplane from the view reference point? Next, the

viewplane normal is set with JVPLN: what vector is the viewplane perpendicular to? The viewplane normal extends from the viewplane through the view reference point parallel to the direction vector (XNRM,YNRM,ZNRM). Together, the viewplane distance and viewplane normal completely determine the viewplane's position.

At this point a third coordinate system, called the "viewing coordinate system", must be introduced. The viewing coordinate system is a three-dimensional system composed of a U-axis, a V-axis, and an N-axis. These axes are established and oriented in a step-by-step fashion. The first step, positioning the N-axis, has already been taken. The N-axis is simply the viewplane normal itself. The origin of the viewing coordinate system is the point where the viewplane normal pierces the viewplane. To orient the two remaining axes requires one more vector: the view up vector.

Which way is up? This is what the view up vector tells AGP. Think of the window which will be used later: the view up vector keeps the window "upright" in the viewplane, affecting the image on a display device as shown in Figure 4-6.

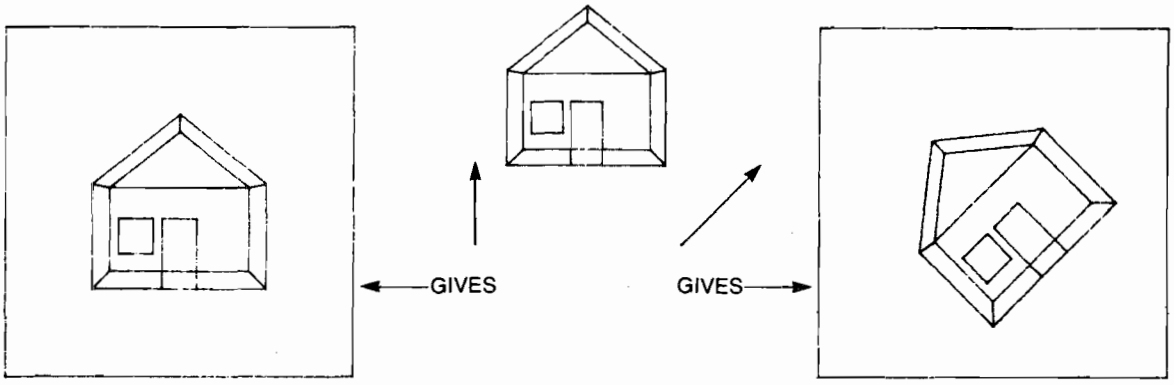


**Figure 4-6. Effect of the View Up Vector**

The view up vector extends from the view reference point parallel to the direction vector (XUP,YUP,ZUP). AGP projects it onto the viewplane to get the V-axis of the viewing coordinate system. (The view up vector and viewplane normal may not be parallel, or the projection will not produce a V-axis.)

Once the N and V-axes have been determined, the U-axis falls out quite nicely: it is the only possible axis that is perpendicular to the other two so as to form a left-handed coordinate system. The viewing coordinate system is thus fully defined as shown in Figure 4-7. The positive N-axis is the viewplane normal, the positive V-axis is the projected view up vector, and the U-axis is perpendicular to both the N and V-axes so as to form a left-handed coordinate system. The origin is the point where the viewplane normal pierces the viewplane.

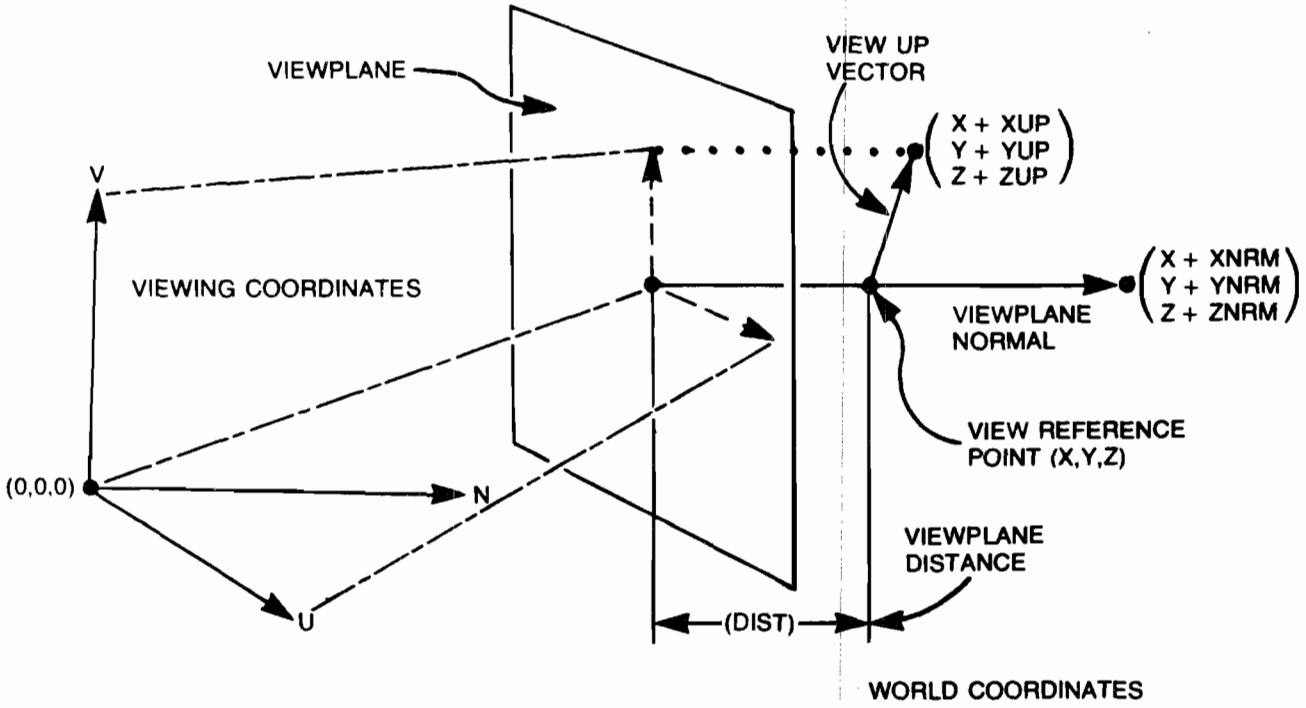


**Figure 4-7. The Viewing Coordinate System**

The Figure 4-8 shows how the viewplane distance and viewplane normal may be manipulated. The subroutine HOUSE creates a three-dimensional house. A previously created house was two-dimensional; it is used as the front and back of this new version. Additional line primitives form the sides, window and door to complete the version shown here. The window and viewport are set at the default values. Therefore, the world coordinates are such that the X and Y values range from -1.0 to 1.0.

```
CALL JVREF (0.0,0.0,0.0)              * world coordinate origin
                                      *  is view reference point
CALL JVDIS (1.0)                      * viewplane is 1 unit away
CALL JVPLN (0.0,-0.3,1.0,0.0,1.0,0.)  * normal gives a good angle
                                      *  and view up is Y axis
CALL HOUSE                            * draw a 3D house
      .
      .
      .
```

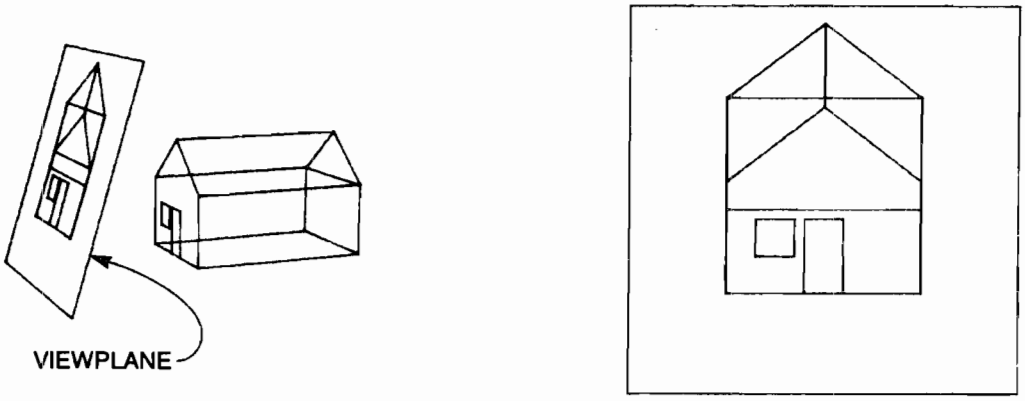


**Figure 4-8. A Three-Dimensional House**

Initially, the viewplane normal is (0.0,0.0,1.0) and the view up vector is (0.0,1.0,0.0). Since the default view reference point is the world coordinate origin and the default view distance is 0.0, the viewing coordinate system is intitially coincident with the world coordinate system. This lets two-dimensional applications make the simplifying assumption that the viewing coordinate system *is* the world coordinate system.

## Projection

JPROJ (OPCODE,DU,DV,DN) Specify projection type and center

The type of projection may be either parallel (OPCODE = 0) or perspective (OPCODE = 1). Recall that parallel projections are really just a special case of perspective projections, where the center of projection is infinitely far from the viewplane. OPCODE distinguishes between these two cases to allow use of a specialized, more efficient algorithm for parallel projections (see Figure 4-9).

```
CALL JVDIS (1.0)            * set viewplane distance of 1 unit
CALL JPROJ (1,0.0,0.0,-4.0) * select perspective projection to
                            *  a point 4 units from the viewplane
CALL HOUSE                  * draw a 3D house
```

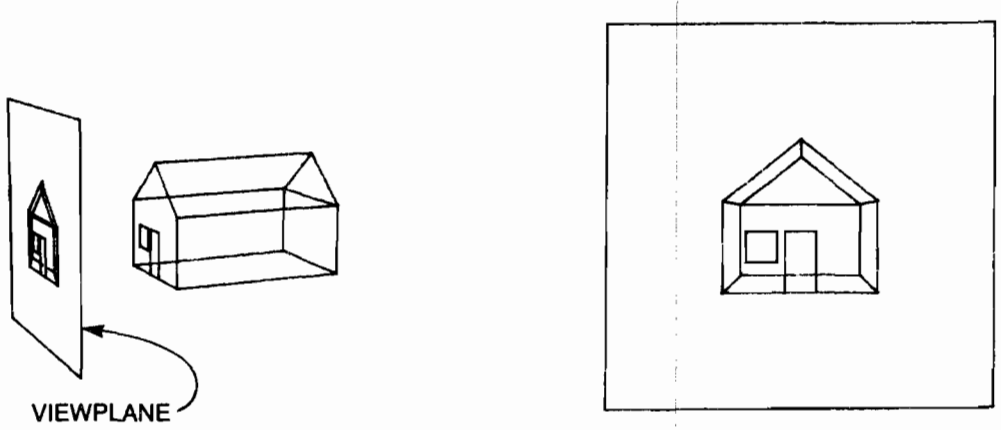


**Figure 4-9. Perspective Projection of House**

The viewing coordinate system is used to specify the next feature of the viewing transformation: the center of projection or parallel projector. When the projection type is perspective, (DU,DV,DN) are the coordinates of a point in the viewing coordinate system which defines the center of projection. This is the point to which projectors from each point on the object converge.

When the projection type is parallel, the projectors are actually parallel to a line between (DU,DV,DN) and the origin of the viewing coordinate system.

When DU and DV are both zero, the projectors are perpendicular to the viewplane, forming a true parallel projection. When one or both of DU and DV are non-zero, an oblique parallel projection is defined, meaning the projectors intersect the viewplane at an oblique angle (see Figure 4-10).

Where can the center of projection or parallel projector be located? It must be in front of the viewplane - that is, on the opposite side of the viewplane from the object to be viewed (and therefore the view reference point). Otherwise, the projectors would not pass through the viewplane, and no image would be formed. This translates to the rule, DN must always be negative.

```
CALL JVDIS (1.0)              * set viewplane distance of 1 unit
CALL JPROJ (0,0.3,0.3,-1.0)   * oblique parallel projection with
                              *   projectors at 45 degree angle
                              *   to all axes
CALL HOUSE                    * draw a 3D house
```

The initial projection is parallel, with center of projection (0.0,0.0,-1.0). Interestingly, this makes three-dimensional objects that are symmetric in the Z direction appear two-dimensional, because the primitives farther away from the viewplane are hidden by those nearer it. Changing the projection type to perspective will allow all primitives to be seen.

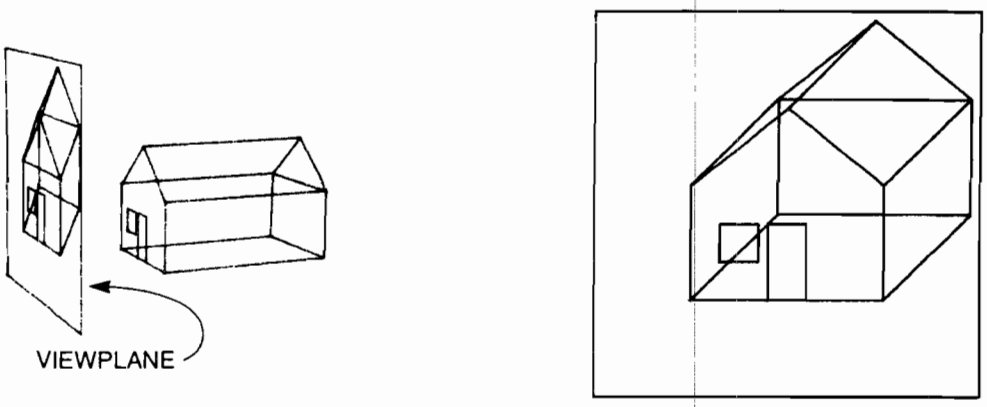


**Figure 4-10. Oblique Parallel Projection of House**

## Depth Clipping

| | |
|---|---|
| JCLPD (HITHR,YON) | Control hither and yon clipping |
| JDPTH (HDIST,YDIST) | Position hither and yon clipping planes |

JCLPD allows the application program to enable or disable hither and yon clipping. JDPTH determines where that clipping will occur. The depth clipping planes are always parallel to the viewplane; HDIST and YDIST specify the distances of the hither and yon planes, respectively, from the viewplane in viewing coordinates. The hither and yon planes are always on the opposite side of the viewplane from the center of projection, and the yon plane is always farther from the viewplane than the hither plane as shown in Figure 4-11.

```
CALL JVREF (0.0, 0.0, 1.0)      * place view reference point
CALL JVDIS (2.0)                *  and set viewplane distance
CALL JPROJ (1, 0.0, 0.0,-4.0)   * projection is perspective
CALL JCLPD (1, 1)               * turn on hither and yon clipping
CALL JDPTH (0.0, 2.0)           * set distances from viewplane
CALL HOUSE                      * draw house, back third clipped
```

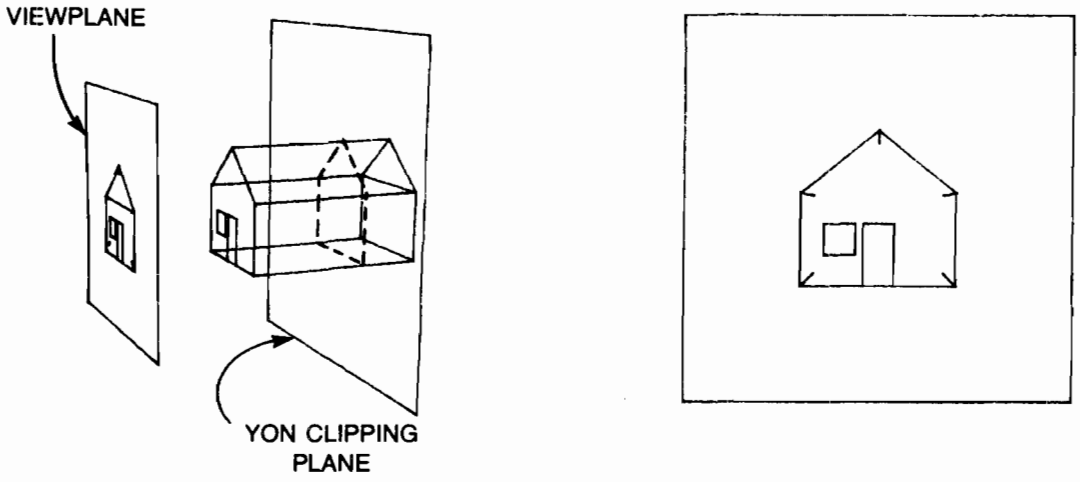


**Figure 4-11. House With Yon Clipping**

At AGP initialization, hither clipping is enabled while yon is disabled. Both HDIST and YDIST are 0.0, meaning hither clipping occurs behind the viewplane. Yon clipping does not occur because it is disabled. These defaults are chosen so as to produce the most straightforward results for a new user.

Why is hither clipping typically enabled? Recall the way the three-dimensional viewing transformation is achieved: projectors passing from an object behind the viewplane to a center of projection in front of it. If some of the object's primitives lie in front of the viewplane, the projectors will not all pass through the viewplane; according to the model, no image should be produced by these primitives. Hither clipping at the viewplane will have exactly this effect. On the other hand, if hither clipping were disabled, unexpected results might be produced. For this reason, hither clipping is usually enabled.

## Three-Dimensional Window

| | |
|---|---|
| JWIND (DWUMIN,DWUMAX,DWVMIN,DWVMAX) | Set window |
| JCLPW (OPCODE) | Control window clipping |

After primitives lying between the depth clipping planes are projected onto the viewplane, they are clipped to the window. The window was discussed previously. It is repeated here to show its relationship to depth clipping and to clarify its role in the three-dimensional viewing transformation.

In two-dimensional applications the window was set in the X Y world coordinate plane which was coincident with the viewplane. However, using JVDIS and JVPLN, three-dimensional applications establish a viewplane that is, in general, not coincident with the world coordinate X Y plane. Thus, in the three-dimensional case, the window cannot be defined in world coordinate units.

The placement of the window within the viewplane is relative to the point where a projector from the view reference point pierces the viewplane as shown in Figure 4-12. For a perspective projection DWUMIN, DWUMAX, DWVMIN, and DWVMAX are specified relative to the point where the projector from the view reference point to the center of projection pierces the viewplane.

For a parallel projection DWUMIN, DWUMAX, DWVMIN, and DWVMAX are specified relative to the point where a parallel projector passing through the view reference point pierces the viewplane.

The effect of locating the window in this manner is that the view reference point will always be visible if DWUMIN and DWVMIN are negative displacements and DWUMAX and DWVMAX are positive. Additionally, if DWUMIN = - DWUMAX and DWVMIN = - DWVMAX, then the view reference point will always appear in the center of the window. Thus, to center an object in the window, place the view reference point on the object and specify the window to be symmetric.

Primitives that are not clipped by the hither or yon planes are subjected to one more clipping operations if window clipping is enabled. They must lie within the window to be displayed. Once the projected primitives have been clipped to the window, the viewing transformation is the same as that for a two-dimensional viewing transformation: the window is mapped to the viewport and the viewport is displayed on the display device.

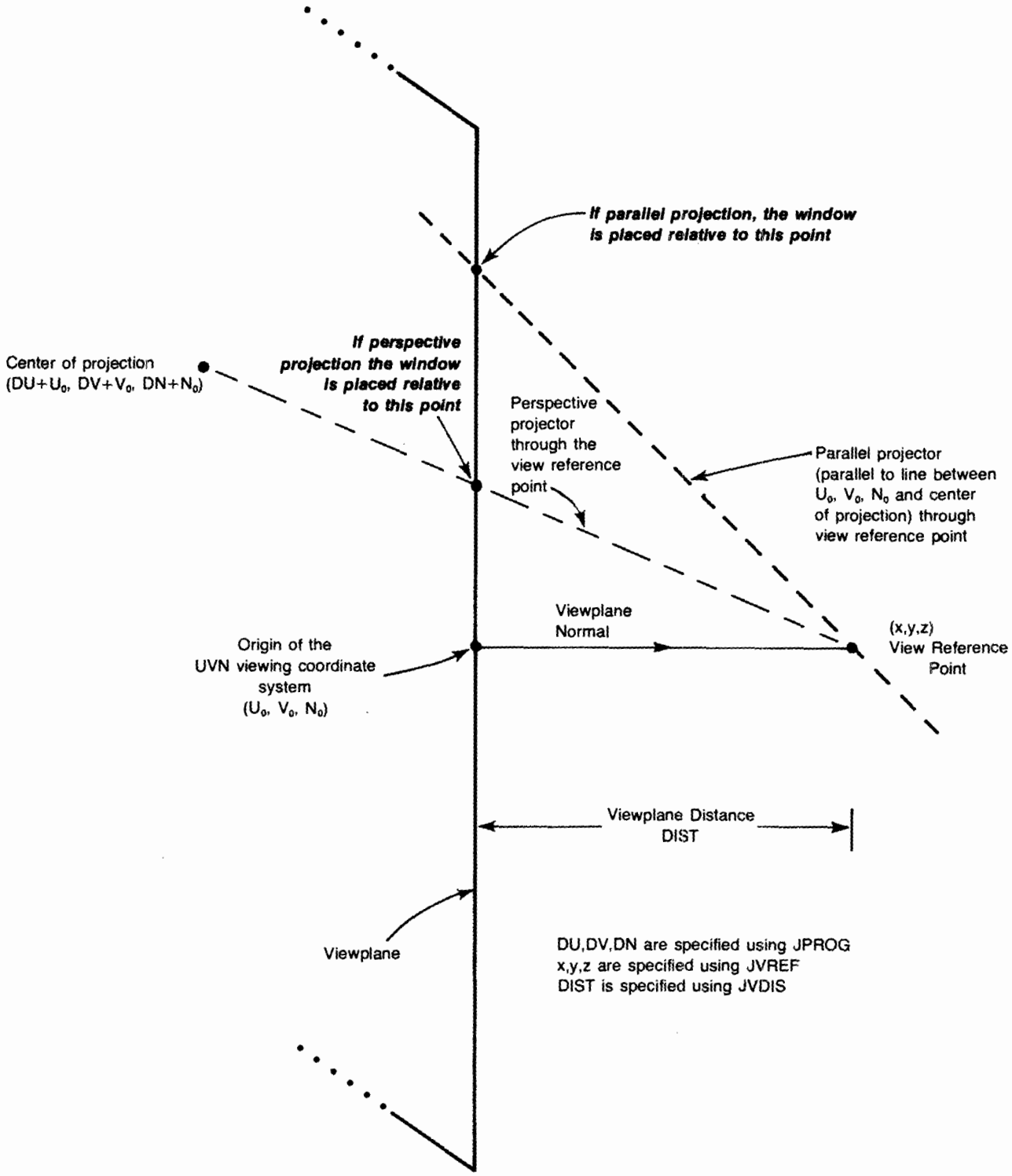


**Figure 4-12. Placement of the Window on the Viewplane**

# ADVANCED CAPABILITIES

Once the previously described viewing functions are understood, the remaining few should be examined. To see why they are needed, consider the following situation.

A poster entitled *Flight* is to be created with AGP. The principal object in the poster will be an airplane, for which the data is already available. Initially, the problem seems straightforward, however, consider the following:

1. The data for the airplane was created using a right-handed world coordinate system.

2. The data consists of absolute points; that is, it draws the airplane centered around the world coordinate origin. You would like, however, to be able to draw the airplane located anywhere in the world coordinate system.

3. A copy of the airplane only looks good if its wingspan is at least 10% of the width of the viewport. You would like to detect when an airplane's image will be smaller than this.

4. You revise your "size criteria" in (3) to: the airplane only looks good if its wingspan is at least 15 mm. You would like to detect when this is true.

5. Once the airplane has been drawn, you would like to restore the initial viewing transformation so text can be drawn on the poster.

Some of these could be done with a combination of previously described functions; others could not. All would be simply realized with the ability to:

1. set the handedness of the world coordinate system,

2. "translate" an object to some other location,

3. convert world coordinates to virtual coordinates,

4. convert virtual coordinates to millimeters on a particular display device, and

5. reset the viewing transformation to the initial values.

These capabilities are described in the following sections.

## Handedness

JHAND (OPCODE) Set handedness of world coordinate system

As defined previously, the world coordinate system is left-handed, meaning the positive Z axis extends into the page as in Figure 4-13.

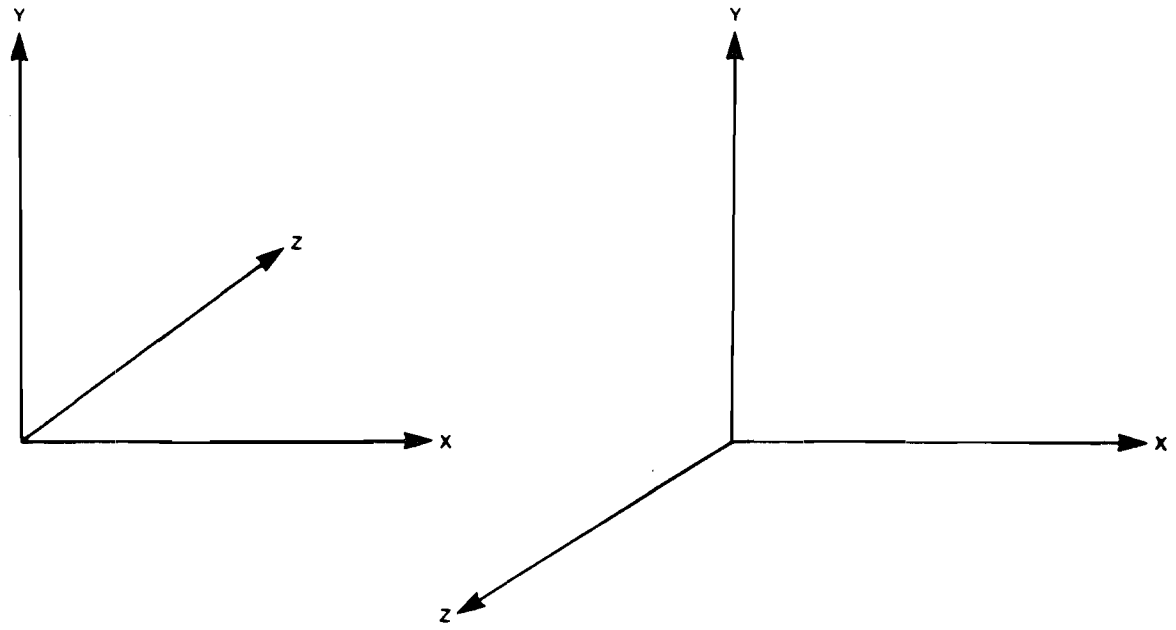


**Figure 4-13. Left and Right Handedness**

JHAND may be used to make it left-handed (OPCODE = 0) or right-handed (OPCODE = 1). A right-handed world coordinate system would have the positive Z axis extending out of the page, toward the user (also shown in Figure 4-13).

It is worth noting that the handedness of the world coordinate system has no effect on the viewing or virtual coordinate systems. The viewing coordinate system is always left-handed. Since the virtual coordinate system is two-dimensional, handedness does not apply.

## Modelling

| | |
|---|---|
| JDMOD (MATRIX) | Define the 4-by-4 modelling matrix |
| JCMOD (OPCODE) | Control application of the modelling matrix |

Many graphics applications require the ability to manipulate an object before it is viewed. This might consist of scaling (making it bigger or smaller), translating (displacing it as a unit to some other position in the world coordinate system) or rotating (with respect to the world axes). These actions are typically part of a *modelling system*. A modelling system can be loosely defined as a way of manipulating objects as a whole prior to applying a viewing transformation. In the graphics world, a modelling system is usually thought of as a complete software and hardware package that provides for generation, storage, and output of complex graphical models of the real world. Although AGP can play a very useful role, it is not all that is required in such a system.

The interface between a modelling system and AGP is provided by JDMOD and JCMOD. JDMOD allows the definition of a 4-by-4 matrix which will be applied to primitives when modelling is enabled with JCMOD (OPCODE = 1). This matrix may, for example, scale, translate, or rotate subsequently created primitives before the viewing transformation is applied. Additional software to build and manage the modelling matrix would form another part of the modelling system.

The (row,column) indices of the modelling matrix are assumed to be:

| | | | | | |
|---|---|---|---|---|---|
| *Linear Translation* | (1,1) | (1,2) | (1,3) | (1,4) | *Perspective* |
| *including local scaling* | (2,1) | (2,2) | (2,3) | (2,4) | *Translation* |
| | (3,1) | (3,2) | (3,3) | (3,4) | |
| *Translation* | (4,1) | (4,2) | (4,3) | (4,4) | *Overall Scaling* |

The following matrices show the role of individual elements of the matrix. Scaling by s:

| | | | |
|---|---|---|---|
| s | 0 | 0 | 0 |
| 0 | s | 0 | 0 |
| 0 | 0 | s | 0 |
| 0 | 0 | 0 | 1 |

If, for example, s = 2.0, the result is a two-fold increase in size.

Translation of an object by (m,n,p):

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| m | n | p | 1 |

Rotation through the origin of A degrees about the X axis:

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | Cos A | -Sin A | 0 |
| 0 | Sin A | Cos A | 0 |
| 0 | 0 | 0 | 1 |

Rotation through the origin of B degrees about the Y axis:

| | | | |
|---|---|---|---|
| Cos B | 0 | Sin B | 0 |
| 0 | 1 | 0 | 0 |
| -Sin B | 0 | Cos B | 0 |
| 0 | 0 | 0 | 1 |

Rotation through the origin of C degrees about the Z axis:

| | | | |
|---|---|---|---|
| Cos C | -Sin C | 0 | 0 |
| Sin C | Cos C | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

Combinations of these operations can be performed by matrix multiplications of the above examples. At initialization, the modelling matrix is the identity matrix and modelling is disabled.

## Conversion Between World and Virtual Coordinates

| | |
|---|---|
| JVTOW (VX,VY,X,Y,Z) | Convert virtual to world coordinates |
| JWTOV (X,Y,Z,VX,VY) | Convert world to virtual coordinates |

The conversion routines JVTOW (virtual to world) and JWTOV (world to virtual) convert a point in one coordinate system to another. For JVTOW the world coordinates corresponding to the virtual point (VX,VY) are calculated using the inverse of the current viewing transformation. The point (X,Y,Z) is always computed to lie in the viewplane. For JWTOV the virtual coordinates corresponding to the world point (X,Y,Z) are returned in (VX,VY). No clipping is performed in this conversion, so (VX,VY) may lie outside of the virtual coordinate rectangle.

## Display Device Conversion

| | |
|---|---|
| JDPMM (ID,VX,VY,XMM,YMM) | Convert a virtual point to millimeters on the display device |

JDPMM allows conversion of a virtual coordinate point (VX,VY) to a point in millimeters (XMM,YMM) on the display device of work station ID. Used together with JWTOV, it lets the user determine the size image an object will produce. That is, knowing only the object's world coordinates, the size in millimeters of the resulting image can be programmatically determined. This is very useful in many applications (e.g., those requiring scaled drawings).

In the following program fragment, the length of a line primitive is being determined before it is output. If the X component is less than 1.0 millimeters in length, it is not drawn.

```
XW1 = -0.5                              * x-coordinate of point 1
XW2 = 0.5                               * x-coordinate of point 2
CALL JWTOV (XW1,0.0,0.0,VX1,VY1)        * find virtual point 1 and
CALL JDPMM (1,VX1,VY1,XMM1,YMM1)        *  convert to mm. on WS 1
CALL JWTOV (XW2,0.0,0.0,VX2,VY2)        * find virtual point 2 and
CALL JDPMM (1,VX2,VY2,XMM2,YMM2)        *  convert to mm. on WS 1
IF (ABS(XMM2-XMM1) .LE. 1.0) GO TO ...  * if they are too close...
CALL J3MOV (XW1,0.0,0.0)                * not too close:
CALL J3DRW (XW2,0.0,0.0)                * draw the line
```

## Resetting the Viewing Transformation

JRSET

Within a program, it is convenient to be able to reset the viewing transformation to a known state in one call. JRSET does this. The viewing transformation is reset to the initial viewing state, with a few differences: handedness, logical display limits, aspect ratio, and viewport are not reset to their initial values.

Table 4-2. Initial Values of Viewing Transformation Components.

| Viewing Component | Initial Value |
|---|---|
| View Reference Point | (0.0,0.0,0.0)<br>{in world coordinates} |
| Viewplane Distance | (0.0)<br>{between view ref. point and viewplane} |
| Viewplane Normal | (0.0,0.0,1.0)<br>{relative to viewing coordinate origin} |
| View Up Vector | (0.0,1.0,0.0)<br>{relative to viewing coordinate origin} |
| Projection | parallel to (0.0,0.0,-1.0)<br>{perpendicular to the X Y plane} |
| Window Clipping | enabled at (-1.0,1.0,-1.0,1.0)<br>{relative to projection of view reference point onto viewplane} |
| Hither Clipping | enabled at (0.0)<br>{relative to the viewplane} |
| Yon Clipping | disabled at (0.0)<br>{relative to the viewplane} |
| Modelling | disabled with matrix set to identity |

The handedness of the world coordinate system, the logical display limits currently in effect, and the aspect ratio of the virtual coordinate system remain at their current values. However, the viewport is neither reset to

its initial value (a unit square), nor is it left as is, but is reset to the limits of the virtual coordinate system. Recall that this is exactly the effect of a JASPK call.

JRSET returns us (nearly) to the place we began: the initial viewing transformation.

# Chapter 5
# Segments and Primitives Outside of Segments

## GENERAL

As graphics applications become more sophisticated, the ability to subdivide complex graphics pictures into logical entities, or segments, becomes valuable. A segment is a named group of output primitives that define an image.

Segments are actually stored in a data structure in the work station program called the segment display area, SDA (see Figure 5-1). The segment display area is simply an in-memory version of the picture on the display device. Without it, once a picture was drawn, the computer would "forget" about it - that is, only the device would have the data with which to recreate the picture. Of course, the application program could redraw the entire picture from scratch. However, that would be a time-consuming process if only one segment were to be modified.

Primitives do not have to be included in segments. In some applications, there is no need for picture modification capabilities provided by segmentation. The primitives' images could simply be drawn on the display device without saving them in a segment. Indeed, primitives outside of segments have been used in all the examples described previously. Note that primitives outside of segments are temporary in nature. They exist on a display device only until a new-frame-action, which removes all primitives not contained in visible segments.

It is important to recognize that AGP includes no capability for copying a segment into other segments. Also, no segment can reference or call another segment, as in a picture/sub-picture hierarchy. Thus, there is only one level of segmentation for defining a picture.

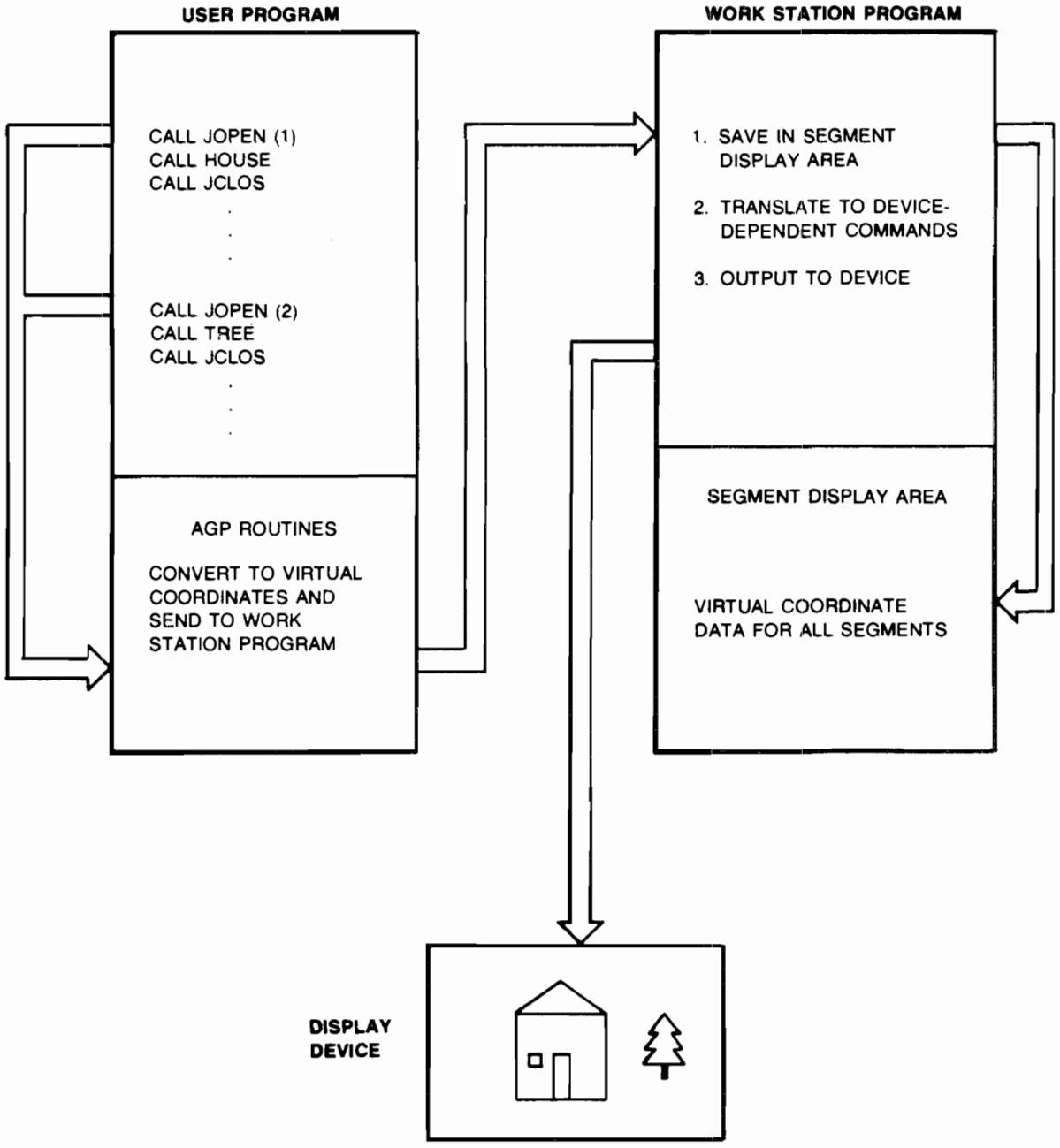


**Figure 5-1. Segment Display Area**

## SEGMENT CREATION, RENAMING, AND DELETION

| | |
|---|---|
| JOPEN (NAME) | Create/open a segment |
| JCLOS | Close the segment |
| JRNAM (ONAME,NNAME) | Rename an existing segment |
| JPURG (NAME) | Purge an existing segment |
| JCLR | Purge all existing segments |

JOPEN creates and opens a new segment. NAME, an integer, is used to refer to the segment in future calls. The segment is "bound" to each enabled work station; that is, it is saved in the segment display area of each work station being used. JCLOS closes a segment, meaning no more output primitives may be added to it. Once closed, a segment may never be opened again. A segment may be renamed with JRNAM from ONAME to NNAME, and it may be purged from the segment display area and display device of every work station with JPURG. JCLR purges *all* segments, effectively emptying segment display areas and clearing display devices.

The following program fragment shows a house segment being created, renamed and purged.

```
CALL JOPEN (1)              * open segment 1
CALL HOUSE                  * draw a house
CALL JCLOS                  * close segment 1
     :
CALL JRNAM (1,23)           * now the house is segment 23
     :
CALL JPURG (23)             * remove the house segment
     :
```

## ATTRIBUTES OF SEGMENTS

| | |
|---|---|
| JGHI (OPCODE) | Set system-maintained highlighting attribute |
| JSHI (NAME,OPCODE) | Set a segment's highlighting attribute |
| JGDET (OPCODE) | Set system-maintained detectability attribute |
| JSDET (NAME,OPCODE) | Set a segment's detectability attribute |
| JGVIS (OPCODE) | Set system-maintained visibility attribute |
| JSVIS (NAME,OPCODE) | Set a segment's visibility attribute |
| JVSAL (OPCODE) | Set the visibility of all segments |

Segment attributes affect the appearance of the image stored in a segment. Two segment attributes make the segment's image visible or invisible, highlighted (to draw the operator's attention to it) ,or non-highlighted. A third segment attribute controls whether the pick input device can be used

to select primitives in a segment. This attribute has the values detectable and non-detectable.

Attributes are divided into two classes: output primitive attributes and segment attributes. Output primitive attributes are static; they may not be changed after the primitive is created. An application program can only change an attribute of an output primitive by deleting the segment that contains the primitive, changing the system-maintained attribute value, and then creating a new primitive. This is not necessary for segment attributes. In fact, a primary reason for grouping primitives into segments is so that their segment attributes may be dynamically changed. As with output primitive attributes, system-maintained values for segment attributes are assigned when a segment is created. These system-maintained values are established with the JGxxx routines. During a segment's construction, as well as after it is closed, the values of its attributes may be changed from the original values with the JSxxx routines.

In these routines, OPCODE = 1 turns *on* an attribute, while OPCODE = 0 turns that attribute *off*. NAME is the segment name specified when it was opened. While all but one of these calls treat segments individually, JVSAL sets the visibility attribute of *all* existing segments, which may be useful to temporarily clear the display device.

The following example shows an invisible house segment being created. After some intermediate processing, the house is made visible.

```
CALL JGVIS (0)          * subsequent segments will be invisible
CALL JOPEN (23)         * create a segment
CALL HOUSE              *   that contains a house
CALL JCLOS              * nothing appears on display device
      :
      :
CALL JSVIS (23,1)       * now the house appears
```

Segments are also useful for picking. Each output primitive in a segment has two components that are reported when that primitive is picked. The first is the name of the segment containing the primitive. The second component is the name of the pick-ID when the primitive was created. Pick-ID is an output primitive attribute set by JPKID (see Chapter 3).

Pick-IDs are useful in menus. A menu is a list of options presented to an interactive user, who chooses one of the menu options with a pick device. This menu is a single segment, with each option labeled by a separate pick-ID. When the user selects one of the menu options, the segment name of the menu and the pick-ID of the selected option are returned, as shown in the following example.

```
CALL JOPEN (9)                  * open the menu segment
CALL JPKID (1)                  * create option 1
```

```
CALL J2MOV (0.0, 0.8)                 *   set current position
CALL JTEXM (12, 12H Jelly Beans)      *   and output text
CALL JPKID (2)                        * create option 2
CALL J2MOV (0.0, 0.6)                 *   set current position
CALL JTEXM (12, 12H   Gum Drops)      *   and output text
CALL JPKID (3)                        * create option 3
CALL J2MOV (0.0, 0.4)                 *   set current position
CALL JTEXM (12, 12H Candy Canes)      *   and output text
CALL JCLOS                            * close the menu segment
       :
       :
CALL JPICK (1,1,IBUTN,NAME,PICKID)    * issue a pick request
IF (NAME .NE. 9) GO TO 1000           * check for menu segment
IF (PICKID .EQ. 1) GO TO 110          * is it option 1?
IF (PICKID .EQ. 2) GO TO 120          * is it option 2?
IF (PICKID .EQ. 3) GO TO 130          * is it option 3?
       :
```

The pick-ID attribute is also useful for picking data points within a data plot.

## NEW-FRAME-ACTION

JNEWF — Cause a new-frame-action

To erase one of several images on a display device, many devices must be cleared entirely and then redrawn with only the images that are to remain. This is equivalent to erasing part of a chalkboard by erasing the whole thing, then re-writing the part that was not to be erased. Although this is certainly an awkward way to erase a chalkboard, many display devices function in this manner (for example, a plotter); therefore, AGP must treat them appropriately. The mechanism it uses is called a new-frame-action.

A new-frame-action erases all primitives outside of segments from a display device. If no segments have been used at all, it effectively clears the display device. The effect of a new-frame-action depends on the particular device being used. On chart advance plotters the page is advanced before the visible segments are re-drawn. On a terminal, the screen might be cleared then the segments re-drawn, or the terminal may be able to remove primitives outside of segments explicitly without clearing its screen.

JNEWF causes a new-frame-action on every enabled work station. New-frame-actions also occur on individual work stations when:

1. a visible segment is purged (JPURG or JCLR)
2. a visible segment is made invisible (JSVIS)
3. the highlighting of a visible segment is changed (JSHI)
4. the logical display limits are redefined (JDLIM)

# Chapter 6
# Input

## GENERAL

Previous chapters have been concerned with the AGP functions that generate output. This chapter is concerned with the opposite process: receiving input.

AGP applications may receive input from the following devices:

1. Button
2. Keyboard
3. Locator
4. Valuator
5. Pick

A button device returns an integer corresponding to a particular button that is pushed, a keyboard returns an alphanumeric string, a locator returns a real coordinate pair, a valuator returns a single real value and a pick device returns a segment name and pick-ID. Refer to Chapter 2 for a review of the nature of these devices.

Two mechanisms exist for obtaining data from input devices: requesting and sampling. Requested input implies that the program will wait until the operator enters a termination command (e.g., presses a key) before continuing. All AGP input devices may be requested. Sampling means no operator response is needed: the current value of the input device is returned without delay. Only the locator and valuator devices may be sampled; the other input devices are by nature unsuitable for sampling.

Concurrent graphics output to multiple work stations is fundamental to AGP. By enabling more than one work station, the same image may be seen simultaneously on several display devices without the application having to create or send it multiple times. This is not the case with input. In general, the responses of an operator must be handled individually by the application. For this reason, each of the calls described in this chapter addresses a specific workstation; that is, each has an ID parameter in which the desired work station is specified.

A single work station may contain, at most, one each of the five logical input devices. More than one logical device may be found in a single physical device. For example, the HP 2623A Graphics Terminal can be used for both keyboard and locator input. Some input devices which are not physically available at a particular work station may be simulated using software and another device. A valuator is simulated on the HP 2623A by the X coordinate (or the Y coordinate) of the graphics cursor.

Most input devices use echoing. Echoing gives operator feedback during an input operation, reflecting its status to allow refinement before it is completed. The characters that appear on a terminal screen when a key is pressed are a familiar example of echoing. In most systems, these characters have actually been echoed by the computer after it has received them from the keyboard. By seeing what is typed, an operator may correct any typographical errors before terminating the input operation by pressing return. With AGP, more elaborate echoing than this is possible.

There are two types of echoes: those on the input device itself and those on the graphics display device. All input functions may be echoed on the input device (depending, of course, on the device's echoing capabilities). However, only locator and pick input may be echoed on the display device. These echoes begin at a user-settable point called the locator or pick echo position. Some display device echoes also utilize this point throughout the echo. For example, when a rubber-band-line echo is used with locator input, the fixed end of the rubber-band-line begins at the locator echo position (see Figure 6-1).

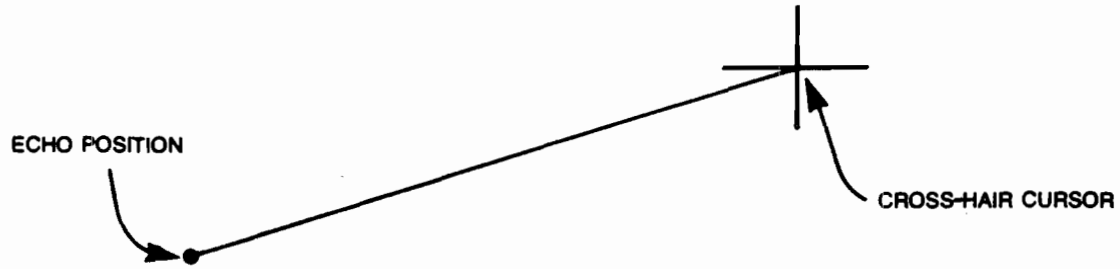


**Figure 6-1. Rubber Band Line Echo**

Just as a portion of the display device was selectable for graphics output by setting logical display limits, portions of the locator and pick devices may be selected for input by setting logical locator and pick limits. These limits allow the range of possible operator inputs to be controlled. As shown in Figure 6-2, the logical locator and pick limits map directly to the display surface.

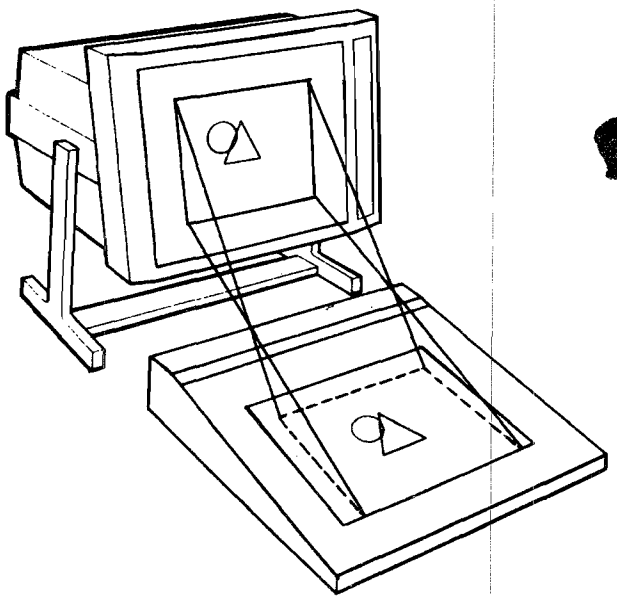



**Figure 6-2. Logical Locator or Pick Limits Mapping**

Only those points inside the locator limits are eligible to be located, and only those segments within the pick limits are eligible to be picked.

## BUTTON

JBUTN (ID,ECHO,VALUE) Await button press, report value

JBUTN requests button input from the work station ID. It waits until a button is pressed, then returns the button number in VALUE. If a key other than a valid button key is pressed, 0 is returned.

```
INTEGER VALUE                    * buttons are always integers
        :
        :
        :
CALL JBUTN (1,0,VALUE)           * button input from
                                 *   Work Station 1, no echo
IF (VALUE .EQ. 1) GO TO 100      * button 1 was pushed
IF (VALUE .EQ. 2) GO TO 200      * button 2 was pushed
IF (VALUE .EQ. 3) GO TO 300      * button 3 was pushed
        :                        * if here bad button was pushed
```

Echoing, as controlled by the ECHO parameter, depends on the capabilities of the input and display device being used. Possibilities include ringing a bell or echoing an integer corresponding to the button that was pressed.

## KEYBOARD

JKYBD (ID,ECHO,MAX,ACTUAL,STRING)  Request text entry, report va

JKYBD requests keyboard input from work station ID. It waits until a line of text has been entered on the keyboard and terminated (e.g., by a carriage return), then returns the text in the array STRING. The characters will be in packed ASCII format (that is, ASCII left justified, one character per byte and blank filled.). Up to MAX characters will be allowed, and the actual number entered will be returned in ACTUAL.

As with button input, echoing is device-dependent.

```
INTEGER ACTUAL, STRING(40)       * 40 integers = 80 characters
        :
        :
CALL JKYBD (1,1,80,ACTUAL,STRING)  * request up to 80 characters
                                 *   from Work Station 1
                                 *      with echo 1
CALL JTEXL (ACTUAL,STRING)       * output as low quality text
        :
```

# LOCATOR

| | |
|---|---|
| JWLOC (ID,ECHO,BUTTON,VX,VY) | Request locator input, report value |
| JSLOC (ID,ECHO,VX,VY) | Sample locator, report value |
| JLLIM (ID,XMIN,XMAX,YMIN,YMAX) | Set limits on the locator device |
| JLOCP (ID,PX,PY) | Set the locator echo position |
| JLPMM (ID,PX,PY,XMM,YMM) | Convert virtual point to millimeters |

JWLOC and JSLOC invoke locator input. JWLOC waits for a locator button to be pressed before returning to the application. JSLOC returns the current value of the locator without delay. Each locator device has its own set of buttons that may or may not be the same buttons that make up the button device.

The locator returns a virtual coordinate point (VX,VY). The corresponding world coordinate point may be derived with JVTOW, as shown below.

```
CALL JWLOC (1,1,BUTTON,VX,VY)      * request locator input from
                                   *   work station 1 with echo 1
CALL JVTOW (VX,VY,X,Y,Z)           * convert to world coordinates
```

Note that the point (X,Y,Z) will always be found in the viewplane. This assumption must be made for JVTOW to be able to derive three coordinates from two.

Depending on the capabilities of the devices being used, locator echoes are possible on either the locator or display device.

JLLIM sets the logical limits of the locator device. The logical locator limits are initially defined as the addressable area of the locator device. With a call to JLLIM, the user can set them to a new rectangle on the locator device. The pairs (XMIN,YMIN) and (XMAX,YMAX) define the corner points of this rectangle in terms of millimeters of offset from the origin of the device. (The position of this origin is device-dependent and is documented in the *Device Handlers Manual*.)

JLLIM does not affect the virtual coordinate system. It affects only the mapping between the virtual coordinate system and the locator device.

If the locator device and the display device are identical, then the logical locator limits and the logical display limits must be identical. Specifically, when they refer to the same device, the interaction between JDLIM and JLLIM is as follows:

1. The logical locator limits are initialized to the same values as the logical display limits when the work station is initialized with a call to JDINT.

2. A call to JDLIM implicitly sets the logical locator limits to the same values.

3. Explicit calls to JLLIM cause an error to be generated. The call is ignored.

The locator echo position is a point in the virtual coordinate system used with locator echoing on the graphics display device. For example, the fixed end of a rubber-band-line echo will be at the locator echo position. JLOCP sets that position for all locator echoes on the display device. The default locator echo position is in the center of the virtual coordinate system. When a call is made that changes either the viewing transformation or the mapping between the display device and locator device, the locator echo position is reset to the default value. The calls which do this are JASPK, JDLIM, and JLLIM.

JLPMM converts a virtual point to millimeters on the locator device. The virtual coordinate point (PX,PY) is returned in (XMM,YMM) by mapping the virtual coordinate system onto the locator device. The returned values (XMM,YMM) are in millimeters of offset from the origin of the locator device.

The following program section shows a likely use of these functions. Locator input is used to let the operator set the logical locator limits interactively by selecting first the lower left corner then the upper right corner.

```
CALL JWLOC (1,2,BUTTON,VXMIN,VYMIN)     * request locator
                                        *   from Work Station 1
                                        *   with display device echo
CALL JLOCP (1,VXMIN,VYMIN)              * set it as echo position
CALL JWLOC (1,4,BUTTON,VXMAX,VYMAX)     * request locator input from
                                        *   Work Station 1,
                                        *   rubber band echo
CALL JLPMM (1,VXMIN,VYMIN,XMMIN,YMMIN) * convert both points to
CALL JLPMM (1,VXMAX,VYMAX,XMMAX,YMMAX) *   millimeters on locator
CALL JLLIM (1,XMMIN,XMMAX,YMMIN,YMMAX) * set locator limits to them
```

# VALUATOR

| | |
|---|---|
| JWVAL (ID,ECHO,SUBVAL,BUTTON,VALUE) | Request valuator input, report value |
| JSVAL (ID,ECHO,SUBVAL,VALUE) | Sample valuator, report value |

JWVAL and JSVAL invoke valuator input. JWVAL waits for the valuator button to be pressed before returning. JSVAL returns valuator information without delay. Each valuator device has its own set of buttons (used only in the valuator request) that may or may not be the same as those that make up the button device.

A valuator device has multiple valuators, called subvaluators. For example, the HP 2623A Graphics Terminal may use either the X coordinate *or* the Y coordinate of the graphics cursor as a valuator. SUBVAL is used to select a particular valuator to be used. Recall that a valuator device returns a real value between 0.0 and 1.0. If SUBVAL = 1, the X position of the HP 2623A's cursor is returned, with 0.0 represented as the extreme left side of the display and 1.0 represented as the extreme right side. If SUBVAL = 2, the Y position of the cursor is returned, with 0.0 represented as the bottom of the display and 1.0 represented as the top of the display.

Echoing is possible with both JWVAL and JSVAL. The type and number of echoes available is device-dependent. Most valuator devices support at least one form of echoing such as beeping, displaying the value sampled, etc.

# PICK

| | |
|---|---|
| JPICK (ID,ECHO,BUTTON,NAME,PICKID) | Request button input, return segment name and pick-ID |
| JPLIM (ID,XMIN,XMAX,YMIN,YMAX) | Set limits of the pick device |
| JPIKP (ID,PX,PY) | Set the pick echo position |
| JPPMM (ID,PX,PY,XMM,YMM) | Convert a virtual point to millimeters on the pick device |

JPICK requests pick input, returning the name of the selected segment and name of the pick-ID attached to the output primitive in NAME and PICKID. For the operator, the pick operation consists of moving a cursor or light pen near a primitive, then pressing a pick button. AGP searches the segment display area for a detectable, visible segment and primitive that is near the location chosen. If no segment can be identified, NAME and PICKID will be 0. For a complete description of the pick algorithm used, see JPICK in the *AGP Reference Manual*.

When a primitive is created within a segment, the current pick-ID is bound to it. Recall that JPKID sets that pick-ID attribute. Primitives outside of a segment are not stored in the segment display area and therefore cannot be picked.

Each pick device has its own set of buttons. These may or may not be the same set of buttons that make up the button device.

Pick input may be echoed either on the graphics display device or on the pick device. Both devices must support the echo for it to be available.

The following example was presented previously when segments were described. It shows picking being used to select choices in a menu.

```
CALL JOPEN (1)                          * open the menu segment
CALL JPKID (1)                          * create choice 1
CALL J2MOV (0.0, 0.8)                   *   set current position
CALL JTEXM (12,12H Jelly Beans)         *   and output text
CALL JPKID (2)                          * create choice 2
CALL J2MOV (0.0, 0.6)                   *   set current position
CALL JWTOV (0.0, 0.6, 0.0, VX, VY)      * translate this point to
                                        *   virtual coordinates
CALL JPIKP (1, VX, VY)                  * set pick echo position
CALL JTEXM (12,12H   Gum Drops)         * output text for choice 2
CALL JPKID (3)                          * create choice 3
CALL J2MOV (0.0, 0.4)                   *   set current position
CALL JTEXM (12,12H Candy Canes)         *   and output text
CALL JCLOS                              * close the menu segment
        :
        :
CALL JPICK (1,1,IBUTN,NAME,PICKID)      * issue a pick request
IF (NAME .NE. 1) GO TO 1000             * check for menu segment
IF (PICKID .EQ. 1) GO TO 110            * choice 1
IF (PICKID .EQ. 2) GO TO 120            * choice 2
IF (PICKID .EQ. 3) GO TO 130            * choice 3
                                        * bad choice if here
```

JPLIM specifies the logical limits of the pick device. The logical limits are initially defined as the addressable area of the pick device. With a call to JPLIM, the user can set them to a new rectangle on the pick device. The pairs (XMIN,YMIN) and (XMAX,YMAX) define the corner points of this rectangle in terms of millimeters of offset from the origin of the device.

JPLIM does not affect the virtual coordinate system. It affects only the mapping between the virtual coordinate system and the pick device.

If the pick and the graphics display are both the same physical device, then the logical pick limits and the logical display limits must be identical. Specifically, the interaction between JDLIM and JPLIM when they refer to the same device is as follows:

1. The logical pick limits are initialized to the same values as the logical display limits when the work station is initialized with a call to JDINT.

2. A call to JDLIM implicitly sets the logical pick limits to the same values.

3. Explicit calls to JPLIM are ignored.

For all pick echoes on the graphics display device, the echo originally appears at the pick echo position, a point in the virtual coordinate system set by JPIKP. Initially, it is centered in the virtual coordinate system, and is reset to there by any call that changes either the viewing transformation or the mapping between the display device and pick device. Those calls are JASPK, JDLIM, and JPLIM.

A conversion routine, JLPMM, is provided to convert virtual coordinates to millimeters on the pick device. The virtual coordinate point (PX,PY) is returned in (XMM,YMM) by mapping the virtual coordinate system onto the pick device. The returned values (XMM,YMM) are in millimeters of offset from the origin of the pick device.

# Chapter 7
# Control

## GENERAL

This chapter completes the description of AGP, with a discussion of the control functions. Some control functions initialize and modify the graphics environment. Others interactively inquire the environment's features - for example, the availability of a button input device. Still others control the timing of image generation. Beyond these general categories, escape functions and error control are also included.

## INITIALIZATION AND TERMINATION

The following functions control the AGP system, work stations, and devices in those work stations.

### AGP System Initialization and Termination

| | |
|---|---|
| JBEGN | Begin AGP |
| JEND | End AGP |

JBEGN is used to initialize the graphics system. It should be used before any other AGP call is made. Likewise, after the last AGP call, JEND should be called to terminate the graphics system. JBEGN guarantees a standard state in which all attributes and viewing transformation components are at their default values.

```
CALL JNEWF              * segment 10 is removed from the display now
   :
   :
CALL JWOFF (1)          * disable work station 1 for output
CALL JWEND (1)          * terminate work station 1
CALL JEND               * end AGP
```

## Enabling and Disabling Devices

JEDEV (ID,CLASS,*machine-dependent parameters*) Enable a logical device
JDDEV (ID,CLASS) Disable a logical device

JEDEV and JDDEV enable and disable devices at work station ID. The type of device is specified with the CLASS parameter. Possible devices are the alphanumeric display and all input devices: button, keyboard, locator, valuator, and pick. Values of the CLASS parameter are:

```
CLASS = 1 - Alphanumeric device
      = 2 - Button device
      = 3 - Keyboard device
      = 4 - Locator device
      = 5 - Valuator device
      = 6 - Pick device
```

As a rule, a device must be enabled before it is used. Also, the work station on which it is being enabled must have been previously initialized. For example,

```
CALL JDINT (2,machine-dependent      * initialize work station program 2,
            parameters,0)
      :                              *   with no control bits
      :
CALL JEDEV (2,4,machine-             * enable work station 2's locator
            dependent parameters)
CALL JWLOC (2,...)                   *   then use it
      :
```

Note that a device is enabled with JEDEV independently of graphics output being enabled through JWON.

An input device will not be enabled if the physical device is the same as a display device being outspooled. AGP will ignore a JEDEV that mentions a device previously assigned to spooled output and give an error.

Not all work station programs will be configured to have a logical device of each type; in fact, it is possible to have a work station with only one device. The devices to be included in a work station program are determined at load time. If JEDEV attempts to enable a logical device that the work station program does not have, an error will be generated. To prevent this, the available devices may be inquired any time after a work station is initialized.

The following example demonstrates initialization and termination of a work station, as well as enabling an input device.

```
CALL JDINT (1, machine-dependent          * initialize work station 1 and
            parameters, 0)
CALL JWON (1)                             * turn it on for graphics output
       :
       :
CALL JEDEV (1,4, machine-dependent        * enable the locator,
            parameters)
CALL JWLOC (1,0,BUTTON,VX,VY)             *   use it,
CALL JDDEV (1,1)                          *   and end it
       :
CALL HOUSE                                * send graphics output
       :
       :
CALL JWOFF (1)                            * turn off graphics output
CALL JWEND (1)                            *  and terminate the work station
```

# INQUIRY

A variety of inquiry functions are available. The status of the picture generation process, the status of AGP, and the physical properties of a work station are general classes of information available through the several inquiry functions.

## Work Station Inquiry

JIWS (ID,OPCODE,*machine-dependent parameters*, ISIZE,RSIZE, *machine-dependent parameters*, ILIST,RLIST) — Inquire work station traits

Work station inquiry is very useful. What is the aspect ratio of the display device? How many linestyles does it have? Questions like these may be answered by JIWS. As in previous calls, ID selects the work station to be

queried. OPCODE specifies what particular information is desired. Answers are returned in arrays: ILIST if they are integers or RLIST if they are reals and another machine-dependent array SLIST, if appropriate. These arrays must be sized appropriately and their sizes passed in ISIZE, RSIZE and a machine-dependent, SSIZE.

There is a correspondence between the values of OPCODE and the type of information returned. The rule is as follows: in an opcode, the ten-thousands digit has a machine-dependent meaning, the thousands digit indicates the number of integers, the hundreds digit indicates the number of reals to be returned, and the remaining digits (the tens and ones) are simply used to make each opcode unique. As an example, two OPCODEs are:

```
 254   Aspect ratio of the graphics display device
       RLIST(1) = Aspect ratio of virtual coordinate system
            (2) = Aspect ratio of logical display limits


1050   Does the graphics display device
       support hardware clipping at physical
       limits?
       ILIST(1) = 0 - No

       ILIST(1) = 1 - Yes, at the view surface boundaries.
                  2 - Yes, but only to the physical limits
                      of the device.
```

As may be seen, opcode 254, with a 2 in the hundreds digit, returns two reals, while 1050, with a 1 in the thousands digit, returns one integer. The remaining JIWS opcodes are documented under the JIWS routine in the *AGP System Supplement*.

One use of JIWS is device optimization: the use of inquiry to enhance the application's use of devices available to it. For example, it may inquire whether a pick input device is currently available. If so, the application will continue with the preferred pick algorithm. If not, it may use a keyboard-entry algorithm instead. (In this way, the interactive capabilities of the environment are fully utilized while preserving device independence.)

The following example shows how JIWS can be used to set the virtual coordinate system's aspect ratio to a work station's logical display surface, making the entire display available in a device-independent manner.

```
INTEGER ILIST(1)                          * 1 integer of information
REAL RLIST(2)                             * 2 real numbers of information
      :
CALL JDINT (1,...)                        * initialize work station 1
CALL JIWS  (1,254,machine-dependent       * inquire its logical display
            parameter,0,
            2,machine-dependent           *   limits
            parameter,ILIST,RLIST)
CALL JASPK (1.0,RLIST(2))                 * set aspect ratio accordingly
      :
```

## Output Primitive Inquiry

```
JICP  (X,Y,Z)                          Returns current position
JITSZ (NCHARS,STRING,SHIGH,SWIDE)      Returns text height and width
```

Graphics output primitives begin at the current position, inquired with JICP. It returns a world coordinate triple, (X,Y,Z), corresponding to the current position in world coordinate space.

JITSZ determines the height and width of a high quality text string before it is output. NCHARS is the number of characters in STRING. The height and width returned in SHIGH and SWIDE are calculated in world coordinates using the current text attributes: size, gap, and font. Note that the character base vector, the character plane vector, slant, and justification do not affect the height or width of a high quality text string.

The following subroutine uses text size inquiry to draw a string in a user-specified rectangle. Its design is complicated by the fact that high quality text fonts may be of variable width. The subroutine accomodates these by setting character size in two steps: first the size is approximated as if the font were fixed width, then the string width this size produces is inquired and used to refine the character size. Once the character size is set, the string is output as high quality text.

```
SUBROUTINE FIT (NCHARS, STRING, XMIN, XMAX, YMIN, YMAX)
INTEGER NCHARS, STRING (80)                * draws a text string in
REAL XMIN, XMAX, YMIN, YMAX                *   the given dimensions

REAL WIDTH, HEIGHT, SHIGH, SWIDE

WIDTH  = (XMAX - XMIN) / NCHARS            * set character height
HEIGHT = YMAX - YMIN                       *   and width as if the
CALL JCSIZ (WIDTH, HEIGHT, 0.0)            *   font were monospaced
CALL JITSZ (NCHARS, STRING, SHIGH, SWIDE)  * use width of the string
WIDTH = WIDTH / (SWIDE / NCHARS)           *   this size produces to
CALL JCSIZ (WIDTH, HEIGHT, 0.0)            *   calculate final size
```

```
CALL J2MOV (XMIN, YMIN)                  * move absolute
CALL JJUST (0.0, 0.0)                    * set justification
CALL JTEXH (NCHARS, STRING)              * draw high quality text

RETURN
END
```

## Polygon Set Inquiry

```
JIPST (ID,PINDEX,DENSTY,ORIENT,EDGE,DIDDF)   Returns current style
JI1IN (OPCODE,VALUE)                         Returns current state of AGP
JIWS  (ID,OPCODE, <machine-dependent         Returns work station
       parameters>)                            characteristics
```

JIPST inquires the contents of an entry (index) in a specified work station's polygon style table. Each entry specifies the following descriptors:

- The density of the polygon interior fill lines (DENSTY)
- The angle of orientation of the polygon interior fill lines to the view surface's horizontal axis (ORIENT)
- Display or non-display of an edge (EDGE)
- Device-independence or dependence (DIDDF)

JI1IN inquires the current polygon style index, interior color index and interior linestyle index for polygon primitives.

JIWS inquires the following work station-dependent information pertaining to polygons:

- Number of polygon styles supported on the graphics display.
- Number of polygon vertices supported by the display device.
- Whether or not the graphics display supports immediate retroactive change of polygon style or color.
- Whether or not the graphics display supports hardware generation of polygons.

## Color Inquiry

```
JICOL (ID,COLOR,COLP1,COLP2,COLP3)           Returns color model parameters
JI1IN (OPCODE,VALUE)                         Returns current state of AGP
JIWS  (ID,OPCODE, <machine-dependent         Returns work station
       parameters>)                            characteristics
```

JICOL returns the color model parameter values (COLP1, COLP2, COLP3) assigned to the specified color index (COLOR) in the specified work station's color table. The interpretation of COLP1, COLP2 and COLP3 depends on the color model in effect (set by JCOLM).

Calls to JDCOL can change the color tables for some graphics output devices; other devices do not allow their color tables to be changed and will ignore both JDCOL and JICOL. See the *Device Handlers Manual* for information on specific devices.

JI1IN inquires the current color index for output primitives, and the current polygon interior color index.

JIWS inquires the following work station-dependent information pertaining to color:

- Number of distinct colors supported by the graphics display.
- Maximum number of distinct colors that can appear on the graphics display at one time.
- Whether or not the device supports immediate retroactive color change.
- Whether or not background color can be redefined.
- Whether or not device supports modification to color table.
- Which color model is in use.
- Size of color table.

## Viewing Transformation Inquiry

| | |
|---|---|
| JIWND (XARRAY,YARRAY,ZARRAY) | Inquire window in world coordinates |
| JIMAT (OPCODE,MATRIX) | Inquire internal matrices |

JIWND and JIMAT inquire features of the viewing transformation. JIWND returns the four corner points of the window in world coordinates as ordered triples in XARRAY, YARRAY, and ZARRAY. This is the window set previously by JWIND. However, recall that the parameters to JWIND were in *viewing* coordinates, not world coordinates. JIWND does the conversion from viewing to world coordinates. The following program shows how JIWND can be used to draw a rectangle that corresponds to the window.

```
REAL XARRAY(4),YARRAY(4),ZARRAY(4)          * 4 points to be returned
        :
        :
CALL JIWND (XARRAY,YARRAY,ZARRAY)           * get window's corners
                                            *   in world coordinates
CALL J3PLY (4,XARRAY,YARRAY,ZARRAY)         * draw 3 sides of window
CALL J3DRW (XARRAY(1),YARRAY(1),ZARRAY(1)) * now draw side 4
        :
```

There are three internally maintained matrices accessible with JIMAT. The first is the viewing transformation. Set by the parameters of the viewing functions, it is used to translate world coordinates to virtual. The second, known as the inverse viewing transformation, is simply the inverse of the first. It converts from virtual coordinates back to world. Finally, the third, known as the modelling transformation, is the user-defined modelling matrix set directly by JDMOD. The matrix to be returned is determined by OPCODE:

1 - viewing transformation
2 - inverse viewing transformation
3 - modelling transformation

Inquiry can be used to change a modelling transformation. First, inquire the transformation, then change one or two of its elements using JDMOD.

## Segment Inquiry

| | |
|---|---|
| JISGA (NAME,VALUE) | Returns segment attributes |
| JISGW (SEGNAM,MAXNAM,COUNT,IDARRY) | Returns names of work stations bound to a segment |

Segment inquiry is available through JISGA and JISGW. JISGA returns the attributes of a segment, while JISGW returns the names of work stations bound to a segment. The three segment attributes are highlighting, visibility and detectability. Recall that they can be changed (through JSHI, JSVIS, and JSDET respectively) making inquiry a very useful tool to control them.

JISGA can be used to find the current attribute values of the segment identified by NAME. The attributes are returned in the VALUE array according to the following table:

```
VALUE(1) = -1 - the segment doesn't exist
            0 - the segment is not highlighted
            1 - the segment is highlighted

VALUE(2) = -1 - the segment doesn't exist
            0 - the segment is not visible
            1 - the segment is visible

VALUE(3) = -1 - the segment doesn't exist
            0 - the segment is not detectable
            1 - the segment is detectable
```

This call can be used to determine whether or not a segment exists: all three VALUEs will be -1 if it does not.

The following example shows JISGA being used when toggling the state of the visibility attribute.

```
INTEGER NAME, VALUE(3)           * segment name and attribute array
      :
      :
NAME = 9                         * segment 9 is of interest
CALL JISGA (NAME,VALUE)          * find its visibility in VALUE(2):
IF (VALUE(2) .EQ. -1) GO TO ...  *   doesn't exist => can't change
IF (VALUE(2) .EQ. 0) OPCODE = 1  *   is not visible => make visible
IF (VALUE(2) .EQ. 1) OPCODE = 0  *   is visible => make not visible
CALL JSVIS (NAME,OPCODE)         * reset the visibility attribute
      :
```

JISGW finds what work stations are bound to a segment. To be bound to a work station means to exist in its segment display area. The number of work stations containing the segment in question is returned in COUNT, and their names are placed in the integer array IDARRY (MAXNAM is the size of IDARRY). If the visibility attribute of a segment is set to invisible on the display device, it can still be returned with a call to JISGW.

## Real and Integer Inquiry

| | |
|---|---|
| JI1IN (OPCODE,VALUE) | Returns integer information |
| JI1RE (OPCODE,VALUE) | Returns 1 real value |
| JI2RE (OPCODE,VAL1,VAL2) | Returns 2 real values |
| JI3RE (OPCODE,VAL1,VAL2,VAL3) | Returns 3 real values |
| JI4RE (OPCODE,VAL1,VAL2,VAL3,VAL4) | Returns 4 real values |

The remaining inquiry functions can best be described as "miscellaneous". They vary by type (integer or real) and quantity (1,2,3 or 4) of the values returned in VALUE. OPCODE is always used to select which particular question is being asked. Its values and the information inquirable are documented under these calls in the *AGP Reference Manual*.

# TIMING

This section is concerned with understanding and controlling timing. The operator interacts with a user program, which transmits device-dependent commands to the work station program, where they are translated into device-dependent commands and sent to the graphics device. To produce computer graphics, each of these components must work together. A primary feature of this interaction is timing.

Note that AGP's timing choices are unrelated to operating system buffering. Instead, they are concerned only with the buffering done within the user program and between the work station program and the user program. AGP has no control over buffering in the operating system.

To simplify this, the following discussion assumes only one work station is initialized and enabled. To comprehend the different timing modes possible, assume that the simplest one, called immediate visibility, is in use. In *immediate visibility*, each output primitive is transmitted to the device prior to returning from the output call. This timing mode is least efficient in use of the system's resources. Figure 7-1 illustrates immediate visibility.

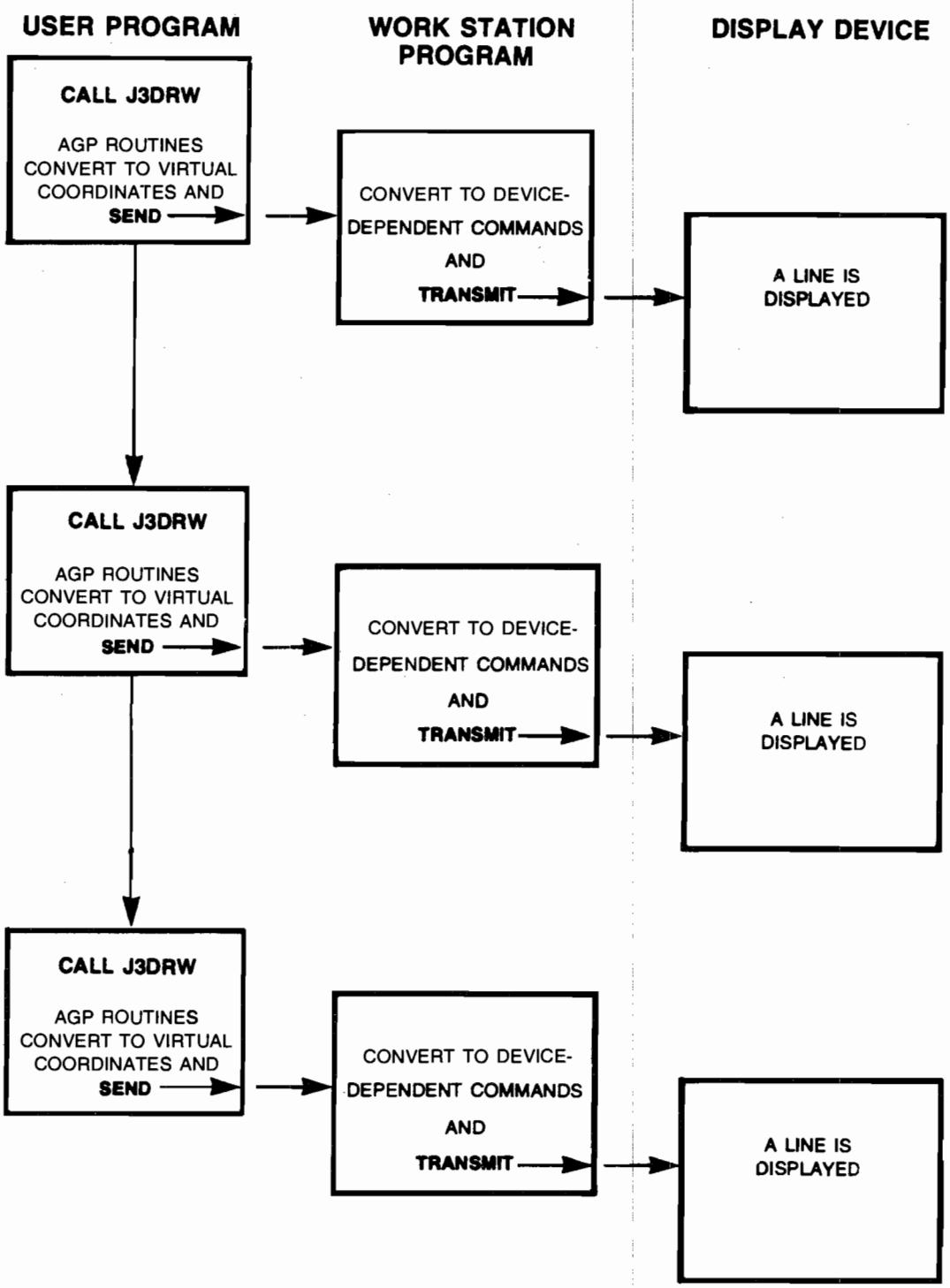


**Figure 7-1. Immediate Visibility.**

In *system buffering* mode, a number of output primitives are saved together in the user program before transmission to the work station program. In turn, the work station program buffers commands to the display device, as shown in Figure 7-2. An output primitive is not necessarily transmitted to the device prior to returning from the output call. With system buffering, the use of system resources for communication between the user and work station programs, as well as between the work station program and the display device, is optimized.

Another timing idea is "batching". Batching can be roughly defined as saving everything up for an interval of time, and then sending it all at once. In AGP, this is called *batch-of-updates*. To update the display device is to add an output primitive, remove a segment or change a segment's attribute. A batch-of-updates is the accumulation of display device updates for execution as a group.

Batch-of-updates is the opposite of immediate visibility. When the program requests batch-of-updates, not only are updates not sent to the display device individually, but they are not sent at all until the batch-of-updates terminate. Instead, they are saved in the segment display area. Notice that even primitives outside of segments are temporarily saved in the segment display area during a batch-of-updates.

Batching and buffering only affect *when* an image reaches a display device. They have no affect whatsoever on what the final picture looks like. With this in mind, let us examine these functions further.

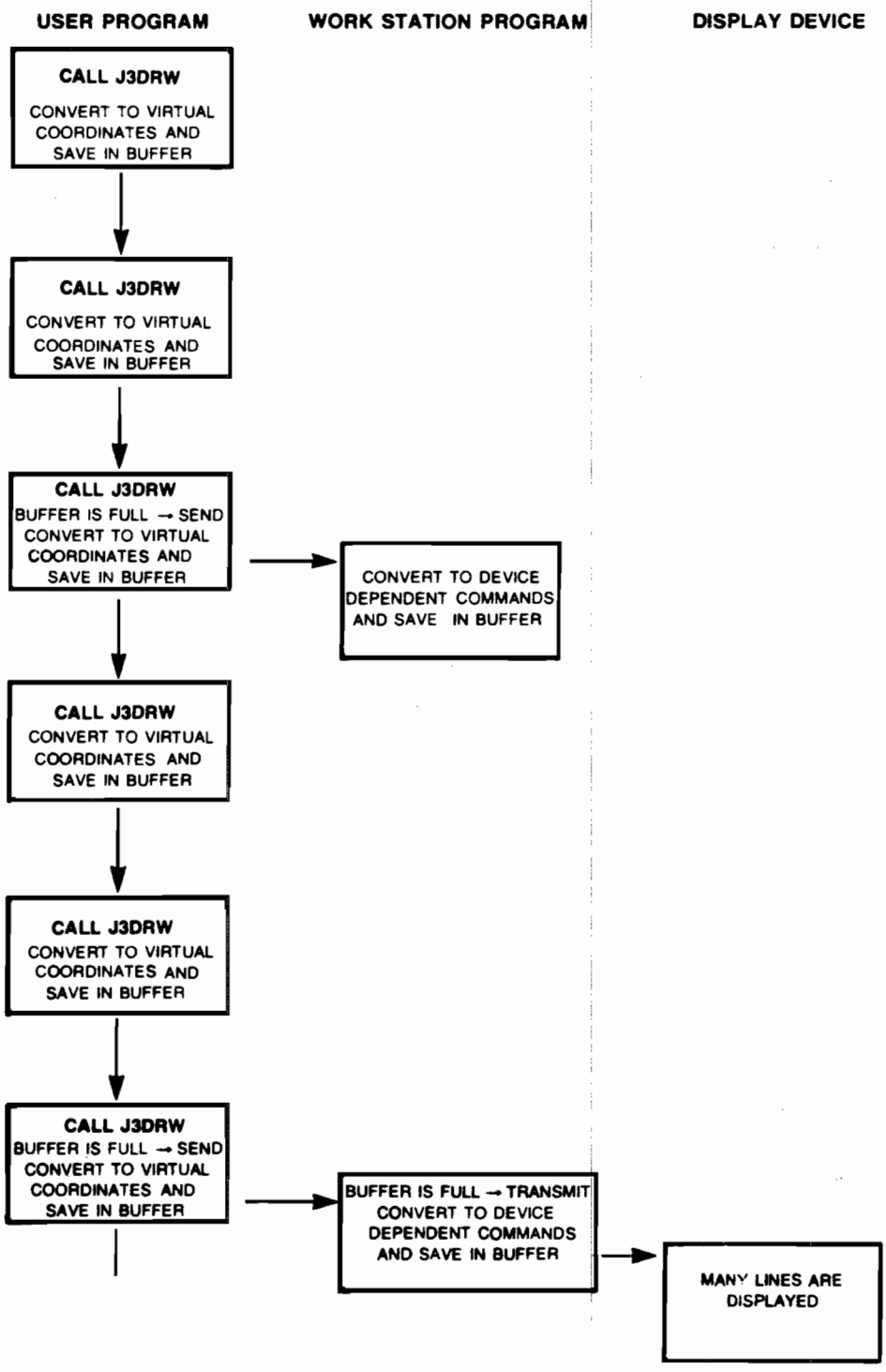


**Figure 7-2. System Buffering.**

## Immediate Visibility

| | |
|---|---|
| JIVON | Set timing mode to immediate visibility |
| JIVOF | End immediate visibility timing mode |

When JIVON has been used to request immediate visibility, display device updates are sent to enabled work stations before the subroutine is exited. Immediate visibility is cancelled with JIVOF, which causes return to the default mode (system buffering).

Immediate visibility is the least efficient of the batching modes. It should be used only when display device updates must take place as soon as they are defined, even if the finished picture will take longer to create. JMCUR is an alternative to immediate visibility that may be used to ensure that the display device is fully updated when in system buffering mode.

## System Buffering

| | |
|---|---|
| JMCUR | Make all enabled display devices current |

System buffering is the default batching mode of AGP. When it is in effect, display device updates requested by an AGP function are not necessarily sent to work stations before return from the function. When in system buffering mode, the display device may be made current at any time with JMCUR, which forces all currently buffered updates to be sent immediately. Note that JMCUR does not change the timing mode from system buffering.

Many AGP functions make the display device current automatically. Each of the input routines, namely JBUTN, JKYBD, JWLOC, JSLOC, JWVAL, JSVAL, and JPICK, make it current to ensure the operator has received all relevant updates before input is requested. Additionally, enabling a work station (JWON), disabling a work station (JWOFF), changing timing modes (JIVON,JBATC, and JUPDT), and error inquiry and setting (JIERR and JSERR) each implicitly updates the display device. There is no need for a program to call JMCUR before any of these functions.

## Batch-of-Updates

| | |
|---|---|
| JBATC | Begin batch-of-updates |
| JUPDT | End batch-of-updates |

JBATC sets the batching mode to batch-of-updates and JUPDT ends the batch-of-updates. Display device changes requested inside a batch-of-updates - between calls to JBATC and JUPDT - will not appear on the display device until JUPDT is called. Instead, they are saved in the segment display area of all enabled work stations. Not until the batch-of-updates is ended do the changes appear on the display devices of enabled work stations. No alphanumeric output calls (JALPH), input calls (JBUTN, JWVAL, JPICK, etc.) or initialization control calls (JDINT, JWON, JWOFF, JEDEV, JDDEV, JWEND) should be made within a batch-of-updates. An attempt to do so will result in an error and the batch-of-updates being closed through an implicit call to JUPDT.

When a segment is purged from a work station, AGP deletes the segment from its segment display area, then implicitly causes a new-frame-action to occur. Typically a new-frame-action erases the display device and then resends the contents of the segment display area. This is an efficient technique when only one segment is being purged. When two or more segments are purged sequentially, however, multiple new-frame-actions are generated, typically causing the display device to be re-drawn many times with only one segment removed at a time. Batch-of-updates will prevent this, allowing only one new-frame-action after all deletions have been made from the segment display area.

The following example illustrates this idea:

```
CALL JBATC          * hold off subsequent display device changes
CALL JPURG (1)      * purge segment 1
CALL JPURG (2)      *   and segment 2 ...
      :
      :
CALL JPURG (n)      *   through segment n
CALL JUPDT          * update the display device now
```

Without the use of batch-of-updates, the purge sequence would implicitly cause 'n' new-frame-actions, which would typically re-draw the screen 'n' times. If only the final result is of interest, in the intermediate steps the screen is re-drawn needlessly. Using JBATC and JUPDT guarantees that at most one new-frame-action is generated. Note that the final picture produced will be the same whether or not batch-of-updates is used - only the intermediate steps differ.

Another example shows a second use of batch-of-updates. The following program section creates a series of rotated views of an object. The subroutine SVIEW modifies the viewing transformation in effect, while HOUSE plots the object as a series of primitives outside of segments. Batch-of-updates lets the next view of the object be computed and stored temporarily in the segment display area before deleting the previous view from the display device.

```
      DO 5 DEG = 1,360,10     * rotate 360 degrees in 10 degree increments
         CALL JBATC           * begin batch-of-updates (so there will
                              *   be no display changes until JUPDT)
         CALL JNEWF           * clear old primitives left from last loop
         CALL SVIEW (DEG)     * set up rotation
         CALL HOUSE           * plot object as primitives outside segments
         CALL JUPDT           * end batch, causing old picture to be removed
    5    CONTINUE             *   (due to JNEWF) and new one to replace it
```

The old view of the object is left on the display device until computation of the new one is complete. In this way, computation of the next view can continue while the current view is being viewed.

## ESCAPE FUNCTIONS

| | |
|---|---|
| JOESC (ID,OPCODE,ISIZE,RSIZE,ILIST,RLIST) | Output escape access |
| JIESC (ID,OPCODE,ISIZE,RSIZE,ILIST,RLIST) | Inquiry escape access |

While AGP provides a device-independent method of addressing graphics devices, it does recognize that some devices may have capabilities not directly supported. For example, suppose a device is capable of "area fill", meaning a rectangular area may be filled with a user-definable pattern. This feature is not accessible through the usual output primitives of AGP, but is still a useful one in many applications. AGP has a convenient, standard mechanism for addressing such device-dependent features.

This standard mechanism is called an escape function. Escape functions are of two types: output escape functions (sent by JOESC) provide access to the graphics display device, and inquiry escape functions (sent by JIESC) return information about the device. The particular feature to be accessed is chosen through a unique integer, OPCODE. Since the different functions OPCODE may select are device-dependent, its possible values are documented in the *Device Handlers Manual*. If the OPCODE used is not supported on a particular device, an error will be produced and the call will be ignored.

Escape functions are device-dependent. Remembering that one of the goals of an AGP application program is to remain device-independent, escape functions can threaten that objective. If the particular hardware feature

being used is not available on other devices, a software substitute should be provided, if possible. Also, escape functions should be localized to make subsequent modification as easy as possible.

Escape functions may be used anywhere in an application program, with certain restrictions. JOESC and JIESC should not be called during a batch-of-updates or while a segment is open. Inquiry escape functions should not be directed to a work station that is being outspooled. If any of these conditions are violated, an error will be generated and the call will be ignored.

The ISIZE, RSIZE, ILIST and RLIST parameters concern integer and real arrays whose significance depends on the opcode chosen. ISIZE is the size of the integer array ILIST, while RSIZE is the size of real array RLIST. If ISIZE or RSIZE do not equal the required number of parameters, an error will be produced and the call will be ignored.

AGP strives at all times to know exactly what is occurring on each of the work stations in use. The segment display area allows AGP to maintain its own version of the display device. Escape functions allow the application programmer to circumvent the AGP subsystem and interact with devices directly. Although AGP maintains the escape function avenue, it does not keep track of the use made of it. Output escape functions are not saved in the segment display area. Also, viewing transformations are not applied to them. In fact, escape functions may threaten the integrity of the work station interface, causing AGP to become uninformed of the status of the work station. The result, from the application program's point of view, is that escape functions should be used carefully.

## ERROR HANDLING

| | |
|---|---|
| JSERR (*machine-dependent parameters*, RLEVEL, ALEVEL) | Set error report conditions |
| JIERR (ERRNUM,LEVNUM,SUBNUM, INFO) | Inquire last error |

JSERR and JIERR enable the application program to control AGP's error handling process.

Several types of errors may be generated by an application program. Examples include programming errors, such as attempting to delete a segment that does not exist, and environment errors, such as requesting input from a work station that does not support input. Whenever an error occurs, AGP creates and logs an error message. A complete listing of possible errors is contained in the *AGP Reference Manual*.

All errors are assigned a severity level from 0 to 4, where level 0 denotes a warning and level 4 denotes a fatal error. When the error is non-fatal,

AGP will attempt recovery before returning to the calling program. The recovery steps taken for specific errors are documented in the *AGP Reference Manual*.

JSERR specifies the level of error that is to be reported and the level of error that causes the graphics system to stop processing. An error must be of severity greater than or equal to the error level (RLEVEL) or it will not be reported. Initially, the error report level is 0, thus, all errors will be logged to a machine-dependent location. If the error report level is set to 4, the highest severity level, only level 4 errors are reported.

ALEVEL specifies the error abort level. Errors of a severity level greater than or equal to ALEVEL will cause the graphics system to stop processing. When ALEVEL is 0, all errors and warnings are fatal; when ALEVEL is 4, the recoverable errors (levels 0-3) will not cause discontinuation of processing. At initialization, ALEVEL is 4. Once the graphics system has discontinued processing due to an error equal to or exceeding ALEVEL, all AGP calls will be ignored except for JIERR, JBEGN, and JEND.

JIERR returns the most recent error detected by AGP. ERRNUM is the error number and LEVNUM denotes the severity level of the last error reported. The number of the routine that detected the error is given by SUBNUM. Although INFO generally provides additional information about the error, for some errors INFO will be undefined and no additional information will be provided.

As a general programming practice, it is *not* a good idea to call JIERR after every AGP function. The mechanism by which this information is returned forces an implicit flush of the buffers used in system buffering. This may have a negative effect on system and application performance.

Once AGP has discontinued processing, it retains the error information that caused it to do so. JIERR can be called to determine that error. This call to JIERR must be made correctly because AGP will not attempt to process it if it is made incorrectly.

# Index

## A



## B

## C

## D



## E

## F



## G



## H



## I

## J

## L



## M

## N



## O

## P

## R



## S

## T

## U



## V

## W



## Y

**READER COMMENT SHEET**

**Advanced Graphics Package**

**User Guide**

**97085-90000 June 1983**

We welcome your evaluation of this manual. Your comments and suggestions help us to improve our publications. Please use additional pages if necessary.

| | | |
|---|---|---|
| **Is this manual technically accurate?** | **Yes [ ] No [ ]** | (If no, explain under Comments, below.) |
| **Are the concepts and wording easy to understand?** | **Yes [ ] No [ ]** | (If no, explain under Comments, below.) |
| **Is the format of this manual convenient in size, arrangement and readability?** | **Yes [ ] No [ ]** | (If no, explain or suggest improvements under Comments, below.) |

**Comments:**

---

**Date:** _____

**FROM:**

**Name** _____

**Company** _____

**Address** _____

_____

_____

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1070 CUPERTINO, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

Publications Manager
Hewlett-Packard Company
Engineering Productivity Division
11000 Wolfe Road
Cupertino, California 95014

FOLD FOLD


MANUAL PART NO. 97085-90000
Printed in U.S.A.
E0683

HEWLETT-PACKARD COMPANY
Engineering Productivity Division
11000 Wolfe Road
Cupertino, California 95014