THE PICTURE SYSTEM
USER'S MANUAL

Evans & Sutherland Computer Corporation
#3 Research Road
Salt Lake City, Utah 84112

Preliminary Edition	April 1974
Revised	June 1974
First Edition	December 1974

All reference to this document should be made to:
No. ES-PS-SOC1-003.

TABLE OF CONTENTS

REFERENCES

1. INTRODUCTION

THE PICTURE SYSTEM is a stand-alone, general purpose, interactive computer graphics system which can display smoothly moving pictures of two- or three-dimensional objects. This system has all the capabilities which have been found to be needed and wanted by users of computer graphics systems. It has been designed as a problem-solving tool, a hardware/software system which satisfies real needs and can be used to solve practical problems.

Evans & Sutherland line drawing systems traditionally have been applied to applications where perspective and dynamic motion, like rotation and zooming, are required. THE PICTURE SYSTEM has the same digital hardware capabilities as the previous systems, but in addition, has digital picture buffering for refreshing the display. The built-in Refresh Buffer memory allows more lines and characters in a picture and eases the time and data storage burden on the computer which controls THE PICTURE SYSTEM.

All the dynamic capabilities for picture processing are standard in THE PICTURE SYSTEM. The basic components of the system are a DEC PDP-11; hardware procesing units for performing such functions as rotation, zooming and perspective; an 8192-point Refresh Buffer; a Picture Generator; a Character Generator; a 21" Picture Display; a Tablet to facilitate picture interaction; and the software to support the system.

2. OVERVIEW OF INTERACTIVE COMPUTER GRAPHICS

Computer graphics is a relatively new and important branch of computer technology in which computers prepare and present pictorial output. Interactive computer graphics goes one step further in that it allows a user to dictate changes to the picture and see the results immediately. If a system's time lag is more than a few seconds, it does not qualify as interactive; in some systems, however, the time lag is a very small fraction of a second, in which case the user gets the feeling that he is actually manipulating the picture itself.

Computer graphics is a very broad subject and even an overview of it can diverge into a great many topics. The purpose of this chapter is to present in general terms the concepts necessary for understanding and using THE PICTURE SYSTEM. Consequently, it devotes little discussion to some aspects of graphics which may be of interest and importance to some readers but which are not prerequisites to understanding the rest of this manual[1].

A study of graphics can be broken down into four major topic areas: presenting a prepared picture, representing structures to be depicted, preparing a picture of such structures and interacting with the picture. Each of these areas is explored in the following sections.

---

[1] Principles of Interactive Computer Graphics, Newman and Sproull, McGraw-Hill, 1973 is a recommended reference covering most aspects of computer graphics.

2.1 PICTURE PRESENTATION

Computer users are familiar with output media such as listings and magnetic tape, where computed results are recorded in numerical form. Often the numerical form is an artifial way of presenting pictorial data. Computer graphics offers a new output medium on which data can be presented visually.

2.1.1 Graphical Output Media

At one end of the graphics spectrum lie plotters, where a computer-driven pen creates a picture on a stroke-by-stroke basis. Plotters are unmatched for resolution (a measure of the density of individually distinguishable output values), but are extremely slow compared to other graphic output devices.

Next there are raster printers, where the computer selectively fills elements of a rectangular mesh with ink. The pattern of filled and empty elements can be assembled by the eye into a picture when viewed from a reasonable distance. Raster printers have rather coarse resolution but are much faster than plotters.

Output on paper is permanent, which can be an advantage or disadvantage. To meet the need for an impermanent graphic output medium, the cathode ray rube (CRT) is used. Information is presented on a CRT by directing a beam of electrons about on its phosphor coated face. One form of CRT, called the storage tube, retains pictures semi-permanently by "capturing" the electrons in tiny cells on its face so that those cells glow until the electrons are "freed" by an erase pulse. The other form is the refresh CRT, whose face emits light for an instant when it is struck by the electron beam and then turns picture to retain the image which is referred to as refreshing.

Like paper, the refresh CRT can be filled with a matrix of dots or can be drawn upon with a set of strokes at any position and any angle. An example of the former is the home television; an example of the latter is THE PICTURE SYSTEM's Picture Display.

2.1.2 Refresh Rate

Since the phosphor on the refresh CRT fades almost immediately after it is struck by the electron beam, the picture must be continually redrawn to be viewed. This rate at which it is redrawn is called the refresh rate usually measured in frames per second. If the picture is not redrawn frequently enough, the eye will notice it fading between refreshes, producing an unsightly effect known as flicker. The flicker threshold varies somewhat from phosphor to phosphor and from observer to observer, but most observers of the common phosphor, P4, begin to see flicker at a refresh rate of about 30 times per second. That is, pictures redrawn more than 30 times per second appear flicker free; pictures drawn less than 30 times per second do flicker; and pictures drawn exactly 30 times per second are marginal.

2.1.3 Line Generation

A line is specified by two end-points (x,y) and (x'y'), expressed in the coordinate system of the CRT, called screen coordinates. The actual movement of the electron beam between the two points is accomplished by a hardware device called a line generator or a vector generator. A sophosticated line generator is also capable of drawing lines with a program-specified intensity, or even varying the intensity of a line from one end to the other. In this most general case, where line endpoints are specified by the three coordinates (x,y,z), the intensity or brightness of lines can appear to trail off in the distance producing an illusion of depth. This technique is known as depth-cueing.

Line generators can often be made to draw lines in any of a choice of modes such as solid, dashed, blinking, dashed and blinking, etc. Line generators which can service more than cne CRT are equipped with a facility for scope selection. A display program may select one or more scopes and then any subsequent lines drawn appear on all the selected scopes.

2.1.4 Update Rate

The advantage of the refresh CRT is that it can show smoothly changing pictures. Lines drawn on a CRT do not really move, of course, but the illusion of motion is imparted by continually redrawing the picture with lines at slightly different positions each time, or each frame. The eye blends this sequence of slightly different frames together into a smoothly moving picture such as a motion picture. The rate at which these different frames can be displayed is called the update rate. In contrast to the refresh rate which counts the number of pictures drawn per second, whether or not they are changed, the update rate counts only those frames that are different.

2.1.5 Picture Buffering

In THE PICTURE SYSTEM a refresh buffer provides storage so that the refresh and update rates may be different. Although refresh of 30-40 frames per second is required to avoid flicker, update of 10-20 frames per second is adequate to provide smooth motion. In effect, each new frame is shown two, three, or even four times while the next frame is being computed.

Data resident in a refresh buffer is called a display file. Full frames stored in this buffer may be read out and used to refresh the CRT any number of times before a new frame is created. Typically, new frames are created 20 times a second and the picture is refreshed 40 times a second; i.e., each frame is shown twice. Thus, the presence of a refresh buffer allows both refresh and update to proceed at their respective optimal rates and the system has a larger line capacity than it otherwise would.

A potential problem area exists when a picture is refreshed from a memory which is simultaneously being filled with a new frame; namely, that a picture displayed may consist of some lines from one frame and some from another. This can produce a number of effects, some very unsightly. To avoid this problem, the refresh buffer can be split into two separate buffers and update and refresh can be switched between the two in a way which avoids conflicts. This is called double-buffering, and its only disadvantage is that the amount of pictorial data which may be buffered is halved. In some cases this can place an unnecessarily low ceiling on the line capacity. The alternative, single-buffering, can be used to take advantage of the entire buffering space when the effects are not too disturbing, usually when the pictures shown are not highly dynamic. In systems without a refresh buffer the update and refresh rates must be the same. This limits the amount of data that can be displayed and the complexity of the picture that can be processed.

2.2 PICTURE DEFINITION

Data ultimately deposited in a refresh buffer must originate in the memory of the computer controlling the system. This computer-resident data is called a data base and may be vastly different in form from the display file which emanates from it.

Data bases may be highly structured, requiring a complex program to weave through them, or they may be very straightforward. The data base contains the coordinates of points in the structure to be displayed, along with instructions for interpreting those points. Along with coordinate information there may be pointers, substructure names, and other non-graphic information and attributes.

Points are the basic geometric entities in the data base. There are three basic instructions for treating a point: move the beam to that point, draw a line to that point, or draw a dot at that point. Graphics systems are often designed to understand codes for several of the most common sequences of the basic instructions (such as; "move,draw,move,draw,..."), so that large tables of points can be processed based on a single pre-specified code.

The most straightforward way to specify the position of a point is simply to state its absolute coordinates. An alternative that often introduces considerable efficiencies, called relative coordinates, entails stating the displacement required to get to a point from the previous point. Codes for common sequences like "absolute, relative, absolute, relative..." can be made recognizable to facilitate handling tables of points.

If a structure to be displayed lies in a plane, it is simplest and most efficient to define it using two-dimensional data. In this case it is typical to supply an "x" and a "y" coordinate for each point in the structure, and then perhaps a single "z" coordinate which applies to all the points.

If however, the structure is non-planar, it must be defined as three-dimensional data where a coordinate triple of the form (x,y,z) is given for each point.

In general a full computer word is devoted to each coordinate of each point and all coordinates are expressed as integers. In a 16-bit computer, then, the largest expressable positive number is 32767. This is sufficient for many applications, but the need to express larger numbers sometimes arises. This need can be met, at the expense of some loss of resolution in data definition, by employing an alternate means of expressing data called homogeneous coordinates. Here a point (x,y,z) is defined by the four coordinates (hx,hy,hz,h•32767), where "h" is an arbitrary number between zero and one.

If each of the numbers x, y, and z is less than or equal to 32767 in magnitude, "h" would be made equal to one (in order to preserve maximum precision) and the expression becomes (x,y,z,32767). If one of the Cartesian coordinates, say x, is 50000, the value of homogeneous coordinates becomes apparent because "h" can be made 1/2 to make x expressable; the point is then defined as (12•50000,1/2•y,1/2•z,1/2•32767) or (25000,1/2•y,1/2•z,16384), all perfectly expressable numbers. It is apparent though that resolution is lost; when "h" is 1/2, it is impossible to exactly express odd values for the original coordinates. In the example above, the expression of an x of 50000 is identical to the expression of an x of 50001. Furthermore, resolution is lost in all three coordinates even if only one of them is out of bounds. Smaller values of "h" impose a correspondingly greater loss of resolution.

It is customary to conserve core by supplying only the first three coordinates (hx,hy,hz) for three-dimensional points, or just two coordinates (hx,hy) for two-dimensional points (with a common value for hz), and to pre-specify a fourth coordinate (usually referred to as "w") which applies to several such points.

The user may be tempted to assume that relative coordinates are another method of extending the bounds of the data space beyond the normal limit of 32767 (e.g. setpoint to (30000,30000), draw relative to (20000,20000), leaving the beam positioned at (50000,50000)). Such is <u>not the case</u> and an attempt to accumulate relative positions beyond the maximum representable values will cause wrap-around, i.e. a number of opposite sign and erroneous magnitude will result.

2.3 PICTURE PREPARATION

The data base is almost never identical to the display file because the data base represents some scene, or collection of structures while the display file represents some view of that scene. To create a display file, transformation of the data base is required. In order to prepare a structure for display, it may have to be changed in size, position, or orientation; it may have to be put in perspective as seen from a given vantage point; parts of it may have to be removed to keep everything within a given field of view; and its coordinate system may have to be changed to conform with the output device. All of these steps can be expressed mathematically and implemented in software or hardware.

It is possible to implement the picture preparation steps in software using a general-purpose computer, but this is relatively slow. Hardware, while less flexible, is much faster. Fortunately, many of the steps involved in picture preparation are invariant from application to application which makes it very worthwhile to implement them with special purpose hardware. Any calculations unique to a given application can still be performed in software.

To meet the demand for fast frame creation, high-performance graphics systems employ special purpose hardware processors to implement the picture preparation steps. These steps are described in the next sections.

2.3.1 Simple Linear Transformations

Linear transformations (rotations, translations, scalings, etc.) can be described by parameters which indicate the type and degree of transformation. If the transformation parameters are properly arranged into a matrix, a vector of original coordinates can be multiplied by this matrix to yield a vector of new coordinates reflecting the desired transformation.

A 4x4 matrix can represent any rotation, translation or change in scale and can be used to transform points represented by homogeneous coordinates or as special cases, two- or three-dimensional coordinates.

This matrix expression of transformations is used because of its simplicity and because system design can then take advantage of the large body of knowledge about matrix arithmetic.

2.3.2 Compound Linear Transformations

All linear transformations can be expressed as a sequence of simple translations, rotations and changes in scale. A transformation expressable only by such a sequence is called a compound transformation. When a compound transformation is to be applied to a set of points, it would be possible, but extremely time-consuming, to apply the first simple transformation to the original coordinates, then apply the second transformation to the resulting coordinates, and so forth, for each point in the set. Enormous savings can be introduced, however, by taking advantage of the fact that matrix multiplication is associative: it is equivalent to first forming a composite matrix by multipling together matrices representing all the simple transformations in the sequence, in the same order in which the data would have encountered the original tranformations and then applying this composite matrix to all points to be transformed. The process is known as transformation concatenation.

2.3.3 Perspective

It is relatively straightforward to prepare two-dimensional data for display on a two-dimensional medium. Three-dimensional data may be converted to two dimensions after transformation by simply dropping the depth (or z) dimension. The resulting picture, however, would not look realistic because in real life the depth dimension has an enormous effect on the appearance of the horizontal and vertical dimensions. This effect, known as perspective, accounts for the convergence of parallel lines in the distance.

The perspective operation entails computing a point projection of three-dimensional points onto a plane representative of the screen, as depicted in Figure 2.3-1. Perspective can be applied to three-dimensional data by taking advantage of the fact that the perspective transformation is expressable in matrix form: a perspective transformation matrix can be included at the end of the sequence of rotation, translation, and scale matrices to transform three-dimensional data into a two-dimensional perspective representation.

2.3.4 Windowing

In some graphics applications, the data base is to be displayed in its entirety on the screen. Often, however, a closeup of some portion of the data base is desired and the rest is preferably omitted. Determining what to omit is not easy, and is particularly difficult if parts of the data base have been transformed. In fact, this determination is so time-consuming in software that it jeopardizes the dynamic movement of the picture.

Sophisticated graphics systems address this so-called windowing problem by performing a visibility check in hardware after the transformation stage and drawing only visible lines on the display. One implementation of windowing is called clipping and entails comparing all lines with the boundaries of a program-specified field of view superimposed on the data base. Lines or portions of lines outside the field of view are eliminated and only visible lines are passed on for display on the screen.

In two dimensions, the field of view is a rectangle called a window, superimposed on the plane of the data base. Clipping is easiest if the sides of the rectangle are parallel with the coordinate axes; however, this presents no restriction since the effect of a rotated window can be obtained by rotating the data in the opposite direction.

A window is specified by supplying values for its left, right, bottom and top boundaries using the same coordinate system used in the data base. Two-dimensional clipping is diagrammed in Figure 2.3-2.

In three dimensions the field of view is a three-dimensional region. It may be a rectangular volume, or, if its contents are to be seen in perspective, a section of a pyramid called a frustrum of vision. Such a frustrum is shown in Figure 2.3-3 along with the parameters necessary to completely specify it.

In Figure 2.3-3 an eye positioned at point E along the Z axis is to see the portion of the data base that lies within the frustrum whose hither (near) boundary is at point H, yon (far) boundary is at point Y, and whose side boundaries are determined, as in the two-dimensional case, by the window left, right, bottom and top boundaries at the hither plane.

As in the two-dimensional case, lines are retained, completely eliminated, or partially eliminated depending on whether they are completely within, completely outside, or partially outside the frustrum of vision.

Another approach to windowing is called scissoring. Scissoring entails making available a screen coordinate drawing space which is somewhat larger than the screen itself and then intensifying only the lines and line segments actually on the screen. Scissoring is easier to implement than clipping and does not take up time in the picture preparation stage. On the other hand, scissoring permits an effective drawing area only slightly larger than the screen as opposed to the vastly larger effective drawing area permitted by clipping. Another disadvantage of scissoring is that the line generator spends time tracing out all lines both visible and invisible, which makes flicker occur more readily.

2.3.5 Conversion to Screen Coordinates

Coordinate data that is not rejected by the clipping process is within limits determined by the field of view which may be of any size and at any position in the data base definition space. However, it is generally undesirable to display that data in a corresponding size and position on the screen. Rather, the data should be properly scaled (or mapped) so that it fills some program-specified region on the screen called a viewport. This can be accomplished by performing a final processing step which linearly maps all data from the window to the viewport.

Left Blank Intentionally.

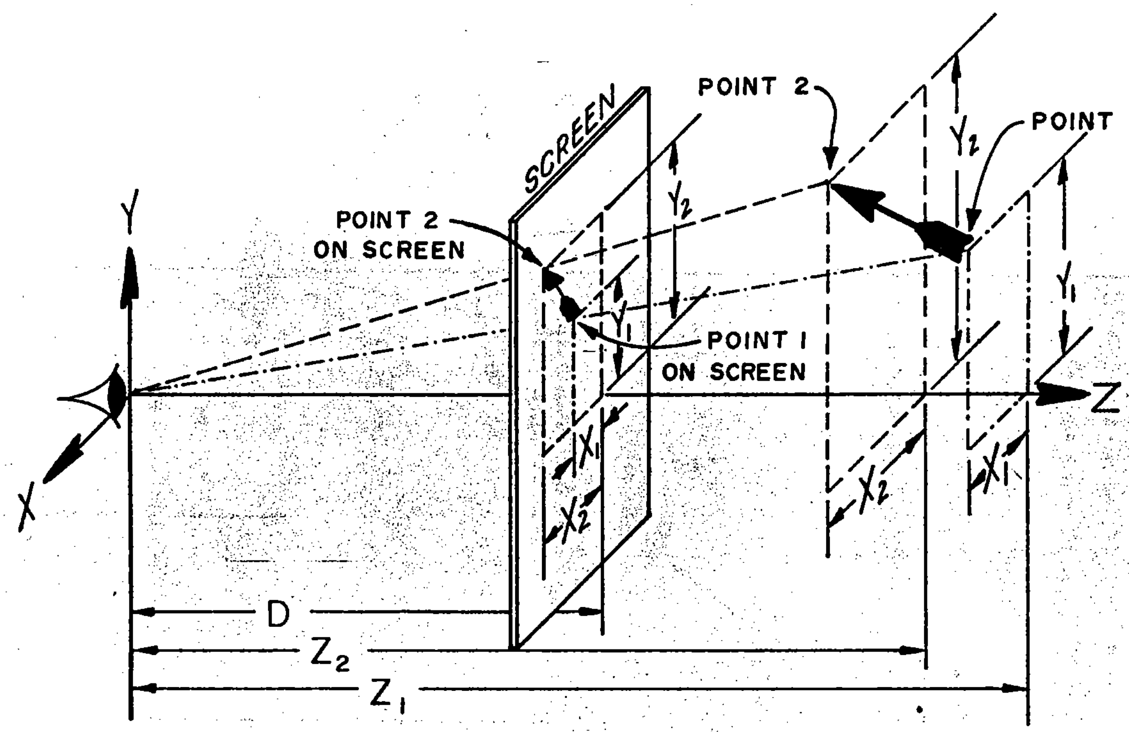


Figure 2.3-1

Three-Dimensional Perspective Projection onto a Two-Dimensional Plane

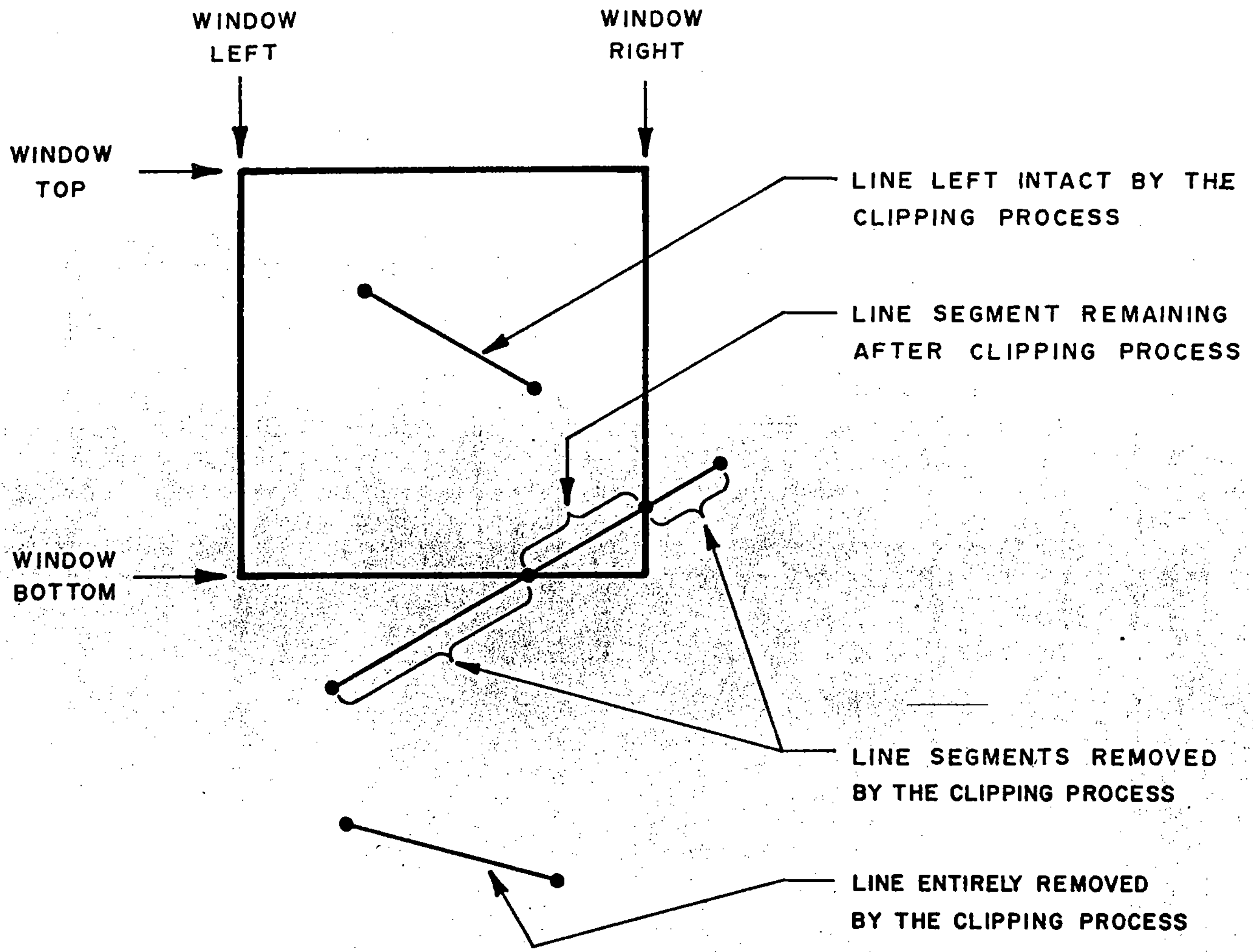


Figure 2.3-2
Two-Dimensional Clipping

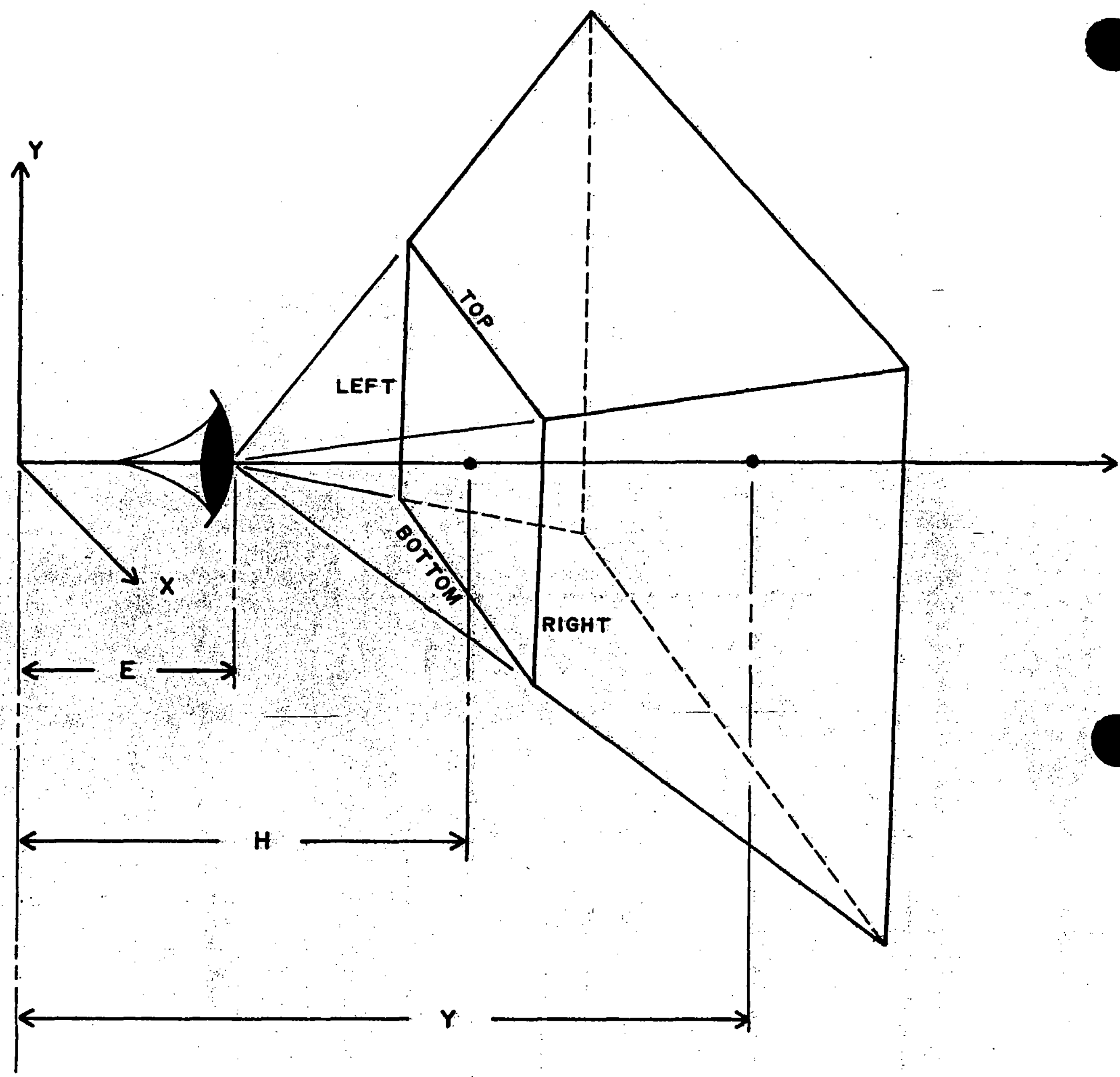


Figure 2.3-3

Frustrum of Vision showing the Eye Position in Relation to an Arbitrary Coordinate Axis

If the viewport is a rectangular region aligned with the screen axes, it can be specified by supplying the screen coordinates for its left, right, bottom and top edges. If the system's Line Generator can draw lines of varying intensity, a viewport may also specify the intensity limits for the data displayed. These limits specify the intensities of the data at the hither and yon boundaries and are called the hither and yon intensities. When the hither and yon intensities are different, the intensity of the displayed picture elements varies between these limits, allowing an illusion of depth to be imparted to the picture. Thus, a viewport is used to specify the region of screen and the intensity limits for the data to which, in the most general case, the frustrum of vision is mapped. Figures 2.3-4a and b show how data may be displayed within a viewport which is the entire screen or only a portion of it. Viewports may also be utilized to map data into the coordinates of devices other than a display. For example, viewport boundaries could be specified in the coordinate system of a plotter or similar device to provide the capability of obtaining hard copy output to the precision of the plotting device.

An advantage of program-specified viewports is that several may be assigned in the same program each receiving different data. This technique proves convenient for many purposes in graphics, such as showing different views of an object or views in different directions from the same point on the same output device simultaneously.

2.3.6 Text Display

Almost all graphics applications call for the presentation of alphanumerics on the screen at one time or another. It is of course possible to define character shapes in the data base like other picture elements and in fact this is necessary if characters are to be treated like other objects, i.e., rotated, clipped, etc. However, it is possible to derive efficiencies from the foreknowledge of character properties when they do not require such sophisticated treatment, by generating the actual strokes of the characters just prior to drawing them and dealing only with character codes up to that point.

A hardware device which accepts character codes and produces the strokes comprising the character is called a character generator. Character generators generally provide flexibilities in the size, shape and orientation of the characters they produce.

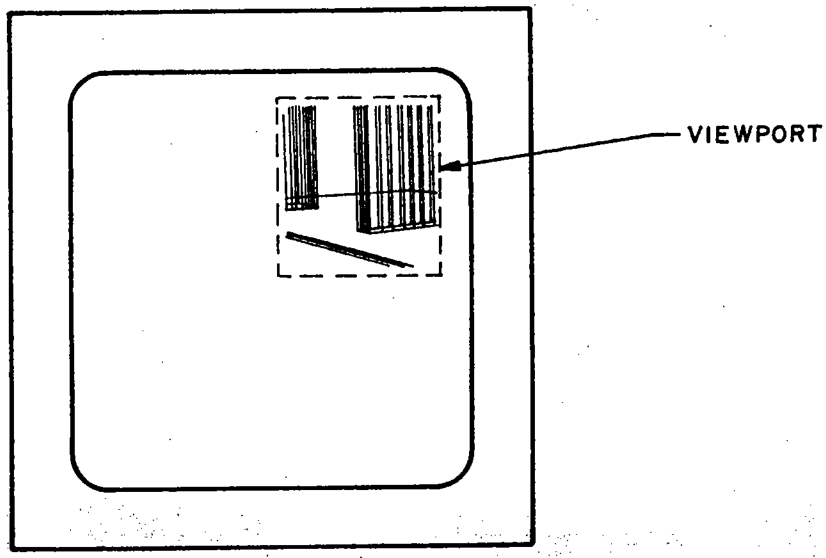


Figure 2.3-4a

Partial Screen Viewport

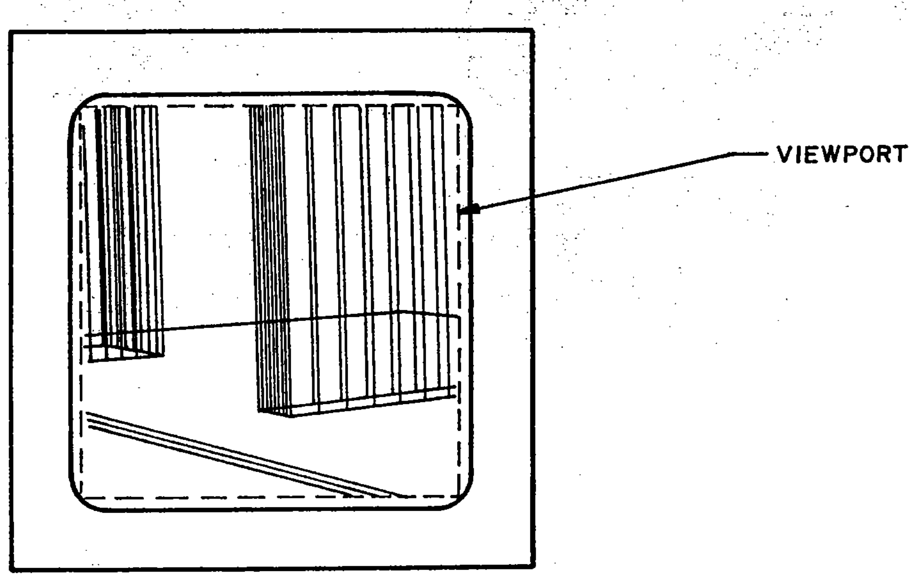


Figure 2.3-4b

Full Screen Viewport

To use such a device to draw a string of characters, a display program must first stipulate character size, shape and orientation values; then position to where the string is to begin and insert a set of packed character codes, called a text string, into the display file. The character generator would then interpret the text string, look up the set of strokes associated with each code, size and orient the strokes properly and draw the characters on the output device. Codes are packed into text strings as a memory conservation measure.

2.4 PICTURE INTERACTION

Sophisticated graphics applications often require that the form or content of the picture be changeable by the user. A number of input devices for this purpose are generally made available and each has its strong points.

A common input device is the light pen which is a light sensitive stylus connected to the computer. When the tip of the stylus is held against the screen and over a line segment, an interrupt is generated. The computer can then determine what line in the display file was being pointed at.

Function switches are frequently attached to the computer in a graphics system. These are toggle switches or push buttons whose polarity can be read. Each switch can be assigned a meaning unique to the program.

Several analog input devices are sometimes used for interaction, including control dials, joysticks and trackballs. These devices offer one or more degrees of freedom over which a user can enter input values used to control rotation, translation, scaling, etc.

A versatile interactive input device is the tablet, which is a flat rectangular plate which may be positioned on a table in front of, or near, the display screen. Associated with the tablet is a pen which may be moved about over the plate and whose position on the plate may be read with fine resolution by the computer controlling the system. The computer can also detect whether the pen is actually touching the plate and may also indicate if the pen is near the plate. To tie pen motion together with a picture, a cursor is generally drawn on the screen. This cursor is a small symbol which continually moves about in concert with the pen. It soon becomes natural to guide the cursor to a desired position on the screen by an appropriate motion of the pen.

A tablet is considered the best input device for entry of precise positional information. It can also be programmed to perform the functions of function switches or the analog devices. In order to enable a tablet to perform the pointing function of the light pen, the system should be equipped with a hit test feature which checks all data as it emerges from the transformation stage for proximity to the pen position. The user positions his cursor over the target structure and initiates the hit test feature (perhaps by touching the pen down). If a target structure is encountered a flag is set which may be later tested or may be programmed to cause an interrupt. This method of pointing has the advantage that the target structure is marked in the data base, not the display file. It is often difficult or impossible to backtrack from an entry in the display file to find its corresponding entry in the data base.

The tablet also has a human engineering advantage over a light pen. The user of the tablet is allowed to sit in a natural writing position and at any distance desired from the graphic display. This reduces user fatigue and improves operating conditions.

3. OVERVIEW OF THE PICTURE SYSTEM

This chapter provides an overview of the hardware components which comprise THE PICTURE SYSTEM. A functional diagram of the Standard Configuration of THE PICTURE SYSTEM is shown in Figure 3-1. The user of THE PICTURE SYSTEM will normally interface with these components by means of the Graphics Software Package described in Chapter 4 of this manual. The user should, however, gain a functional understanding of the hardware components to fully understand the use of the graphics software provided with THE PICTURE SYSTEM.

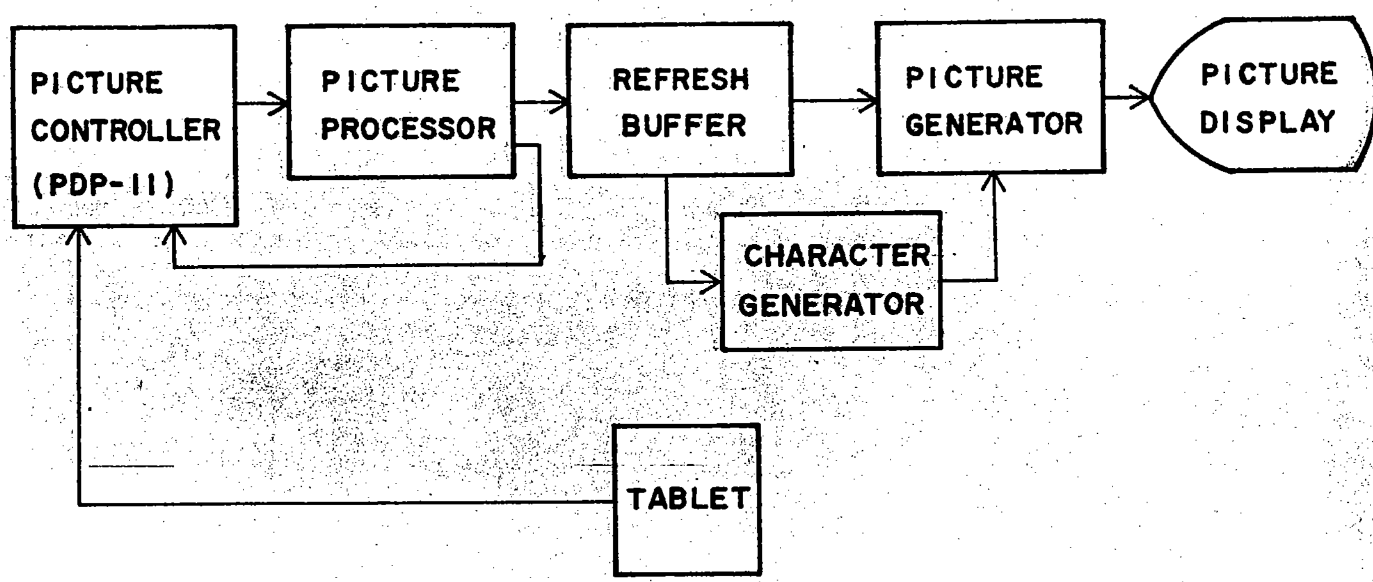


Figure 3-1

The Standard Configuration of THE PICTURE SYSTEM

3.1 THE PICTURE CONTROLLER

The Picture Controller in THE PICTURE SYSTEM is a Digital Equipment Corporation PDP-11 computer. The PDP-11 is a powerful 16-bit general purpose computer which provides the capability of interfacing a large number of peripheral devices for standard system support as well as options for specialized data acquisition or communications applications. In addition, extensive software consisting of paper tape, DECtape and disk systems is available for the PDP-11 family of computers. Software available also includes a Text Editor, Macro Assembler, Linker, File Utility Packages, Debugging Packages and higher level languages including BASIC and FORTRAN. The availability of these software systems and the Graphics Software Package provided with THE PICTURE SYSTEM enables the PDP-11 to act as the Picture Controller.

The Picture Controller is used to:

- Contain the data base which describes the object(s) to be viewed.
- Control the processing of the object coordinate data by the Picture Processor.
- Perform all input and output required to facilitate graphical interaction.
- Compute parameters for use in simulation of object movement, data representation, etc.
- Perform all standard operating functions required by the operating system under which the contrcl program executes.

The Picture Controller communicates with the Picture Processor by an Interface Channel. By means of this interface, all commands and data are communicated to the Picture Processor, Refresh Buffer and Picture Generator.

3.2 THE PICTURE PROCESSOR

The Picture Processor is controlled by the Picture Controller through the use of the graphics software provided with THE PICTURE SYSTEM. The use of this software provides control over the three basic units of the Picture Processor:

1. Interface Channel
2. Matrix Arithmetic Processor
3. Terminal Control

3.2.1 Interface Channel

The Interface Channel contains registers which provide status and commands to the Picture Processor. This interface also handles all data transfers to and from the Matrix Arithmetic Processor.

3.2.2 Matrix Arithmetic Processor

The Matrix Arithmetic Processor consists of a Transformation Matrix, a Transformation Matrix Stack, an Arithmetic Unit and a Parameter Register File.

The Transformation Matrix is a 4x4 element matrix, where each element is a 16-bit word. This 4x4 matrix is used to transform object coordinate data. It can also be concatenated with other 4x4 matrices to obtain a combined transformation.

The Transformation Matrix Stack is a storage area where up to four 4x4 element matrices may be "stacked" or saved for future recall.

The Arithmetic Unit performs all arithmetic operations in the Picture Processor. This includes subtraction, addition, multiplication, division and normalization.

The Picture Processor contains an array of 16-bit registers into which parameters specifying viewport boundaries, scale factors, etc. are stored and may be retrieved.

The Picture Processor utilizes these units to perform digital operations on the data received from the Picture Controller.

These operations are:

- To process two-dimensional data.
- To process three-dimensional data.
- To push the Transformation Matrix onto

the Matrix Stack.

- To pop the top 4x4 matrix of the Matrix Stack into the Transformation Matrix.
- To load the Transformation Matrix with data from the Picture Controller's memory.
- To store the contents of the Transformation Matrix into the Picture Controller's memory.
- To concatenate the contents of the Transformation Matrix with a 4x4 matrix in the Picture Controller's memory to obtain a compound transformation.
- To load and store the registers of the Picture Processor.
- To check transformed coordinate data for visibility by comparison with a two- or three-dimensional viewing window. Lines or portions of lines outside the window are removed by a clipping process so that only visible segments are processed further. At this point three-dimensional data is converted to two dimensions by computing perspective or orthographic views.
- To perform a linear mapping of points from the object's coordinate system into that of the Picture Display.

Each data coordinate that is transformed may be written into the Refresh Memory by the Terminal Control to become a portion of the new frame.

3.2.3 Terminal Control

The Terminal Control is the unit of the Picture Processor that controls the refresh of pictures seen on the Picture Display. The function of the Terminal Control is to receive data from the Matrix Arithmetic Processor and store it in the write portion of the Refresh Buffer. It is usually concurrently reading data from the read portion of the Refresh Buffer and sending it to the Picture Generator.

3.3 THE REFRESH BUFFER

The Refresh Buffer is a memory (distinct from the Picture Controller's) into which processed data is deposited still in digital form. This data represents the picture to be displayed on the Picture Display. For each frame refresh, the Terminal Control reads the data in the Refresh Buffer and passes the data to the Picture Generator, where the data is converted to analog signals to drive the Picture Display. Character strings from the Picture Controller pass through the Picture Processor unmodified and are deposited in the Refresh Buffer as packed character codes.

The Refresh Buffer may be operated in single or double buffer mode under program control. In single buffer mode, the entire Refresh Buffer is used to store a single display frame. In this mode, display refresh may be initiated from partially updated display frame. In double buffer mode, one half of the refresh buffer is designated as an old frame and one half a new frame. Display refresh is then initiated from the old frame, while the new frame is being constructed. When the construction of the new frame is complete, the frame buffers are swapped and the newly constructed frame is displayed and the space occupied by the old frame becomes available for new frame construction.

3.4 CHARACTER GENERATOR

Character strings from the Picture Controller pass through the Picture Processor unmodified and are deposited in the Refresh Buffer as packed character codes. When character words are read out of the Refresh Buffer, the Terminal Control recognizes these codes and calls upon the Character Generator to access a read-only memory containing character stroking data. The strokes are read out of the read-only memory one by one, multiplied by a pre-specified sizing parameter, and drawn by the Picture Generator on the Picture Display.

3.5 THE PICTURE GENERATOR

The Picture Generator receives digital data consisting of x,y coordinate and intensity information read from the Refresh Memory by the Terminal Control Unit. This digital data is converted by the Picture Generator into analog signals and used to draw the picture on the Picture Display.

3.6 THE PICTURE DISPLAY

The Picture Display receives analog signals from the Picture Generator which are used for electron beam positioning and intensity control. The Picture Generator controls beam positioning and the drawing of all vectors and dots on the Picture Display.

3.7 DATA INPUT

All data is input directly to the Picture Controller in THE PICTURE SYSTEM. Data may be input by any of the various standard peripherals available with the PDP-11 or by any of the standard graphical input devices available with THE PICTURE SYSTEM. There are four graphical input devices supported by THE PICTURE SYSTEM:

1. Tablet
2. Control Dials
3. Function Switches & Lights
4. Alphanumeric Keyboard

The use of these standard graphical input devices provides all the capabilities normally required for graphical interaction with THE PICTURE SYSTEM. The appropriate use of these interactive devices along with the dynamic qualities of THE PICTURE SYSTEM provide the user with all of the tools required for a three-dimensional, truly interactive graphics system.

3.7.1 Tablet

The Tablet serves as the standard, general purpose graphic input device in THE PICTURE SYSTEM. The Tablet can be used for positioning or pointing to the picture elements by use of a pen whose x,y coordinates are read by the Picture Controller. A "cursor" may be drawn on the Picture Display to indicate the position of the pen on the Tablet. With these capabilities, the Tablet and pen can perform the interactive functions usually reserved for such graphic input devices as light pens, joy sticks and function switches. The Tablet is fully software supported under the Graphics Software Package provided with THE PICTURE SYSTEM.

3.7.2 Control Dials

Control Dials are available with THE PICTURE SYSTEM which permit the user to dynamically vary values which may be used to control angles of rotation, scaling factors, velocity rates, etc.

3.7.3 Function Switches & Lights

Function Switches & Lights are available with THE PICTURE SYSTEM to provide the capability for the user to utilize switches to be used for functions assigned under program control. An additional capability available with the switches is that the lights (one per switch) which may be used to indicate function switch polarity or for displaying programmed information.

3.7.4 Alphanumeric Keyboard

The Alphanumeric Keyboard available with THE PICTURE SYSTEM is a standard 61 key, 128 character keyboard which may be used for text or data input to the Picture Controller for graphical interaction or other functions required by the user.

4. THE PICTURE SYSTEM GRAPHICS SOFTWARE PACKAGE

The Graphics Software Package furnished with THE PICTURE SYSTEM consists of a set of FORTRAN-callable subroutines written for the Digital Equipment Corporation PDP-11 computer using the MACRO-11 assembly language. These subroutines are written with the intent of providing a user with the full capabilities of THE PICTURE SYSTEM without the necessity of the user to interface, on a system level, with THE PICTURE SYSTEM hardware. These subroutines provide the general user with the facilities necessary for writing interactive computer graphics programs without the need to comprehensively understand the matrix arithmetic utilized within THE PICTURE SYSTEM Processor. Instead, the user merely "calls" a subroutine to perform a required graphical function; i.e. TRANslate, ROTate, SCALE, read TABLET information, display CURSOR, display TEXT, etc.

The graphics subroutines for THE PICTURE SYSTEM have been written utilizing the PDP-11 FORTRAN calling sequence convention of the PDP-11 FORTRAN compiler V06. This calling sequence convention, supported under the DEC RT-11, DOS/BATCH, RSX-11M and RSX-11D operating systems, provides the user the flexibility of utilizing argument lists that are reentrant or non-reentrant in form.

All FORTRAN-callable PICTURE SYSTEM subroutines use the standard call by name (as opposed to call by value) parameter passing technique and specify the non-reentrant inline form of calling sequence[1]. Those subroutines which are not FORTRAN-callable specify no FORTRAN calling sequence.

THE PICTURE SYSTEM Graphics Software Package may be separated into two sets of subroutines:

(1) user subroutines
(2) system subroutines

---

[1]The non-reentrant in-line form of calling sequence need not be used for the Graphics Software Subroutines. Reference Appendix C for specific details of alternate forms of calling sequences.

The user subroutines provide all the capabilities required for the general graphical application programmer. The system subroutines are utilized to implement the user subroutines and are available to the programmer who desires to interface with the system software directly.

The user subroutines provided are:

PSINIT
VWPORT
WINDOW
ROT
TRAN
SCALE
PUSH
POP
DRAW2D
DRAW3D
CHAR
TEXT
INST
MASTER
DASH
BLINK
SCOPE
TABLET
ISPDWN
CURSOR
HITWIN
HITEST
NUFRAM
SETBUF
PSWAIT

The system subroutines provided are:

BLDCON
P$AVE
P$TORE
P$DMA
I$MATX
ERROR
P$DIV
P$MUL

A detailed description of each subroutine is contained in the following sections. Chapter 5 should be referenced for specific examples of the use of these subroutines.

4.1 THE GRAPHICS SUBROUTINES

This section describes in detail the subroutines which comprise THE PICTURE SYSTEM Graphics Software Package. The calling sequence for each of the subroutines and the valid parameter values for each of the arguments is listed. The specification of optional arguments is denoted by the inclusion of the argument in brackets [arg]. Arguments that may be omitted entirely are designated by [,arg]. In particular, the inclusion of a scaling factor [,IW] should always be considered to be optional. In this manner the user familiar with the homogeneous coordinate system of matrix manipulation has the freedom of utilizing the increased range of data values provided by this technique, while the user who is unfamiliar with the technique or who has no need to utilize it may use the shorter calling sequence. For a further description of the use of the homogeneous coordinate [IW] refer to Section 5.2.

Appendix B contains a summary of all FORTRAN and MACRO-11 assembly calling sequences for each of the subroutines described here.

4.1.1 User Subroutines

PSINIT

The PSINIT subroutine is called to initialize THE PICTURE SYSTEM hardware and software. The initialization process includes the following:

THE PICTURE SYSTEM Real Time Clock interrupt handler is connected to the interrupt vector and set to provide automatic refresh of the old frame and timing for frame update at the intervals specified by the calling argument list.

All variables are assigned their default values. All registers used in the Picture Processor are initialized for two-dimensional drawing mode. The Picture Processor is set to display data unrotated, untranslated, at full brightness, within a viewport which just fills the display screen.

A window is set to include the entire definition space (±32767).

The Refresh Buffer is set to double buffer mode with an initial frame consisting of a null cursor. The Picture Generator status is initialized to solid, 0.28 inch character size, and horizontal character mode.

All Picture Displays (scopes 1-4) are selected for output.

FORTRAN Calling Sequence:

[EXTERNAL ERRSUB]

```
CALL PSINIT (IFTIME,INRFSH,[ICLOCK],[ERRSUB],[ISTKCT],
        [ISTKAD][,IFMCNT])
```

where:

IFTIME is an integer used to designate the number of 1/120 second intervals per frame refresh. The refresh rates that may be obtained are:

IFTIME=1 for 120 frames per second
IFTIME=2 for 60 frames per second
IFTIME=3 for 40 frames per second
IFTIME=4 for 30 frames per second

INRFSH is an integer which specifies the number of frame refreshes which must be completed before a frame update will be recognized. If INRFSH contains a value ≤0, then frame update will be allowed upon the next refresh interval after a new frame has been requested.

ICLOCK is an integer variable which, if specified, is incremented upon each frame refresh. This provides the user with the ability to display items for given lengths of time synchronized to the refresh rate.

ERRSUB is a subroutine supplied by the user which is called using the standard FORTRAN calling sequence upon the occurrence of a PICTURE SYSTEM error. One argument is passed to the user's error subroutine specifying the PICTURE SYSTEM subroutine in which the error occurred and the particular error condition encountered. The argument is of the following form:

BYTE 0: PICTURE SYSTEM Subroutine Identifier (0-22).
BYTE 1: Error condition code.

The specification of the user error subroutine is optional. The system subroutine PSERRS will be called if the user error subroutine is omitted from the marameter list.

ISTKCT is an integer which specifies the number of 16-word continuous arrays allocated as software matrix stack area. The amount of matrix stack area that need be allocated by the user is dependent upon the level of Picture Processor Matrix Transformations that are pushed onto the matrix stack (using the PUSH subroutine) by the user. This argument need be specified only if the number of matrix transformations that need be stacked exceeds four, the number implemented with the Picture Processor.

ISTKAD is an integer array allocated as software matrix stack area. This contiguous area need be 16*ISTKCT words in length. If ISTKCT contains the value 0 or is not specified, then this argument will not be utilized.

IFMCNT is an integer variable which, is specified, will be incremented upon each refresh interval by the number of 1/120 seconds that have elapsed since the last frame refresh. This provides the user with the ability to determine the frame update rate for given display segments.

ASSEMBLY CALLING SEQUENCE:

PSINIT, as well as all FORTRAN-callable subroutines, may be called in assembly language by following the FORTRAN calling sequence convention, described in Appendix C. To illustrate this, the assembly calling sequences for PSINIT are shown here. The other graphics subroutines described in this section may be called in a similar manner using assembly language.

EXAMPLE 1: 6-word Parameter List

```
        MOV     #ADR,R5      ;MOVE THE ADDRESS OF THE ARGUMENT
                             ;LIST TO R5
        JSR     PC,PSINIT    ;JUMP TO THE SUBROUTINE
ADR:    BR      .+14.        ;SPECIFY NO. OF PARAMETERS AND
                             ;BRANCH
        .WORD   IFTIME       ;ADDRESS OF REFRESH RATE
        .WORD   INRFSH       ;ADDRESS OF FRAME UPDATE RATE
        .WORD   ICLOCK       ;ADDRESS OF CLOCK INCREMENTAL WORD
        .WORD   ERRSUB       ;ADDRESS OF ERROR SUBROUTINE
        .WORD   ISTKCT       ;ADDRESS OF MATRIX STACK COUNT
        .WORD   ISTKAD       ;ADDRESS OF ARRAY RESERVED FOR STACK
```

EXAMPLE 2: 7-word Parameter List

```
        MOV     #ADR,R5      ;MOVE THE ADDRESS OF THE ARGUMENT
                             ;LIST TO R5
        JSR     PC,PSINIT    ;JUMP TO THE SUBROUTINE
ADR:    BR      .+16.        ;SPECIFY NO. OF PARAMETERS AND
                             ;BRANCH
        .WORD   IFTIME       ;ADDRESS OF REFRESH RATE
        .WORD   INRFSH       ;ADDRESS OF FRAME UPDATE RATE
        .WORD   ICLOCK       ;ADDRESS OF CLOCK INCREMENTAL WORD
        .WORD   ERRSUB       ;ADDRESS OF ERROR SUBROUTINE
        .WORD   ISTKCT       ;ADDRESS OF MATRIX STACK COUNT
        .WORD   ISTKAD       ;ADDRESS OF ARRAY RESERVED FOR STACK
        .WORD   IFMCNT       ;ADDRESS OF REFRESH INTERVAL
                             ;INCREMENTAL WORD
                  .
                  .
                  .

IFTIME:    .WORD    3        ;REFRESH RATE OF 40 FRAMES/SEC
INRFSH:    .WORD    0        ;DYNAMIC UPDATE RATE
ICLOCK:    .WORD    0        ;WORD TO BE INCREMENTED EACH
                             ;REFRESH
ISTKCT:    .WORD    1        ;DEPTH OF USER MATRIX STACK
ISTKAD:    .=.+32.           ;RESERVE 16 WORDS FOR MATRIX STACK
IFMCNT:    .WORD    0        ;WORD TO BE UPDATED EVERY REFRESH
   .                         ;INTERVAL
   .
   .
ERRSUB:                      ;USER'S ERROR SUBROUTINE
```

The user should note that the address of the parameter list is passed to the subroutine in R5 and that the elements of the parameter list are the addresses of the arguments.

ERRORS:

1,0: Invalid number of arguments in the parameter list.
1,1: Invalid parameter values. This error may be caused by:
   IFTIME≤0.
   ISTKCT<0.
   ISTKAD omitted in parameter list for ISTKCT>0.
1,2: Direct Memory Access Error. This is a system error indicating that an error occurred during the last DMA operation.

VWPORT

The VWPORT subroutine is called to set a viewport specified by the values supplied by the calling parameters.

FORTRAN Calling Sequence:

CALL VWPORT(IVL,IVR,IVB,IVT,IHI,IYI)

where:

IVL	is an integer which specifies the viewport left position in display screen (or other output medium) coordinates. Normal range for IVL is -2048 to 2047.
IVR	is an integer which specifies the viewport right position in display screen (or other output medium) coordinates. Normal range for IVR is -2048 to 2047.
IVB	is an integer which specifies the viewport bottom position in display screen (or other output medium) coordinates. Normal range for IVB is -2048 to 2047.
IVT	is an integer which specifies the viewport top position in display screen (or other output medium) coordinates. Normal range for IVT is -2048 to 2047.
IHI	is an integer which specifies the display intensity at the hither clipping plane. The normal range for IHI is 255 for full intensity to 0 for no intensity.
IYI	is an integer which specifies the display intensity at the yon clipping plane. The normal range for IYI is 255 for full intensity to 0 for no intensity.

ERRORS:

3,0: Invalid number of arguments in the parameter list.

WINDOW

The WINDOW subroutine concatenates a two-dimensional or three-dimensional windowing transformation to the Picture Processor Transformation Matrix. This subroutine can be used to perform two-dimensional windowing, orthographic projection or a true perspective transformation of data. The windowing transformation is constructed from the arguments specified in the parameter list.

FORTRAN Calling Sequence:
For two-dimensional windowing:

```
    CALL WINDOW(IWL,IWR,IWB,IWT[,IW])
```

For three-dimensional windowing:

```
    CALL WINDOW(IWL,IWR,IWB,IWT,IWH,IWY[,IE[,IW]])
```

where:

IWL	is an integer which specifies the scaled window left boundary in definition space coordinates (±32767).
IWR	is an integer which specifies the scaled window right boundary in definition space coordinates (±32767).
IWB	is an integer which specifies the scaled window bottom boundary in definition space coordinates (±32767).
IWT	is an integer which specifies the scaled window top boundary in definition space coordinates (±32767).
IWH	is an integer which specifies the scaled window hither boundary in definition space coordinates (±32767). For two-dimensional windowing, the window front, or hither boundary is 0.
IWY	is an integer which specifies the scaled window yon boundary in definition space coordinates (±32767). For two-dimensional windowing, the window rear, or yon boundary is equal to IW. If this parameter=IWH, the yon boundary is positioned at infinity on the side cf the hither clipping plane opposite the eye so that no yon clipping will be performed.
IE	is an integer which, if specified, is the scaled Z position of the eye. If this parameter is omitted or equals IWH, the eye is positioned at -∞, which produces an orthographic view of the data.
IW	is an integer used to scale the window boundaries and eye position. If the scale factor is omitted, cr is given as zero, it is treated as 32767.

ERRORS:

4,0:	Invalid number of arguments in the parameter list.

rotation specified in the parameter list. The transformation is then concatenated to the Picture Processor Transformation Matrix.

FORTRAN CALLING SEQUENCE:

```
CALL ROT(IANGLE,IAXIS)
```

where:

IANGLE is an integer which specifies the angle of rotation. The angle is given by dividing a circle into 2^{16} equal parts, with zero being equal to zero degrees and -2^{15} equaling 180 degrees. Two's complement addition, ignoring overflow, causes the angle to increase counter-clockwise through 360 degrees, when viewed along the specified axis in the positive direction.

IAXIS[1] is an integer which specifies the axis of rotation. Valid values for IAXIS are:

1 for rotation about X axis.
2 for rotation about Y axis.
3 for rotation about Z axis.

ERRORS:

9,0: Invalid number of arguments in the parameter list.
9,1: Invalid argument specified for the axis of rotation.

[1]THE PICTURE SYSTEM software is designed for a left-handed coordinate system.

TRAN

The TRAN subroutine is called to build a translation transformation based on the X, Y and Z translational values specified in the parameter list. The transformation is then concatenated to the Picture Processor Transformation Matrix.

FORTRAN Calling Sequence:

```
CALL TRAN(ITX,ITY,ITZ[,IW])
```

where:

- ITX is an integer which specifies the scaled X translation value.
- ITY is an integer which specifies the scaled Y translation value.
- ITZ is an integer which specifies the scaled Z translation value.
- IW is an integer which specifies the factor used to scale the translational values. If the scale factor is omitted, or is given as zero, it is treated as 32767.

ERRORS:

8,0: Invalid number of arguments in the parameter list.

SCALE

The SCALE subroutine is called to build a scaling transformation based on the X, Y and Z scaling terms specified in the parameter list. The transformation is then concatenated to the Picture Processor Transformation Matrix.

FORTRAN Calling Sequence:

```
CALL SCALE(ISX,ISY,ISZ[,IW])
```

where:

ISX	is an integer which specifies the X scaling value.
ISY	is an integer which specifies the Y scaling value.
ISZ	is an integer which specifies the Z scaling value.
IW	is an integer which specifies the factor used to scale the scaling definition values. If the scale factor is omitted, or is given as zero, it is treated as 32767.

ERRORS:

17,0:	Invalid number of arguments in the parameter list.

PUSH

The PUSH subroutine is called to push the current Picture Processor Transformation Matrix onto the matrix stack (hardware or memory stack, dependent on the current stack depth).

FORTRAN Calling Sequence:
CALL PUSH

ERRORS:

6,0: PUSH error (matrix stack overflow). This indicates that the matrix stack requirement has exceeded the amount allocated by the user during the call to PSINIT.

POP

The POP subroutine is called to pop the top element of the matrix stack (hardware or memory stack, dependent on the current stack depth) into the Picture Processor Transformation Matrix.

FORTRAN Calling Sequence:
CALL POP

ERRORS:

7,0: POP error (matrix stack underflow). This indicates that the user has attempted to retrieve a matrix which had not been previously saved (or pushed) cnto the matrix stack.

DRAW2D

The DRAW2D subroutine is called to draw two-dimensional data coordinate points using the drawing mode specified in the parameter list. The data to be drawn is arranged in x,y pairs and is displayed at the intensity specified by the IZ parameter.

FORTRAN Calling Sequence:
CALL DRAW2D(IDATA,INUM,IF1,IF2,IZ[,IW])

where:

IDATA is an integer array (2*INUM words in length) which contains the x,y coordinate points to be drawn. This data will be drawn in the drawing mode specified by the arguments IF1 and IF2 and at the intensity specified by argument IZ.

INUM is an integer which specifies the number of coordinate pairs to be drawn.

IF1 is an integer which specifies the type of draw function to be performed. Valid values for IF1 are:

- 0 = disjoint lines from new position.
- 1 = disjoint lines from current position.
- 2 = connected lines from new position.
- 3 = connected lines from current position.
- 4 = dot at each point.

IF2 is an integer which specifies the mode in which the coordinates are interpreted. Valid values for IF2 are:

- 0 = absolute-relative-relative-relative-etc.
- 1 = relative always.
- 2 = absolute always.

IZ is an integer which specifies the Z position of the x,y coordinate pairs drawn. This Z position is used to compute the intensity of the data to be drawn. A value of IZ=0 will produce lines of maximum intensity when drawn using a two-dimensional window[1].

IW is an integer used to scale the coordinate data. If the scale factor is omitted, or given as zero, it is treated as 32767.

[1]The maximum intensity is specified using the VWPORT subroutine.

ERRORS:

10,0: Invalid number of arguments in the parameter list.

10,1: Invalid parameter value.
This error may be caused by:
  INUM ≤0.
  IF1<0 or >4.
  IF2<0 or >2.
  For IF2=0 or 1, IW does not equal that of the previous draw.

DRAW3D

The DRAW3D subroutine is called to draw three-dimensional data coordinate points using the drawing mode specified in the parameter list. The data to be drawn is arranged in x,y,z triplets and is displayed at the intensity dependent upon the z coordinates and the values specified for the hither and yon planes.

FORTRAN Calling Sequence:
CALL DRAW3D(IDATA,INUM,IF1,IF2[,IW])

where:

IDATA is an integer array (3*INUM words in length) which contains the x,y,z coordinate points to be drawn. This data will be drawn in the drawing mode specified by the arguments IF1 and IF2.

INUM is an integer which specifies the number of coordinate triples to be drawn.

IF1 is an integer which specifies the type of draw function to be performed. Valid values for IF1

0 = disjoint lines from new position.
1 = disjoint lines from current position.
2 = connected lines from new position.
3 = connected lines from current position.
4 = dot at each point.

IF2 is an integer which specifies the mode in which the coordinates are interpreted. Valid values for IF2 are:

0 = absolute-relative-relative-relative-etc.
1 = relative always.
2 = absolute always.

IW is an integer used to scale the coordinate data. If the scale factor is omitted, or given as zero, it is treated as 32767.

ERRORS:

11,0: Invalid number of arguments in the parameter list.
11,1: Invalid parameter value.
This error may be caused by:
INUM ≤0
IF1<0 or > 4.
IF2<0 or > 2.
For IF2=0 or 1, IW does not equal that of the previous draw.

CHAR

The CHAR subroutine is called to update the status used by the Character Generator when characters are to be displayed on the display screen.

FORTRAN Calling Sequence:
CALL CHAR(IXSIZE,IYSIZE,ITILT)

where:

IXSIZE is an integer which specifies the X character size.
IYSIZE is an integer which specifies the Y character size.

Valid values for IXSIZE and IYSIZE are:

0 = .07 inches
1 = .14 inches
2 = .21 inches
3 = .28 inches
4 = .35 inches
5 = .42 inches
6 = .49 inches
7 = .56 inches

The specification of a value <0 or >7 will cause the value to be modified (modulo 8) to a value in the range 0 to 7.

ITILT is an integer which specifies the horizontal/vertical tilt status. Valid values for ITILT are:

ITILT = 0 for horizontal character status.
ITILT ≠ 0 for 90° counter-clockwise character status.

ERRORS:

18,0: Invalid number of arguments in the parameter list.

TEXT

The TEXT subroutine is called to display the text string specified in the parameter list. The display of the text will be from the current beam position and at the intensity associated with the last information displayed. The character status will be that as initialized by PSINIT or updated by the CHAR subroutine if previously called by the user.

FORTRAN Calling Sequence:

```
CALL TEXT(NCHARS,ITEXT)
```

where:

NCHARS is an integer which specifies the number of characters to be displayed.

ITEXT is an integer array which contains the text to be displayed, packed two characters per word, with the right byte to be displayed first (as in a FORTRAN DATA statement).

ERRORS:

12,0: Invalid number of arguments in the parameter list.

INST

The INST subroutine concatenates a two- or three-dimensional instancing transformation to the Picture Processor Transformation Matrix. This subroutine is used, in conjunction with the MASTER subroutine, to produce multiple instances of an object or symbol. For each desired appearance of the object, the INST subroutine is called to specify the location (and implicitly the size) of that appearance; then the user-supplied routine describing the object is called to display the object previously defined within a two-dimensional or three dimensional enclosure. The INST subroutine pushes the initial Transformation Matrix onto the Transformation Stack before concatenating the instancing transformation, so that it may be restored (POPped) by the user after the object has been drawn.

FORTRAN Calling Sequence:

For two-dimensional instancing:

```
    CALL INST(INL,INR,INB,INT[,IW])
```

For three-dimensional instancing:

```
    CALL INST(INL,INR,INB,INT,INH,INY[,IW])
```

where:

- INL is an integer which specifies the scaled instance left boundary in definition space coordinates (±32767).
- INR is an integer which specifies the scaled instance right boundary in definition space coordinates (±32767).
- INB is an integer which specifies the scaled instance bottom boundary in definition space coordinates (±32767)
- INT is an integer which specifies the scaled instance top boundary in definition space coordinates (±32767).
- INH is an integer which specifies the scaled instance hither boundary in definition space coordinates (±32767). For two-dimensional instancing the window front or hither boundary is 0.
- IWY is an integer which specifies the scaled instance yon boundary in definition space coordinates (±32767). For two-dimensional windowing the instance rear or yon boundary is equal to IW.
- IW is an integer used to scale the instance boundaries. If the scale factor is omitted, or given as zero, it is treated as 32767.

ERRORS:

5,0: Invalid number of arguments in the parameter list.

MASTER

The MASTER subroutine concatenates a two-dimensional or three-dimensional master transformation to the Picture Processor Transformation Matrix. This subroutine is used in conjunction with the INST subroutine for instancing of data. The master transformation is constructed from the arguments specified in the parameter list.

FORTRAN Calling Sequence:

For a two-dimensional master:

```
CALL MASTER(IML,IMR,IMB,IMT[,IW])
```

For a three-dimensional master:

```
CALL MASTER(IML,IMR,IMB,IMT,IMH,IMY[,IW])
```

where:

IML	is an integer which specifies the scaled master left boundary in definition space coordinates (±32767).
IMR	is an integer which specifies the scaled master right boundary in definition space coordinates (±32767).
IMB	is an integer which specifies the scaled master bottom boundary in definition space coordinates (±32767).
IMT	is an integer which specifies the scaled master top boundary in definition space coordinates (±32767).
IMH	is an integer which specifies the scaled master hither boundary in definition space coordinates (±32767). For a two-dimensional master, the front, or hither, boundary is 0.
IMY	is an integer which specifies the scaled window yon boundary in definition space coordinates (±32767). For a two-dimensional master, the rear, or yon, boundary is equal to IW.
IW	is an integer used to scale the master boundaries. If the scale factor is omitted, or is given as zero, it is treated as 32767.

ERRORS:

4,1: Invalid number of arguments in parameter list.

DASH

The DASH subroutine is called to set the Picture Generator status such that all subsequent lines drawn will be dashed or non-dashed dependent on the value of the argument.

FORTRAN Calling Sequence:
CALL DASH(ISTAT)

where:

ISTAT is an integer which specifies the line mode status.

ISTAT = 0 for solid line mode.
ISTAT ≠ 0 for dash line mode.

ERRORS:

19,0: Invalid number of arguments specified in the parameter list.

BLINK

The BLINK subroutine is called to set the Picture Generator status such that all subsequent lines drawn will blink[1] or will not blink, dependent on the value of the argument.

FORTRAN Calling Sequence:
CALL BLINK(ISTAT)

where:

ISTAT is an integer which specifies the blink/non-blink mode.

ISTAT = 0 for non-blink mode.
ISTAT ≠ 0 for blink mode.

ERRORS:

20,0: Invalid number of arguments in the parameter list.

---

[1] Data drawn in Blink mode will blink at approximately 90 blinks per minute.

SCOPE

The SCOPE subroutine is called to select the Picture Display to which output will be directed.

FORTRAN Calling Sequence:
CALL SCOPE(INUM)

where:

INUM is an integer which specifies the scope unit to select. This will cause the scope selected to be connected for output as well as any previously selected scopes. Valid values for INUM are:

INUM = 1,2,3,4 to select scope units 1,2,3, or 4.
INUM <1 or >4 to deselect all scope unit selections.

ERRORS:

21,0: Invalid number of arguments parameter list.

TABLET

The TABLET subroutine is called to read the current pen position and status in relation to the tablet. The user may also specify initiation of automatic tablet mode. This will cause the current pen position to be updated at each frame refresh. This ability, used in conjunction with the automatic cursor mode, allows completely dynamic cursor tracking irrespective of new frame update rate. It should be noted that once the pen information is updated with the pen down bit set (bit 1), the pen position will not be updated until the user has cleared (zeroed) the pen value word (IPEN) indicating that the pen down position has been read or until the pen is set down again.

FORTRAN Calling Sequence:
CALL TABLET(ISTAT[,IX,IY,IPEN])

where:

ISTAT
: is an integer which specifies the automatic tablet mode:

  ISTAT = 0 for automatic tablet mode off.
  ISTAT ≠ 0 for automatic tablet mode on.
  The four-argument parameter list is required for ISTAT ≠ 0 and optional if ISTAT=0.

IX
: is an integer which is updated with the current X pen position. In automatic tablet mode, this value will be updated upon each frame refresh. The approximate limits of IX are ±32700.

IY
: is an integer which is updated with the current Y pen position. In automatic tablet mode, this value will be updated upon each frame refresh. The approximate limits of IY are ±32700.

IPEN
: is an integer which is updated with the current pen information. Bit 1 will be set if the pen is down and bit 0 will be set if the pen is within proximity of the tablet surface. If bit 1 of IPEN is set then IX and IY will be updated only if the pen is down.

ERRORS:

13,0: Invalid number of arguments in the parameter list.

ISPDWN

ISPDWN (Is Pen DoWN) is a FORTRAN-callable integer function subroutine which may be used to determine whether the pen is down (i.e. pressed against the surface of the tablet). This function routine allows FORTRAN applications programs, which do not have the ability to perform bit testing, to test the pen up/down status.

Typical FORTRAN Calling Sequences:

```
C
C   SET PEN DOWN FLAG
C
       IDOWN = ISPDWN(IPEN)

         or

C
C   IF PEN IS DOWN GO TO 100
C
       IF(ISPDWN(IPEN).EQ.1)  GO TO 100
```

where:

IPEN is an integer which contains the pen information returned by the TABLET subroutine.

ISPDWN(IPEN) = 0 if the pen is not down.
ISPDWN(IPEN) = 1 if the pen is down.

CURSOR

The CURSOR subroutine is called to display a cursor at the position specified by the parameter list. The user may also specify initiation of automatic cursor mode. This will cause a cursor to be displayed upon each frame refresh irrespective of the new frame update rate. The cursor displayed in automatic cursor mode will be at the position specified by the x and y position values and within the viewport that had been specified at the time of the initial CURSOR call. The cursor displayed consists of a cross whose center is at the x and y position specified.

FORTRAN Calling Sequence:
```
CALL CURSOR(IX,IY,ISTAT[,IPEN])
```

where:

IX — is an integer which specifies the x cursor position. In automatic cursor mode, the cursor will be placed at the position specified by the contents of this word at the time of frame refresh. The value of IX should be in the approximate range of ±32767.

IY — is an integer which specifies the y cursor position. In automatic cursor mode, the cursor will be placed at the position specified by the contents of this word at the time of frame refresh. The value of IY should be in the approximate range of ±32767.

ISTAT — is an integer which specifies the automatic cursor mode:

ISTAT = 0 for automatic cursor mode off.
ISTAT ≠ 0 for automatic cursor mode on.

IPEN — is an integer which, if specified, should be the pen information which is returned from the TABLET subroutine. The specification of this parameter allows the cursor to be increased in intensity whenever the pen is down providing visual feedback of the pen status.

ERRORS:

14,0: Invalid number of arguments in the parameter list.

NOTE: In automatic cursor mode, the cursor is displayed the viewport that had been specified at the time of the initial CURSOR call. This is done by saving the addresses of the viewport values in effect at that time. When the cursor is displayed the viewport is set from the values found in these addresses.

HITWIN

The HITWIN subroutine is called to specify a window through which data will be passed to determine whether data is being drawn within a given area. The user specifies an x and y coordinate at which to center a window transformation of the specified size. This window transformation is then pre-concatenated with the transformation in the Picture Processor Transformation Matrix, after first saving the original transformation so that it may be restored after all hit testing has been completed. The Picture Processor status is then set to prohibit all data drawn from being output to the Refresh Buffer. The subroutine then returns to allow the user to draw all data against which hit testing is to be performed.

FORTRAN Calling Sequence:
CALL HITWIN(IX,IY,ISIZE[,IW])

where:

IX	is an integer which specifies the hit window x coordinate. The value of IX should be in the approximate range of ±32700.
IY	is an integer which specifies the hit window y coordinate. The value of IY should be in the approximate range of ±32700.
ISIZE	is an integer which specifies the hit window half size. This parameter is used to determine whether lines pass within a given distance (ISIZE) of the specified point (IX,IY).
IW	is an integer used to scale the hit window parameters. If the scale factor is omitted, or is given as zero, it is treated as 32767.

ERRORS:

15,0:	Invalid number of arguments in the parameter list.

HITEST

The HITEST subroutine is called to determine if any output data has passed within a pre-specified hit window (see HITWIN). The procedure for this test is of the form:

1. CALL HITWIN to set up the desired hit window.
2. Draw data (DRAW2D and/or DRAW3D) for comparison against that window.
3. CALL HITEST to determine if there was a "hit".
4. Repeat steps 2 and 3 as often as necessary, setting HITEST argument 2 to a non-zero value on the last call to HITEST to restore the former user transformation.

FORTRAN Calling Sequence:

```
CALL HITEST(IHIT,ISTAT)
```

where:

IHIT is an integer which is set to zero by the HITEST subroutine if there has been no hit or set to one if there has been a hit.

ISTAT is an integer supplied by the user which indicates whether the hit testing has been completed or not.

ISTAT = 0 for intermediate hit test.
ISTAT ≠ 0 for final hit test.

The Picture Processor Transformation Matrix will be restored to the transformation that existed before the call to the HITWIN subroutine and the Picture Processor status reset so that all subsequent data drawn will be sent to the Refresh Buffer.

ERRORS:

16,0: Invalid number of arguments in the parameter list.

NUFRAM

The NUFRAM subroutine is called to initiate the switch from displaying the old frame data to displaying the new frame data (the actual frame switch does not occur until the appropriate refresh interval).

FORTRAN Calling Sequence:
CALL NUFRAM

ERRORS:
None

SETBUF

The SETBUF subroutine is called to set the Refresh Buffer to single- or double-buffer mode. Once the Refresh Buffer has been set to a mode, it may be reset at any time to the other mode. The user need call this subroutine only if the Refresh Buffer is used in single buffer mode. PSINIT during the initialization process sets the Refresh Buffer to the default double buffer mode.

FORTRAN Calling Sequence:
CALL SETBUF(ISTAT)

where:

ISTAT   is an integer which specifies the mode the Refresh Buffer is to be set to.
Valid values for ISTAT are:

1 = single buffer mode.
2 = double buffer mode.

ERRORS:

22,0:   Invalid number of arguments in the parameter list.
22,1:   Invalid parameter value.
This error may be caused by:
ISTAT <1 or >2.

PSWAIT

The PSWAIT subroutine is called whenever it is necessary to wait until the Picture Processor and Direct Memory Access Unit have completed their present operations before continuing. This is used to insure that the data transfer to or from the Picture Controller's memory is complete before the data is referenced or modified.

FORTRAN Calling Sequence:
CALL PSWAIT

ERRORS:
None

4.1.2 System Subroutines

BLDCON

The BLDCON subroutine is called to perform all transformation operations and matrix manipulations.

FORTRAN Calling Sequence:
CALL BLDCON(ITYPE,IARRAY)

where:

ITYPE is an integer which specifies the type of call. Valid values for ITYPE and the operation performed for each are:

- 0= Initialize matrix stack pointer and stack length.
- 1= Load the Transformation Matrix from the 16-word array specified as argument 2.
- 2= Concatenate the Transformation Matrix with the 16-word array specified as argument 2.
- 3= Store the Transformation Matrix into the 16-word array specified as argument 2.
- 4= Pop the top element of the matrix stack into the Transformation Matrix.
- 5= Push the Transformation Matrix onto the matrix stack.

IARRAY is an integer array (16 words in length) which is used as specified by argument 1. This argument must be a 16-word array for only those operations which utilize this parameter (operations 1, 2 and 3).

ERRORS:

- 0,0: Invalid number of arguments in parameter list.
- 0,1: Invalid parameter value (ITYPE < 0 or > 5).
- 6,0: PUSH error (matrix stack overflow). This indicates that the matrix stack requirements have exceeded the amount allocated by the user during the call of PSINIT.
- 7,0: POP error (matrix stack underflow). This indicates that the user has attempted to retrieve a matrix which had not been previously saved (i.e. PUSHed) onto the matrix stack.

P$AVE

The P$AVE subroutine is called to save registers R0-R5 on the program stack.

Assembly Calling Sequence:

```
JSR     PC,P$AVE
```

R$TORE

The R$TORE subroutine is called to restore registers R0-R5 from the program stack.

Assembly Calling Sequence:

```
JSR     PC,R$TORE
```

P$DMA

The P$DMA subroutine is called to initiate a Direct Memory Access (DMA) transfer and check for the correct completion of the operation.

Assembly Calling Sequence:

```
R0 = Repeat Status Register (RSR) value
R1 = DMA word count value
R2 = DMA base address for transfer

JSR     PC,P$DMA
```

ERRORS:

1,2: DMA error. This indicates that an error occurred in the last Direct Memory Access operation.

I$MATX

The I$MATX subroutine is called to initialize a 16-word array in memory (P$MATX) to a 4x4 identity matrix.

Assembly Calling Sequence:

```
JSR     PC,I$MATX
```

ERRORS:
None

ERROR

The ERROR subroutine is called by all PICTURE SYSTEM subroutines that encounter an error condition during the course of execution. This subroutine in turn calls the user error subroutine specified in the call to PSINIT or the default system error routine.

Assembly Calling Sequence:

```
JSR     PC,ERROR
.BYTE   ICODE,IERR
```

where:

ICODE is the error code used to indicate the origin of the error detected.[1]

IERR is the error type used to indicate the error condition encountered.

ERRORS:
None

---

[1]Reference Table 4-1 for the subroutine-error code correspondence list.

The following two function subroutines are optimized for the particular PDP-11 hardware configuration.

P$DIV

The P$DIV function subroutine divides the signed divided in R0 and R1 by the signed divisor in R2, leaving the quotient in R0 and the remainder in R1, with R2 undisturbed. The quotient bears the algebraic sign of the division, while the remainder retains the sign of the dividend.

Assembly Calling Sequence:
R0,R1 = Dividend
R2 = Divisor

```
JSR     PC,P$DIV
```

ERRORS:
v=1 (overflow condition code set) if the magnitude of the dividend is not less than half that of the divisor, or if the divisor is zero.

P$MUL

The P$MUL function subroutine mutliplies the signed multiplicand in R0 by the signed multiplier in R2, leaving a signed product in R0 and R1, with R2 undisturbed.

Assembly Calling Sequence:
R0 = Multiplicand
R2 = Multiplier

```
JSR     PC,P$MUL
```

ERRORS:
None

4.2 PICTURE SYSTEM ERRORS

Error detection by the Graphics Software Package is performed to ensure program integrity and to facilitate program debugging. A user may make four types of programming errors that will be detected by the Graphics Software Package. These are:

1. The call of a graphics subroutine with an invalid number of parameters specified.
2. The call of a graphics subroutine with an invalid parameter value.
3. The attempt by the user to PUSH the matrix stack to a depth greater than that specified by the user in the call to PSINIT.
4. The attempt by the user to POP a transformation from the matrix stack which had not been previously PUSHed.

When an error is detected by a graphics subroutine, the system subroutine ERROR is called with an argument that specifies the origin of the error detected and the error condition encountered. The system subroutine ERROR then calls the user error subroutine, specified in the call to PSINIT. When called, the user error subroutine will be passed a parameter which specifies the origin and type of error detected. The error parameter is of the following form:

	BYTE 1	BYTE 0
ICODE,IERR:	IERR	ICODE

where:

- ICODE is the error code used to indicate the origin of the error detected.
- IERR is the error type used to indicate the error condition encountered.

A summary of the error codes and their meaning is contained in Table 4-1. Return from the user error subroutine will result in the termination of the program. If, in the call to PSINIT, the user does not specify an error subroutine, the graphics error subroutine PSERRS will be called. PSERRS, when called, will output the following message to the console terminal:

ERROR X DETECTED IN GRAPHICS SUBROUTINE YY.

and terminate the execution of the program. X and YY (YY,X) are the error codes listed in Table 4-1.

NOTE: Unless the users error subroutine is named PSERRS, the resultant core image created by the LINKER will include the graphics error subroutine PSERRS.

TABLE 4-1
SUBROUTINE INFORMATION

	Subroutine Name	Length[1] Bytes$_{10}$	Length[1] Bytes$_{8}$	Registers Destroyed	Error Codes and Meaning
1.	PSINIT	1250	2342	None	1,0-Invalid No. of Parameters
					1,1-Invalid Parameter
					1,2-Direct Memory Access Error
2.	NUFRAM	(1)	(1)	None	None
3.	VWPORT	(1)	(1)	None	3,0-Invalid No. of Parameters
4.	WINDOW	216	732	None	4,0-Invalid No. of Parameters
5.	MASTER	(4)	(4)	None	4,1-Invalid No. of Parameters
6.	INST	(4)	(4)	None	5,0-Invalid No. of Parameters
7.	PUSH	(1)	(1)	None	6,0-PUSH Error
8.	POP	(1)	(1)	None	7,0-POP Error
9.	ROT	386	602	None	9,0-Invalid No. of Parameters
					9,1-Invalid Parameters
10.	TRAN	150	226	None	8,0-Invalid No. of Parameters
11.	SCALE	138	212	None	17,0-Invalid No. of Parameters
12.	DRAW2D	260	404	None	10,0-Invalid No. of Parameters
					10,1-Invalid Parameter
13.	DRAW3D	(12)	(12)	None	11,0-Invalid No. of Parameters
					11,1-Invalid Parameter
14.	TEXT	188	274	None	12,0-Invalid No. of Parameters
15.	TABLET	188	274	None	13,0-Invalid No. of Parameters
16.	CURSOR	440	662	None	14,0-Invalid No. of Parameters
17.	HITWIN	276	422	None	15,0-Invalid No. of Parameters
18.	HITEST	(17)	(17)	None	16,0-Invalid No. of Parameters
19.	PSWAIT	(1)	(1)	None	None
20.	CHAR	258	402	None	18,0-Invalid No. of Parameters
21.	DASH	(20)	(20)	None	19,0-Invalid No. of Parameters
22.	BLINK	(20)	(20)	None	20,0-Invalid No. of Parameters
23.	SCOPE	(20)	(20)	None	21,0-Invalid No. of Parameters
24.	SETBUF	88	72	None	22,0-Invalid No. of Parameters
					22,1-Invalid Parameter

[1]The numbers in these columns within parenthesis (i.e., (1)) indicate that the subroutine is included as part of the subroutine whose number is in parenthesis.

TABLE 4-2
SYSTEM LEVEL SUBROUTINE INFORMATION

	Subroutine Name	Length[1] $Bytes_{10}$	Length[1] $Bytes_{8}$	Registers Destroyed	Error Codes and Meaning
25.	BLDCON	(1)	(1)	None	0,0-Invalid No. of Parameters 0,1-Invalid Parameter
26	R$TORE	(1)	(1)	R0-R5	None
27.	P$AVE	(1)	(1)	None	None
28.	I$MATX	(1)	(1)	R0,R1,R2	None
29.	P$DMA	(1)	(1)	None	1,0-Direct Memory Access Error
30.	ERROR	(1)	(1)	None	Branch to user error routine or branch to graphics error routine PSERRS
31.	P$DIV	(1)	(1)	R0,R1	Overflow set on error
32.	P$MUL	(1)	(1)	R0,R1	None

---

[1]The numbers in these columns within parenthesis (i.e., (1)) indicate that the subroutine is included as part of the subroutine whose number is in parenthesis.

Because a comprehensive set of system diagnostics is provided with THE PICTURE SYSTEM, hardware error detection is performed to a minimal level. There are, however, three error codes which may indicate a hardware failure. These are:

1,2: Direct Memory Access Error. This indicates that an error occurred during the last Direct Memory Access operation.

6,0: POP error. This error may be induced by a user software error or by a hardware failure in the Transformation Matrix Stack. If this error occurs, an exhaustive software verification should be made. If no software error is apparent, the PUSH/POP diagnostic routine may be run to verify the integrity of the hardware.

7,0: POP error. This error may be induced by user software error or by a hardware failure in the Transformation Matrix Stack. If this error occurs, an exhaustive software verification should be made. If no software error is apparent, the PUSH/POP diagnostic routine may be run to verify the integrity of the hardware.

CHAPTER FIVE

5. PROGRAMMING THE PICTURE SYSTEM

This chapter demonstrates the use of THE PICTURE SYSTEM Graphics Subroutines to perform general purpose graphics functions. The intent of this chapter is not tc provide instruction in programming technique, but rather to illustrate the use of the Graphics Software Package. Each of the user subroutines described in Chapter 4 is used, with typical parameter values, in the programming examples contained in this chapter.

5.1 GENERAL PROGRAM STRUCTURE

Programs written for THE PICTURE SYSTEM generally contain the following segments:

1. Data Definition
2. Program Initialization
3. Display Loop

The Data Definition segment typically contains no executable code, but rather contains the data which is displayed and with which the user interacts during the course of program execution. The Program Initialization segment usually is executed but once during the course of the program, but may provide for initialization of values thus allowing a programmed restart capability. The Display Loop of typical PICTURE SYSTEM applications programs is structured as shown in Figure 5.1-1. This program structure lends itself to the interactive environment of THE PICTURE SYSTEM by providing data input and update of dynamic values for each new frame displayed. The frame update rate, or the time required to complete the execution of the display loop, has been made independent of the frame refresh rate by the Refresh Buffer. This feature allows programs to be time constrained only by the frame update rate required for dynamic motion of the data displayed.

As Figure 5.1-1 illustrates the display loop consists of:

1. Data Input (i.e. status of function switches, etc.
2. Update of Dynamic Values
3. Picture Display
4. Frame Update

This functional program structure insures that:

a. all dynamic values are updated by the most recent data input
b. the most recently updated values are used to create the new frame to be displayed
c. frame update is completed and (for double-buffer mode) the buffers set to be switched allowing data input and the update of dynamic values to proceed while the buffers are waiting to be actually switched.

Item c, above is important in the design of the software the processing power of THE PICTURE SYSTEM

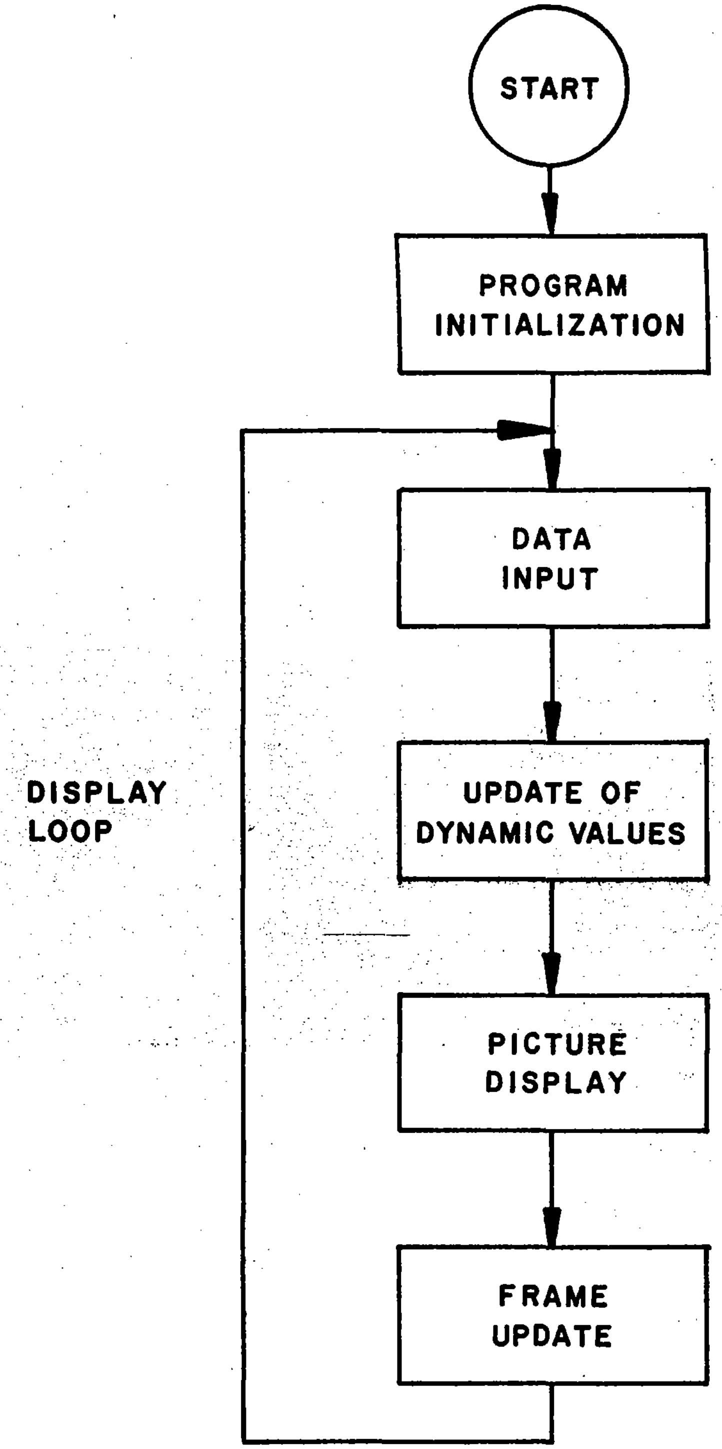


Figure 5.1-1

General Interactive Program Structure

system as it allows maximum utilization of the processing Controller and increases the frame update rate. The program structure of Figure 5.1-1 may be modified as shown in Figure 5.1-2 to increase the frame update rate. A comparison of Figures 5.1-1 and 5.1-2 shows that the difference in program structure is the inclusion of the test for "Data to Input". This test, while not necessary, improves the frame update rate by allowing the data input procedure to be bypassed unless the user initiated some form of data input since the previous frame update. This technique is particularly valuable when used in conjunction with the tablet. In this case, menu selection testing or hit testing need not be done unless the pen is "down", ie.e. touching the surface of the tablet. The TABLET subroutine may be used in automatic or non-automatic mode to perform this function. The user need only test the IPEN parameter to determine whether data input is to be done from the tablet (see Section 5.11).

The program structure of Figure 5.1-1 may be further modified to increase the response of the system to data input and provide that frame update be done only as required. This new program structure, shown in Figure 5.1-3, is a modification of Figure 5.1-2 in that a test for Values to Update" is made prior to the "Update of Dynamic Values". This test allows a more efficient use of the Picture Controller, since a new frame is created <u>only</u> if a portion of the picture is changed. The inclusion of this test is a function of the program design and the particular application of the program. For example, if a picture contains an object which changes with each frame update, the inclusion of the test would be superfluous, but if a picture is essentially static and changes only upon user interaction, the response to user input will be improved by the inclusion of a test of this type. If should be noted that the program structures of Figures 5.1-1 and 5.1-2 create a new frame with each execution of the display loop, whether a new frame creation is necessary or not.

If THE PICTURE SYSTEM is operating in a stand-alone environment in which graphics display and interaction is the only function of the Picture Controller, frame update rate is the only time constraint, and the user program may remain in the display loop in Figures 5.1-1, 5.1-2 or 5.1-3 without concern for processing time. However, if the graphics system shares a Picture

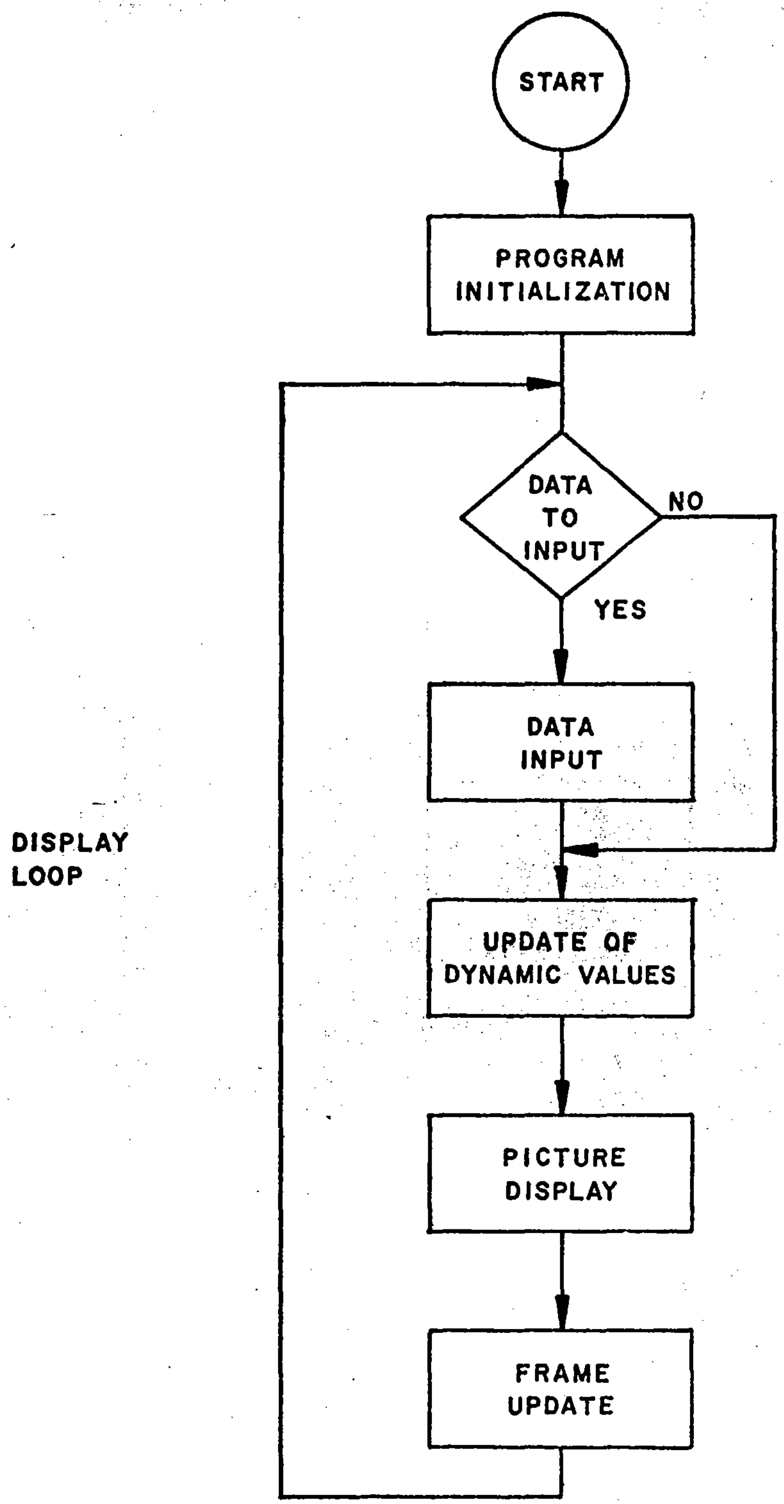


Figure 5.1-2

Program Structure to Increase Frame Update Rate

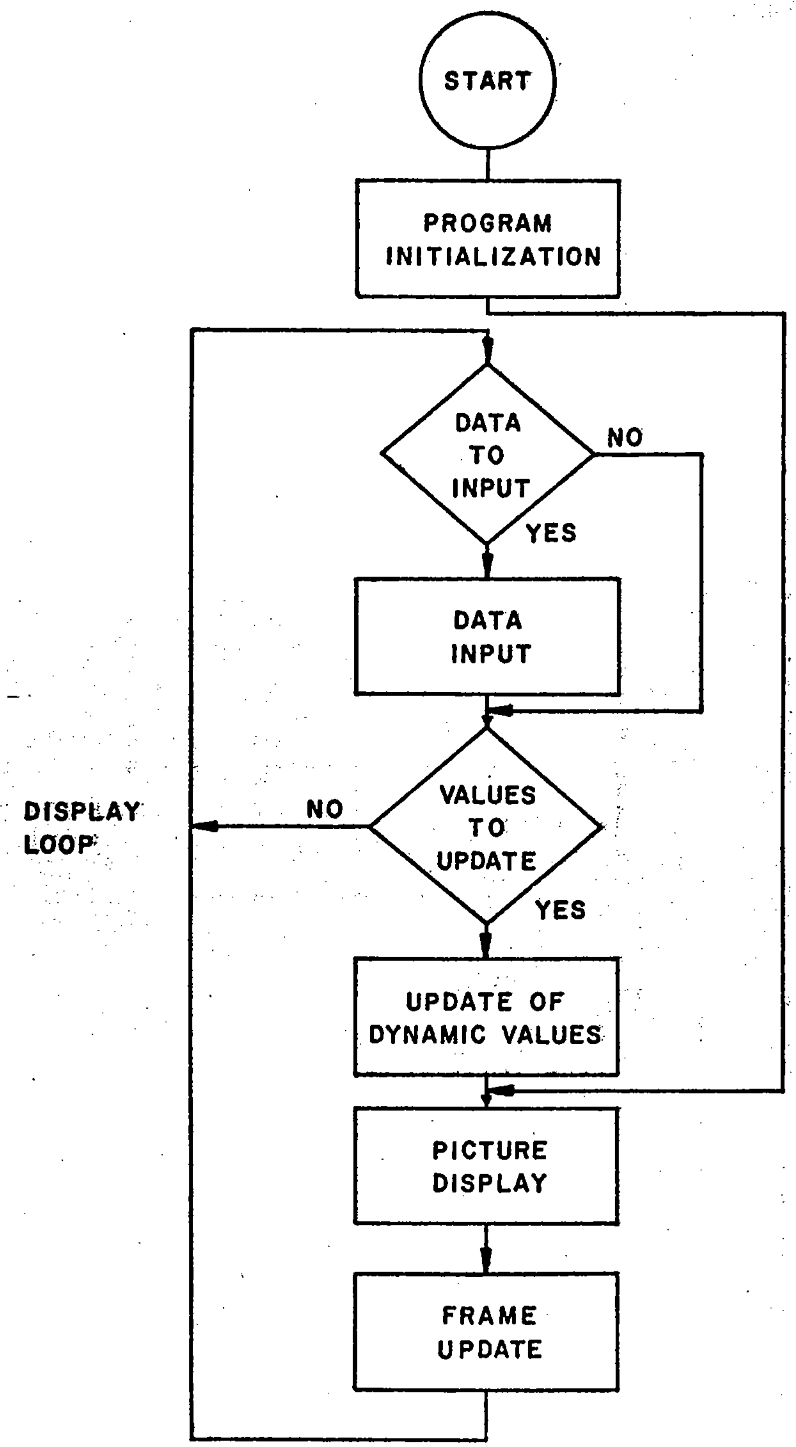


Figure 5.1-3

Program Structure to Increase System Response to User Action

Controller in a Foreground/Background mode[1] of operation, the display loops in these program structures would be disasterous unless the graphics application executed in the Background mode. However, a program in the Background mode may suffer in response time to user interaction, depending upon the Foreground program which is executing. To overcome this difficulty, user programs may wish to utilize the program structure shown in Figure 5.1-4. This structure would allow a graphics program to execute in Foreground mode, with all the priorities and privileges afforded a Foreground program, and yet allow Background programs to execute whenever possible.

---

[1]See RT-11 F/B Operating System Reference Manual, Digital Equipment Corporation.

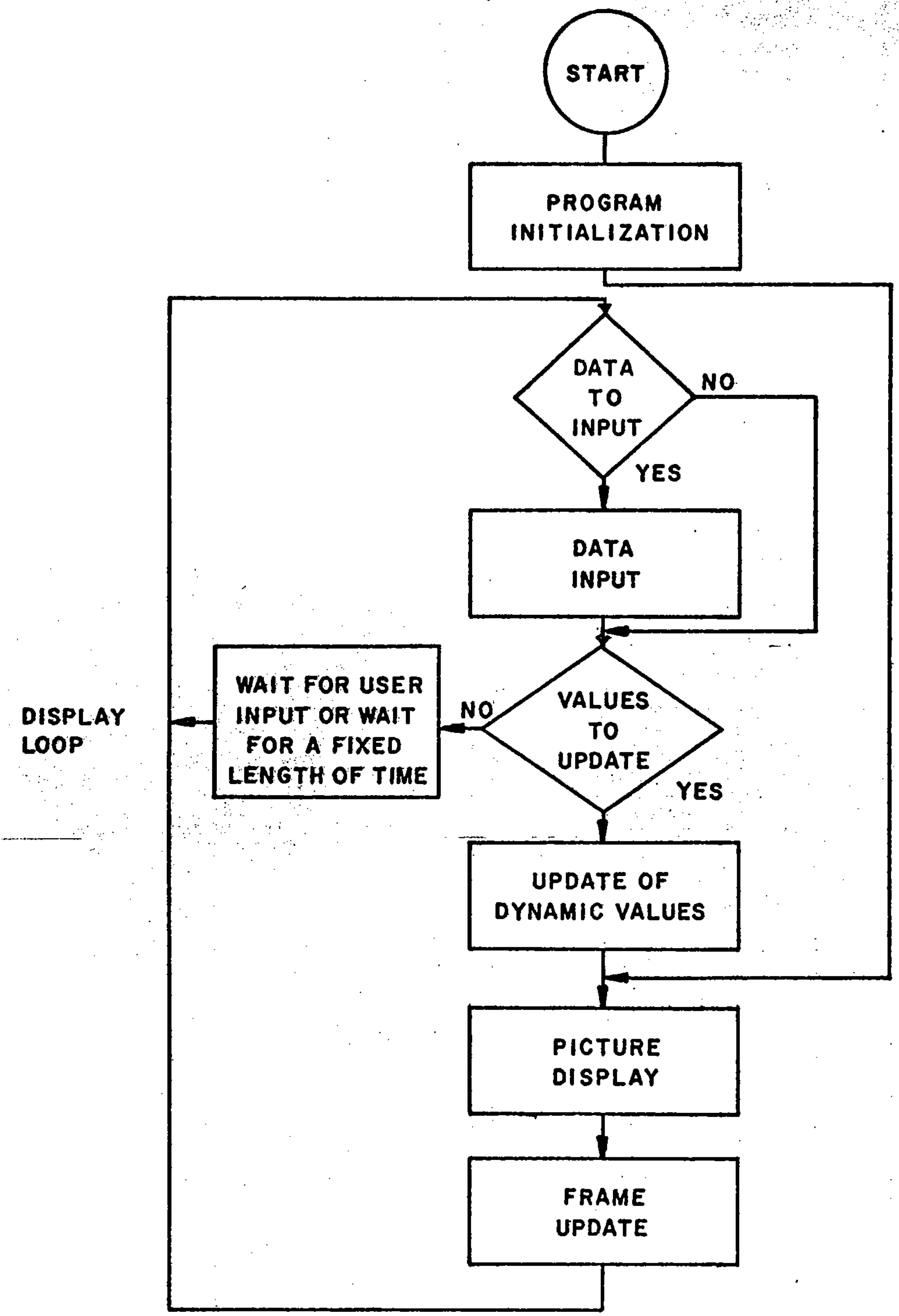


Figure 5.1-4

Program Structure for Foreground Execution

5.2 SCENE DEFINITION

All data that is displayed on THE PICTURE SYSTEM may be considered to be a scene which is viewed by the user. The way in which a scene is constructed is dependent upon the coordinate system the data was defined in, the definition of the data and the transformations which may be applied to the data. The following sections describe the coordinate systems which are available for data definition and display, the manner in which data is defined within these coordinate systems and the transformations and the order in which they should be applied to the data.

5.2.1 Coordinate Systems

The user of THE PICTURE SYSTEM need only be concerned with the data space coordinate system in which the data to be displayed is defined. However, the user may optionally choose to expand the range of the data space available or to provide for convenient scaling of defined data by use of the homogeneous coordinate system. In either case, the image which is ultimately displayed is viewed within the screen coordinate system of the Picture Display. Following is a description of each of these coordinate systems.

5.2.1.1 Data Space Coordinates

The data space coordinate system is the region of definition space in which all data which is to be viewed is defined. The data space by convention is treated as a left handed coordinate system. Thus, positive X increases to the right and positive Y increases upward, while positive Z increases away from the X-Y plane when viewed as in Figure 5.2-1. Any data point may be uniquely represented within this coordinate system by providing the x,y,z coordinates which define the position of the data point in three-space. Within this data space resides all of the parameters which define the windowing boundaries, the eye position for perspective views, the translational values, the scaling values as well as all of the data which is to be viewed. The bounds of the data space are $\pm 2^{15}-1$, but may be extended to an effective range of $\pm 2^{30}$ by using the homogeneous coordinate system.

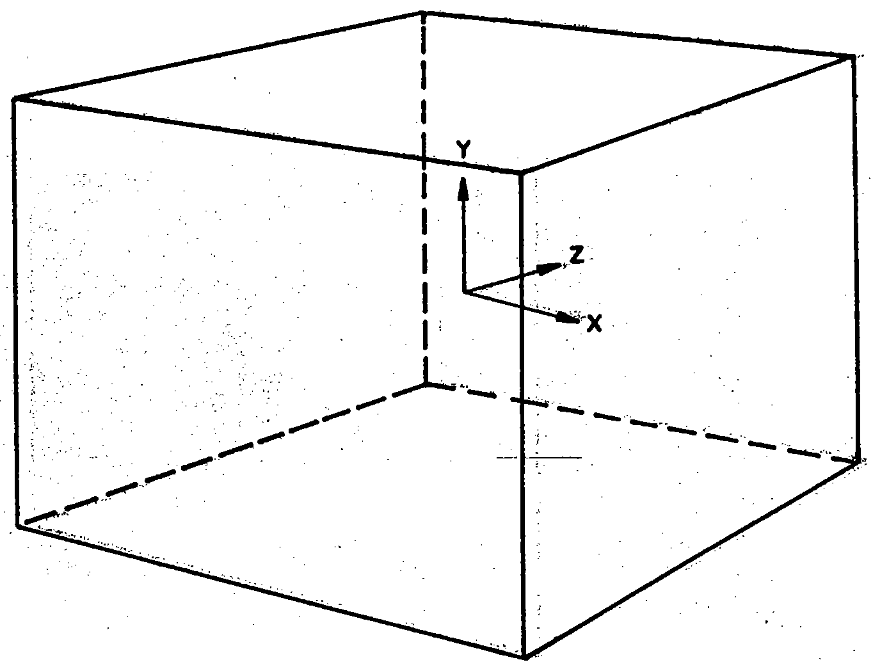


Figure 5.2-1

The Data Space Coordinate System

5.2.1.2 Homogeneous Coordinates

All data defined for use on THE PICTURE SYSTEM is treated by the hardware as homogeneous coordinate data; that is, each data point consists of x,y,z,w coordinates. This coordinate system was made available to the user because the need to express numbers larger than 32767 (the largest expressable integer value of the Picture Controller's 16-bit word size) arises in some applications. The homogeneous coordinate system allows the user the capability of expressing numbers of $\pm 2^{30}$ in magnitude, by representing a point in three dimensions whose coordinates are x,y and z by the four coordinates (h•x,h•y,h•z,h•32767), where "h" is an arbitrary number between zero and one. If each of the numbers x,y,z are less than or equal to 32767 in magnitude, "h" would be made equal to 1 and the expression becomes (x,y,z,32767). But if one of the coordinates of the point is greater than 32767 in magnitude, "h" may be adjusted such that the number is expressable. For example, if the data point (100000,60000,-16000) were to be expressed in homogeneous coordinates so that each of the numbers could be represented by a 16-bit integer, "h" could be chosen to be 1/4 resulting in (1/4•100000,1/4•60000,1/4•-16000,1/4•32767) or (25000,15000,4000,8192). It should be noted that "h" could not be chosen to be 1/2 since this would result in an x coordinate of 50000, again unexpressable as a 16-bit integer. This example illustrates how "h" may be chosen. However, it may be required in some instances to minimize the loss of resolution that results in the conversion of unexpressable numbers to homogeneous coordinates. In these instances, the following formula may be used to compute an "h" which minimizes the resolution loss:

$$h = \frac{32767}{|\text{maximum } x, y \text{ or } z \text{ coordinate}|}$$

In the above example, this would result in:

$$h = \frac{32767}{|100000|} = .32767$$

or the homogeneous coordinates:

(32767,19660,-5243,10737)

Usually though, a convenient value (such as 1/2, 1/4, 1/10, etc.) may be chosen for "h" which yields homogeneous coordinates whose loss of resolution is not significantly greater than if the resolution loss had been minimized.

The previous discussion emphasizes the use of the homogeneous coordinate system to extend the effective range of the data-space. However, the homogeneous coordinate may also be used to define objects according to their own coordinate system and scale. For example, an object which may have been previously defined with 2000 data units/inch as its scale may be required to be displayed in relation to a similar object which had been defined to the scale of 1000 data units/centimeter. One of the objects may be connected to the scale of the other by merely supplying the appropriate homogeneous coordinate (IW) when drawing the data using the DRAW2D or DRAW3D subroutine. To determine the appropriate homogeneous coordinate for this example the following equation would be used:

$$h = \frac{1000 \text{ data units}}{1 \text{ centimeter}} = \frac{2000 \text{ data units}}{1 \text{ inch}}$$

or

$$h = \frac{1000 \text{ data units}}{1 \text{ centimeter}} \cdot \frac{2.54 \text{ centimeters}}{1 \text{ inch}} = \frac{2000 \text{ data units}}{1 \text{ inch}}$$

or

$$h = \frac{2540 \text{ data units}}{1 \text{ inch}} = \frac{2000 \text{ data units}}{1 \text{ inch}}$$

or

$$h = \frac{2000}{2540} = .78740$$

Therefore, the homogeneous coordinate, IW, would be:

$IW = .78740 \bullet 32767 = 25800$

The homogeneous coordinate would then be used to "scale" the data that was previously defined in inches into the centimeter data space, as shown in Example 5.2-1.

```
C
C   DRAW THE CENTIMETER DEFINED DATA
C
       CALL DRAW3D(ICENT,500,0,2)
C
C   DRAW THE "SCALED" INCH DEFINED DATA.
C
       CALL DRAW3D(INCHS,500,0,2,25800)
       CALL NUFRAM
```

Example 5.2-1

As specified in Chapter 4, there are eight subroutines in THE PICTURE SYSTEM Graphics Software in which the user may utilize the homogeneous coordinate system. They are:

WINDOW
TRAN
SCALE
DRAW2D
DRAW3D
MASTER
INST
HITWIN

In each of these subroutines, the inclusion of the homogeneous coordinate, IW, is optional so that the user who has no need to utilize the homogeneous coordinate is not even required to specify the argument in the calling sequence to the subroutine. Those users who initially do not use homogeneous coordinates may easily modify their programs to utilize their capabilities if required at a later time.

5.2.1.3 Screen Coordinates

All data within the data space (homogeneous or not) that is defined for display is ultimately mapped into the screen coordinate system for display by the Picture Generator. This mapping from the dat a space to the screen coordinate system is called the viewport mapping and occurs after the data has been transformed, clipped and the perspective projection performed. This process, accomplished by the hardware of the Picture Processor, is transparent to the user who need be concerned with the screen coordinate system only when specifying viewport boundaries.

The screen coordinate system for the Picture Display of THE PICTURE SYSTEM is shown in Figure 5.2-2. As the figure illustrates the origin of this coordinate system is at the center of the display screen and has a range of -2048 to +2047 display units. This two-dimensional screen coordinate system may be considered a three-dimensional coordinate system whose third dimension is the intensity range of the display. This is shown in Figure 5.2-3. It is within this coordinate system that all viewports are specified. Since viewports may encompass a portion of the screen and pictorial data is mapped within the viewport boundaries (NOT TO THE SCREEN BOUNDARIES), the screen may be used to define multiple viewports. This allows the screen to be used to view a single object in many orientations or many objects simultaneously. The user should be cautioned, however, that should a viewport specification exceed the range of the screen coordinate system, lines mapped to the edges of the viewport will wrap-around to the opposite side of the screen.

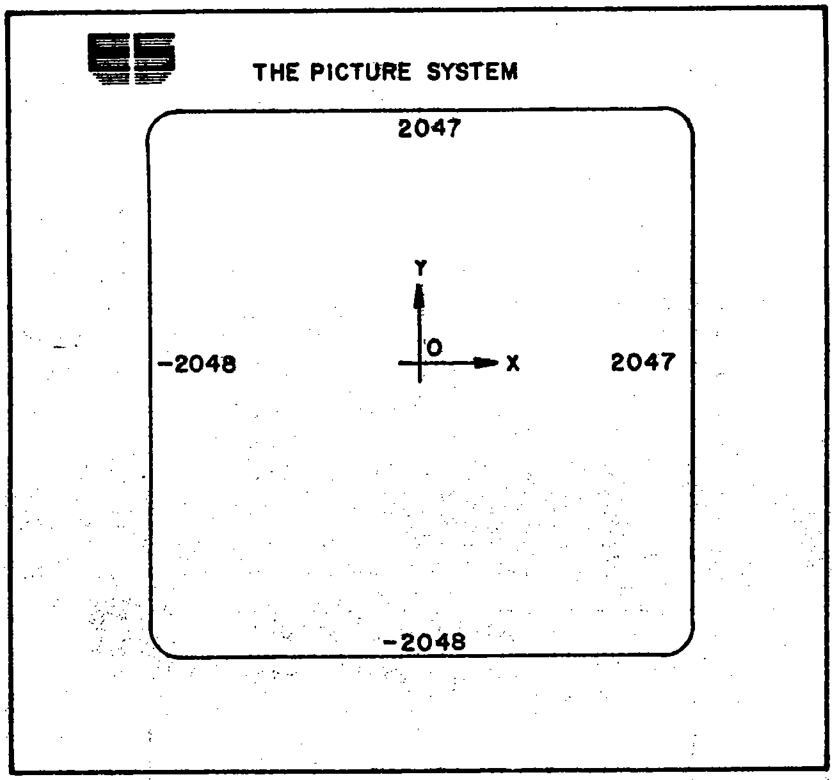


Figure 5.2-2

Screen Coordinate System of the
Picture Display

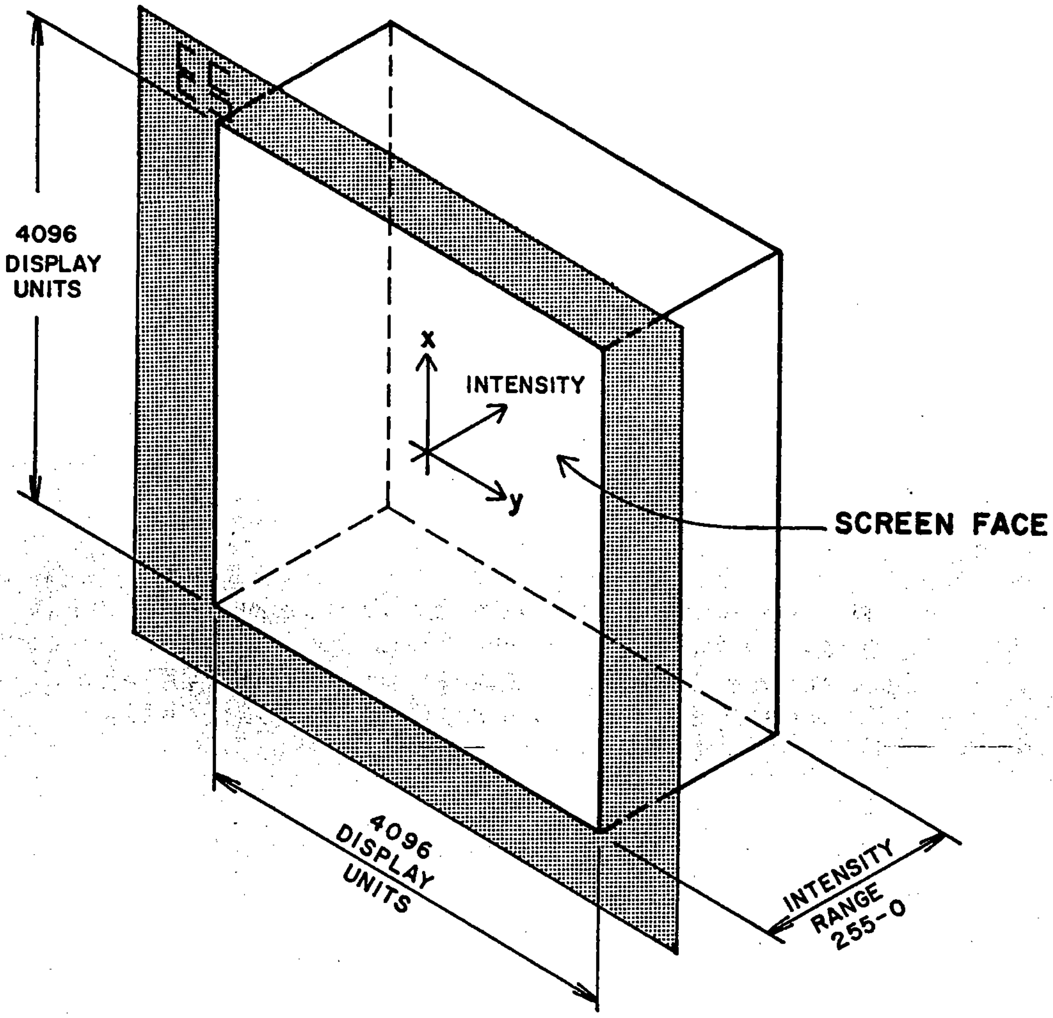


Figure 5.2-3

The Two-dimensional Screen Coordinate System considered as a Three-dimensional System whose Third Dimension is the Intensity Range.

5.2.2 Data Definition

Graphic data that is to be displayed on THE PICTURE SYSTEM is defined in the Picture Controller in the form of what may be termed a data set. A data set is an array of two- or three- dimensional coordinate points that are to be drawn in a particular drawing mode.

All data displayed on THE PICTURE SYSTEM is treated by the hardware as homogeneous coordinate data; that is, each data points consists of x,y,z and w coordinates. Thus two-dimensional data consists of x,y pairs with constant z and w coordinates (two-dimensional data is actually three-dimensional data that resides in a constant z plane), and threedimensional data consists of x,y,z triples with a constant w coordinate. The notation used to represent a data set is illustrated in Figures 5.2-4a and b. All data of a particular data set that is to be displayed should be stored in the memory of the Picture Controller in a contiguous integer array, to facilitate the accessing of the data by the Direct Memory Access (DMA) interface of the Picture Processor. To ensure that data is stored as contiguous data elements in memory, the user should understand the array storage convention of PDP-11 FORTRAN IV, summarized as follows:

> Arrays are stored in contiguous storage locations that are addressed in ascending order with the first subscript varying most rapidly. For instance, the two-dimensional array N(J,K) is stored in the following order:[1]
>
> N(1,1), N(2,1), . . . , N(J,1)
> N(1,2), N(2,2), . . . , N(J,2)
> .
> .
> .
> N(1,K), . . .   , N(J,K)

---
[1]See Reference 3, Part 7, Section 5.3.1 for further details.

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ x_n & y_n \end{bmatrix} \quad \text{with constant } z \text{ and } w \text{ coordinates} \quad = \begin{bmatrix} x_1 & y_1 & z & w \\ x_2 & y_2 & z & w \\ x_3 & y_3 & z & w \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ x_n & y_n & z & w \end{bmatrix}$$

(1) (2)

Figure 5.2-4a

Two-dimensional data showing: (1) the notation of the data set as stored in the memory of the Picture Controller (with implied constant z and w coordinates) and (2) the equivalent homogeneous data set as processed by the Picture Processor.

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ x_n & y_n & z_n \end{bmatrix} \quad \text{with constant } w \text{ coordinate} \quad = \begin{bmatrix} x_1 & y_1 & z_1 & w \\ x_2 & y_2 & z_2 & w \\ x_3 & y_3 & z_3 & w \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ x_n & y_n & z_n & w \end{bmatrix}$$

(1) (2)

Figure 5.2-4b

Three dimensional data showing: (1) the notation of the data set as stored in the memory of the Picture Controller (with implied constant w coordinate) and (2) the equivalent homogeneous data set as processed by the Picture Processor.

This convention should be used in the following manner to ensure that all two- and three-dimensional data is accessed properly:

All two-dimensional data should be stored in an array specified as:

DIMENSION IDATA(2,n)[1]

All three-dimensional data should be stored in an array specified as:

DIMENSION IDATA(3,n)[1]

In this manner the data will appear as:

IDATA(1,i) = xi
IDATA(2,i) = yi

and for the three-dimensional data:

IDATA(3,i) = zi

A data set specified as described above may then be displayed by calling the appropriate display subroutine (DRAW2D or DRAW3D) and providing the drawing specifications. Figures 5.2-5a and b show the calling sequence used to display two- and three-dimensional data. Although the z and w coordinates are constant for a particular data set when used in a DRAW2D call, they may be varied from call to call. In this manner, a two-dimensional data set may reside in any z-plane and all data sets may be scaled (using the w coordinate) by any value. It should be noted by the user that the intensity of a picture displayed is dependent upon the z position of the data in relation to the hither clipping plane (assuming that depth-cueing is being used). Thus to decrease the intensity of a data set, the user need only to increase the distance of the data set from the hither clipping plane (normally the hither clipping plane = 0 for two-dimensional display).

Data that is displayed on the Picture Display is transformed, clipped and mapped to a portion of the display screen (viewport mapped) by the Picture Processor and stored into the Refresh Buffer for display. Because of this, data within the Refresh Buffer is referred to as transformed data and may bear little resemblance to the original data.

---

[1] Similarly, a one-dimensional array may be used to contain two- or three-dimensional coordinate data.

2-dimensional
data set

number of
coordinate pair

type of draw
function

drawing mode

z position

scaling factor
(optional)

```
CALL DRAW2D(IDATA(1,1),N,IF1,IF2,IZ,IW)
```

Figure 5.2-5a

Calling Sequence for Two-dimensional Display of Data

3-dimensional
data set

number of
coordinate trip

type of draw
function

drawing mode

scaling factor
(optional)

```
CALL DRAW3D(IDATA(1,1),N,IF1,IF2,IW)
```

Figure 5.2-5b

Calling Sequence for Three-dimensional Display of Data

5.2.3 Transformations

All data that is displayed on the Picture Display is transformed by multiplying each coordinate point to be drawn by a 4x4[1] matrix which represents the linear transformation to be applied to the data. This process is performed by the Picture Processor hardware, greatly increasing the speed at which the data may be transformed and displayed. The use of linear transformations in the programming of interactive graphics programs is discussed in detail in the following sections.

5.2.3.1 The Identity Transformation

THE PICTURE SYSTEM initialization subroutine, PSINIT, initializes the Picture Processor's Transformation Matrix to a 4x4 identity matrix of the form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

THE PICTURE SYSTEM subroutines which alter the transformation matrix do so by matrix concatenation. Initializing to the identify matrix assures that the first concatenation is equivalent to loading the desired matrix. It should be noted that any homogeneous vector or matrix may have all its elements multiplied by some non-zero scalar quantity without changing its graphic effect at all. Thus, THE PICTURE SYSTEM automatically scales all concatenated matrices to the greatest value short of overflow, in order to preserve arithmetic precision. The 1's in the above matrix, therefore, are shown merely for mathematical clarity, and in fact, subroutine PSINIT uses the value, 16384, in their place.

---

[1]For an in-depth discussion of the properties and theory of matrices and linear transformations, see Reference 2.

5.2.3.2 Simple Linear Transformation

All transformations performed by the graphics subroutines (i.e. WINDOWing, ROTation, TRANSlation and SCAL(E)ing) are simple linear transformations; each are expressable as a 4x4 matrix. When called, the subroutines create a 4x4 matrix to perform the required linear transformation (e.g. ROTate 90°, etc.) and concantenate it with the Picture Processor's Transformation Matrix to form a compound matrix. If the initial contents of the Transformation Matrix was the identity matrix, the resultant compound matrix would be the simple transformation created by the graphics subroutine; otherwise, the compound matrix would be a combination of the transformations previously concatenated and the newly concantenated matrix.

Figure 5.2-6 illustrates the matrix multiplication involved in transforming a data point [x y z w] by the transformation matrix A to get the transformed data point [x'y'z'w']. If data is to be displayed without transformation in any manner, the Transformation Matrix must contain the identify matrix as shown in Figure 5.2-7.

5.2.3.3 Compound Transformations

A compound transformation may be thought of as a series of two or more 4x4 matrices multiplied together as illustrated in Figure 5.2-8.

Typically, all transformations that are to be applied to a given set of PICTURE SYSTEM data are concatenated into one matrix as in Figure 5.2-8, so that the data to be displayed may be transformed (i.e. multiplied) by the compound transformation.

$$[x\ y\ z\ w]\begin{bmatrix} A \end{bmatrix} = [x'y'z'w']$$

(1) (2)

Figure 5.2-6[1 2 3]

Transformation of a data point by a single transformation showing (1) the transformation notation and (2) the transformed data.

---

[1]In this discussion all data will be represented by the homogeneous coordinate point [x y z w] which may by thought of as a representative data point (two- or three-dimensional) of any data set.

[2]In this discussion, all 4x4 matrices will be represented by the notation:

$$\begin{bmatrix} \text{"NAME"} \end{bmatrix} = \begin{matrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{matrix} \quad (e_{ij} = \text{element}_{ij})$$

where "NAME' identifies the linear transformation represented by the 4x4 matrix. For a detailed discussion of the contents (i.e. each of the 16 elements) of the 4x4 matrices used in THE PICTURE SYSTEM graphics software see Reference 1, Chapter 12.

[3]The transformed data [x'y'z'w'] is usually clipped, viewport mapped and stored into the Refresh Buffer to be displayed on the Picture Display.

$$[x\ y\ z\ w]\underset{(1)}{\begin{bmatrix} & I & \end{bmatrix}} = \underset{(2)}{[x'y'z'w]} = \underset{(3)}{[x\ y\ z\ w]}$$

Figure 5.2-7[1]

Transformation of a data point by the identity matrix I showing (1) the transformation notation, (2) the transformed data, and (3) the equivalence of the transformed data and the original data.

$$\begin{bmatrix} & A & \end{bmatrix}\begin{bmatrix} & B & \end{bmatrix}\begin{bmatrix} & C & \end{bmatrix} = \begin{bmatrix} ABC \end{bmatrix}$$

Figure 5.2-8

Three Simple Transformations and an Equivalent Compound Transformation.

[1]In this discussion, the indentity matrix will be represented by the notation:

$$\begin{bmatrix} & I & \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The associative property of matrices and the use of this property in compound transformations are illustrated in Figure 5.2-9. In this figure, matrix A is post-multiplied by matrix B resulting in the (compound) matrix AB which is then used to transform data points.

$$\left([x\ y\ z\ w] \begin{bmatrix} A \end{bmatrix} \right) \begin{bmatrix} B \end{bmatrix} = [x\ y\ z\ w] \left(\begin{bmatrix} A \end{bmatrix} \begin{bmatrix} B \end{bmatrix} \right) =$$

(1) (2)

$$[x\ y\ z\ w] \begin{bmatrix} AB \end{bmatrix} = [x'y'z'w']$$

(3) (4)

Figure 5.2-9

Transformation of a data point by compound transformations showing (1) the transformation notation, (2) the use of the associative property of matrices, (3) the compound matrix and (4) the transformed data.

Figure 5.2-9 is indicative of the technique used to specify the transformations used in PICTURE SYSTEM application programs. The transformations to be performed upon a given set of data are determined and diagrammed, IN THE ORDER THAT THE DATA IS TO ENCOUNTER THE TRANSFORMATIONS TO BE PERFORMED.[1] A suggested order in which transformations may be performed is:

---

[1]The order in which matrices are multiplied is very important as matrix multiplication, in general, is not commutative; i.e.

$$\begin{bmatrix} A \end{bmatrix} \begin{bmatrix} B \end{bmatrix} \neq \begin{bmatrix} B \end{bmatrix} \begin{bmatrix} A \end{bmatrix}$$

1. Scaling of the data (SCALE)
2. Rotation about the origin of the data (ROT)
3. Translation of the data (TRAN)
4. Windowing of the data and setting the angle and point of view (WINDOW)

Figure 5.2-10 illustrates this order of transformation i the matrix notation previously defined.

$$\begin{bmatrix}\mathrm{SCALE}\end{bmatrix}\begin{bmatrix}\mathrm{ROTx}\end{bmatrix}\begin{bmatrix}\mathrm{ROTy}\end{bmatrix}\begin{bmatrix}\mathrm{ROTz}\end{bmatrix}\begin{bmatrix}\mathrm{TRAN}\end{bmatrix}\begin{bmatrix}\mathrm{WINDOW}\end{bmatrix}\begin{bmatrix}\mathrm{I}\end{bmatrix} = \begin{bmatrix}\mathrm{comp.} \\ \mathrm{tran}\end{bmatrix}$$

Figure 5.2-10

A Suggested Order in which Transformations may be Performed.

It should be noted that inclusion of all of the transformations is not necessary and, in fact, is often undesirable. For example, in displaying two-dimensional data, a rotation about the X or Y axis results in making a three-dimensional picture of two-dimensional data. A suggested order of transformations for two-dimensional data is shown in Figure 5.2-11.

$$\begin{bmatrix}\text{SCALE}\end{bmatrix}\begin{bmatrix}\text{ROTz}\end{bmatrix}\begin{bmatrix}\text{TRAN}\end{bmatrix}\begin{bmatrix}\text{WINDOW}\end{bmatrix}\begin{bmatrix}\ \text{I}\ \end{bmatrix} = \begin{bmatrix}\text{comp.}\\ \text{tran.}\end{bmatrix}$$

Figure 5.2-11

A Suggested Order in which Two-dimensional Transformation may be Performed.

A comparison of Figures 5.2-10 and 5.2-11 shows that the display of two-dimensional data is a special case of the more general three-dimensional case. Therefore, all further discussions of transformations and the examples given will be for the three-dimensional case. Discussions and examples for the two-dimensional case may be formed in a similar manner.

Figure 5.2-12 shows the transformation of a data point by the transformations of Figure 5.2-10.

$$[x\ y\ z\ w]\underbrace{\left(\begin{bmatrix}\text{SCALE}\end{bmatrix}\begin{bmatrix}\text{ROTx}\end{bmatrix}\begin{bmatrix}\text{ROTy}\end{bmatrix}\begin{bmatrix}\text{ROTz}\end{bmatrix}\begin{bmatrix}\text{TRAN}\end{bmatrix}\begin{bmatrix}\text{window}\end{bmatrix}\begin{bmatrix}\ \text{I}\ \end{bmatrix}\right)}_{(1)} =$$

$$[x\ y\ z\ w]\underbrace{\begin{bmatrix}\text{comp.}\\ \text{tran.}\end{bmatrix}}_{(2)} = \underbrace{[x'y'z'w']}_{(3)}$$

Figure 5.2-12

Transformation of a data point showing (1) the use of the associative property of matrices, (2) the compound transformation and (3) the transformed data.

Once the transformations to be performed on a set of data have been diagrammed as in Figure 5.2-12 it is a relatiavely simple task to implement them in a graphics application program. Because the matrix concatenation implemented in hardware pre-multiplies the existing transformation matrix by the new component matrix and retains the result as the new transformation matrix, the order in which the transformation matrix must be created using the system software is: windowing, rotation, translation and scaling. Note that this order is the reverse of the order in which the transformations will be effectively applied to the drawn data. The most recent transformation applies first! This is illustrated in Figure 5.3-13 along with the FORTRAN subroutine calls required to implement this transformation sequence in a user program. Usually, the transformation sequence would be executed repeatedly by changing the parameters in the transformations to produce a dynamic picture. The PUSH and POP operation s facilitate multiple use of compound matrices. For example, the identity matrix might be saved prior to the concantenation of the transformations and restored after all the data had been transformed and before the "next" series of transformations. This technique is shown in Figure 5.2-14. It should be emphasized that the saving (PUSHing) and restoring (POPping) of the Transformation Matrix is performed in hardware, therefore incurring very little overhead. The saving of transformations need not be limited to the identity matrix. Any transformation may be saved for future recall by similarly PUSHing in onto the matrix stack. For example, if the WINDOWing transformation of Figure 5.2-14 were constant, an increase in frame update rate could be achieved by creating the WINDOW transformation only once and saving and restoring that transformation rather than the identiy matrix. This is shown in Figure 5.2-15.

[I]	CALL PSINIT (3,0,,,,)
[WINDOW]	CALL WINDOW (IWL,IWR,IWB,IWT,IH,IY,IE)
[TRAN]	CALL TRAN (TX,ITY,ITZ)
[ROTz]	CALL ROT (IAZ,3)
[ROTy]	CALL ROT (IAY,2)
[ROTx]	CALL ROT (IAX,1)
[SCALE]	CALL SCALE (ISX,ISY,ISZ)
[x y z w]	CALL DRAW3D (IDATA,N,IF1,IF2)
	CALL NUFRAM

Figure 5.2-13

The Order in which Transformations are Concatenated into the Corresponding FORTRAN Subroutine Calls

Save Identity
Matrix Here

$$[x\ y\ z\ w]\begin{bmatrix}\text{SCALE}\end{bmatrix}\begin{bmatrix}\text{ROTx}\end{bmatrix}\begin{bmatrix}\text{ROTy}\end{bmatrix}\begin{bmatrix}\text{ROTz}\end{bmatrix}\begin{bmatrix}\text{TRAN}\end{bmatrix}\begin{bmatrix}\text{WINDOW}\end{bmatrix}\begin{bmatrix}\ I\ \end{bmatrix} =$$

$$[x'y'z'w']$$

```
C     INITIALIZE THE PICTURE SYSTEM
C
            CALL PSINIT(3,0,,,,)
C
C     SAVE THE IDENTITY MATRIX AND BEGIN THE DISPLAY LOOP
C
100         CALL PUSH
C
C     MODIFY OR OBTAIN NEW TRANSFORMATION PARAMETERS
C
            .
            .
            .
C
C     CONCATENATE THE TRANSFORMATIONS
C
            CALL WINDOW(IWL,IWR,IWB,IWT,IH,IY,IE)
            CALL TRAN(ITX,ITY,ITZ)
            CALL ROT(IAY,3)
            CALL ROT(IAX,2)
            CALL ROT(IAX,1)
            CALL SCALE(ISX,ISY,ISZ)
C
C     NOW TRANSFORM THE DATA BY THE COMPOUND TRANSFORMATION
C
            CALL DRAW3D(IDATA,N,IF1,IF2)
C
C     RESTORE THE IDENTITY MATRIX AND DISPLAY TOTE DATA
C
            CALL POP
            CALL NUFRAM
            GO TO 100
```

Figure 5.2-14

Diagrammed Saving of the Identity Matrix and the Corresponding FORTRAN Code.

Save the WINDOW
Transformation here

$$[x\ y\ z\ w]\begin{bmatrix}\text{SCALE}\end{bmatrix}\begin{bmatrix}\text{ROTx}\end{bmatrix}\begin{bmatrix}\text{ROTy}\end{bmatrix}\begin{bmatrix}\text{ROTz}\end{bmatrix}\begin{bmatrix}\text{TRAN}\end{bmatrix}\begin{bmatrix}\text{WINDOW}\end{bmatrix}\begin{bmatrix}\text{I}\end{bmatrix} =$$

$$[x'y'z'w']$$

```
C    INITIALIZE THE PICTURE SYSTEM
C
           CALL PSINIT(3,0,,,,)
C
C    SET THE WINDOWING TRANSFORMATION
C
           CALL WINDOW(IWL,IWR,IWB,IWT,IH,IY,IE)
C
C    SAVE THE WINDOWING TRANSFORMATION AND BEGIN THE DISPLAY LOOP
C
100        CALL PUSH
C
C      MODIFY OR OBTAIN NEW TRANSFORMATION PARAMETERS
C
                .
                .
                .
C
C    CONCATENATE THE TRANSFORMATIONS
C
           CALL TRAN(ITX,ITY,ITZ)
           CALL ROT(IAZ,3)
           CALL ROT(IAY,2)
           CALL ROT(IAX,1)
           CALL SCALE(ISX,ISY,ISZ)
C
C    NOW TRANSFORM THE DATA BY THE COMPOUND TRANSFORMATION
C
           CALL DRAW3D(IDATA,N,IF1,IF2)
C
C    RESTORE THE ORIGINAL WINDOW THAT WAS SAVED
C
           CALL POP
           CALL NUFRAM
           GO TO 100
```

Figure 5.2-15

Diagrammed Saving of the Windowing Transformation
and the Corresponding FORTRAN Code.

The ability to save and restore transformations is a powerful capability which can be used to effectively increase the speed with which data can be transformed and dynamically displayed. An example of this capability is a modification of Figure 5.2-15. If the data array IDATA were to be displayed twice, in the same orientation but SCALEd differently to emphasize different aspects of its geometry, the technique would be used as illustrated in Figure 5.2-16. This ability to nest or stack transformations is available to four levels in hardware, and may be extended by the software to any level required.

The capability of merely call ing a subroutine to perform a given transformation, the speed with which matrices can be concatenated, the ability to stack transformations and the speed with which data can be transformed and displayed make the use of 4x4 matrices and the associated linear transformations a powerful feature of THE PICTURE SYSTEM.

save the compound transformation here

save the WINDOW transformation here

$$[x\ y\ z\ w]\ \begin{bmatrix}\text{SCALE}\end{bmatrix} \text{ or } \begin{bmatrix}\text{SCALE}\end{bmatrix} \begin{bmatrix}\text{ROTx}\end{bmatrix} \begin{bmatrix}\text{ROTy}\end{bmatrix} \begin{bmatrix}\text{ROTz}\end{bmatrix} \begin{bmatrix}\text{TRAN}\end{bmatrix} \begin{bmatrix}\text{WINDOW}\end{bmatrix} \begin{bmatrix}\ \text{I}\ \end{bmatrix} =$$

$$[x'y'z'w']$$

```
C    INITIALIZE THE PICTURE SYSTEM
C
        CALL PSINIT(3,0,,,,)
C
C    SET THE WINDOWING TRANSFORMATION
C
        CALL WINDOW(IWL,IWR,IWB,IWT,IH,IY,IE)
C
C    SAVE THE WINDOWING TRANSFORMATION
C
100     CALL PUSH
C
C    MODIFY OR OBTAIN NEW TRANSFORMATION PARAMETERS
              :
C    CONCATENATE THE TRANSFORMATIONS AND DISPLAY THE DATA TWICE
C
        CALL TRAN(ITX,ITY,ITZ)
        CALL ROT(IAZ,3)
        CALL ROT(IAY,2)
        CALL ROT(IAX,1)
        CALL PUSH
        CALL SCALE(ISX1,ISY1,ISZ1)
        CALL DRAW3D(IDATA,N,IF1,IF2)
        CALL POP
        CALL SCALE(ISX2,ISY2,ISZ2)
        CALL DRAW3D(IDATA,N,IF1,IF2)
C
C    RESTORE THE ORIGINAL WINDOW THAT WAS SAVED
C
        CALL POP
        CALL NUFRAM
        GO TO 100
```

Figure 5.2-16

Diagrammed Nesting of Compound Trasnformations
and the Corresponding FORTRAN Code.

5.3 PROGRAM INITIALIZATION [PSINIT]

Program initialization for PICTURE SYSTEM graphics applications programs consists of:

1. Calling the subroutine PSINIT to initialize THE PICTURE SYSTEM hardware and software.
2. Initiating automatic operations
3. initializing all user variables to their initial state.

Each of these steps in the program initialization process is described and illustrated in the following sections.

5.3.1 Initialization of THE PICTURE SYSTEM Hardware and Software

Typically, the first statement in a user applications program is the call to PSINIT[1] to initialize the hardware and software of THE PICTURE SYSTEM. A typical call to PSINIT is shown in Example 5.3-1.

refresh at 40 frames per second

dynamic frame update

```
CALL PSINIT(3,0,,,,)
```

Example 5.3-1

In this example, typical parameter values have been chosen: a refresh rate of 40 frames per second and the specification of a dynamic frame update rate. One should note that the last four parameters in the subroutine call are specified as null parameters (e.g.,,,,).

---

[1]See Section 4.1 for a detailed specification of PSINIT.

When null parameters are specified, the default values are assumed for these parameters. The following is the PSINIT calling sequence specification of Section 4.1:

```
[EXTERNAL ERRSUB]
CALL PSINIT(IFTIME,INRFSH,[ICLOCK],[ERRSUB],[ISTKCT]
          ,[ISTKAD][,IFMCNT])
```

A discussion of the uses of each of the parameters follows:

IFTIME is used to specify the rate with which the Picture Display is to be refreshed. Typical values for this are 2, 3 or 4 indicating refresh of 60, 40 or 30 frames per second, respectively, appropriate for the P4 phosphor of the Picture Display. If the current frame refresh has not been completed when the refresh interval has elapsed, then the frame refresh will occur upon the next 1/120 second interval after the frame refresh is completed. Table 5.3-1 contains all valid values and the corresponding refresh rates of IFTIME.

INRFSH is used to designate the number of frame refreshes which must be completed before a frame update (NUFRAM) will be recognized. Typically, INRFSH=0 indicates that dynamic frame update is desired. However, certain applications require fixed lengths of time between frame updates. In these applications, this parameter provides this capability. Example 5.3-2 demonstrates the calling sequence which specifies that frame update be done no sooner than every 20th of a second.

```
CALL  PSINIT(3,2,,,,)
```

Example 5.3-2

TABLE 5.3-1

IFTIME	REFRESH RATE
1	120.0 frames per second
2	60.0 frames per second
3	40.0 frames per second
4	30.0 frames per second
5	24.0 frames per second
6	20.0 frames per second
7	17.1 frames per second
8	15.0 frames per second
9	13.3 frames per second
10	12.0 frames per second
11	10.9 frames per second
12	10.0 frames per second
13	9.2 frames per second
14	8.6 frames per second
15	8.0 frames per second
16	7.5 frames per second

The update rate in seconds may be computed from the parameters of the PSINIT call in example 5.3-2 as follows:

$$\text{update rate} = \frac{\text{IFTIME}}{120} * \text{INRFSH} = \frac{3}{120} * 2 = \frac{1}{20} \text{ second}$$

If a longer interval of time is required to generate a new frame, the update rate will automatically extend in order for the system to complete the new frame. A new frame will not be displayed more often than specified by INFRSH but may take longer depending on the time required to compute the new frame. Parameter IFMCNT may be used to determine the number of 1/120 second increments required to create a new frame.

ICLOCK is used to allow user synchronization with the refresh of the display. When specified, this parameter is incremented with each frame refresh and is typically used to display an item for a fixed length of time (or number of refreshes). For example, if a user error message is to be displayed for 10 seconds it would be programmed as shown in Example 5.3-3. It should be noted that in the example, the number 400 used to terminate the display loop was derived from a refresh rate of 40 frames/second (IFTIME=3) * 10 seconds = 400 frames.

```
      CALL PSINIT(3,0,ICLOCK,,,,)
            .
            .
            .
C   BEGIN ERROR DISPLAY LOOP
C
C   CALL USER SUBROUTINE TO DISPLAY THE ERROR MESSAGE
C
            CALL ERMSG
            CALL NUFRAM
C
C   RESET CLOCKING VALUE
C
        ICLOCK=0
C
C   DONE?
C
100     IF(ICLOCK.NE.400)   GO TO 100
C
C   FINISHED... CONTINUE WITH USER PROGRAM
C
            .
            .
            .
```

Example 5.3-3

ERRSUB is a user error subroutine, which if specified may determine where the user error occurred. If no user error subroutine is provided, the graphics error subroutine PSERRS[1] is called to report the occurrence of the user error. Typically, the system error subroutine is specified by default as shown in Examples 5.3-2 and 5.3-3. The user requiring more memory may consider a user error subroutine which is shorter in length than PSERRS. It is suggested in this case however, that the user error subroutine be named PSERRS or that a global symbol PSERRS be declared to avoid loading the system error subroutine PSERRS. Example 5.3-4 demonstrates the use of a user error subroutine which avoids the loading of the system error subroutine, PSERRS.

```
EXTERNAL PSERRS
CALL  PSINIT(3,0,,PSERRS,,)
  .
  .
  .
END

SUBROUTINE  PSERRS
STOP
RETURN
END
```

Example 5.3-4

ISTKCT is used to specify the number of 16 word contiguous arrays allocated as matrix stack area. This is required only if the stacking (PUSHes) of transformations exceeds 4, the number implemented within the Picture Processor. If this is required, ISTKCT is the number of additional levels of matrix stack space that are required.

----

[1] For Paper Tape software users a halt will occur rather than an e ror message being printed out. See Appendix D for specific Paper Tape details.

ISTKAD is an integer arrray allocated as matrix stack area. This contiguous area need be 16*ISTKCT words in length. If ISTKCT contains the value 0 or is not specified, then this argument will not be utilized. Example 5.3-5 illustrates the use of this feature.

IFMCNT is an optional parameter which, if specified, will allow a user to determine the frame update rate at which his picture is being created. This parameter is intended for information purposes only. For example, if a frame update rate of 15 frames per second is required, this parameter may be monitored to determine if frame update is proceeding at this rate. IFMCNT should be initialized (or zeroed) by the user each time the frame update rate is to be determined. IFMCNT is never initialized by the system software, but rather is always incremented upon each refresh interval by the number of 1/120 seconds that have elapsed since the last frame refresh. Table 5.3-2 shows the values of IFMCNT for frame update rates down to 10 frames per second. Example 5.3-6 illustrates the use of IFMCNT.

```
      DIMENSION  ISTKAD(16,1)
      CALL PSINIT(3,0,,,1,ISTKAD)
C
C   LEVEL 1
C
      CALL PUSH
        :
C
      CALL PUSH
        :
C
      CALL PUSH
        :
C
      CALL PUSH
        :
C
      CALL POP
        :
C
      CALL PUSH
        :
C
      CALL PUSH
        :
C
      CALL POP
        :
C
      CALL POP
        :
C
      CALL POP
        :
C
      CALL POP
        :
C
      CALL POP
        :
```

Example 5.3-5

TABLE 5.3-2

IFMCNT	UPDATE RATE
1	120.0 frames per second
2	60.0 frames per second
3	40.0 frames per second
4	30.0 frames per second
5	24.0 frames per second
6	20.0 frames per second
7	17.1 frames per second
8	15.0 frames per second
9	13.3 frames per second
10	12.0 frames per second
11	10.9 frames per second
12	10.0 frames per second

```
              CALL PSINIT(3,0,,,,,IFMCNT)
                .
                .
                .
C
C   BEGIN DISPLAY LOOP
C
100           IFMCNT=0
                .
                .
                .
              CALL NUFRAM
C
C   ENSURE UPDATE RATE OF AT LEAST 15 FRAMES PER SECOND
C
              IF(IFMCNT.LE.8)  GO TO 100
C
C   SLOW FRAME UPDATE, PRINT A MESSAGE AND THEN CONTINUE
C
              X=120./IFMCNT
              PRINT 2000, X
2000          FORMAT('FRAME UPDATE RATE=',F5.2,'FRAMES PER SECOND')
              GO TO 100
```

Example 5.3-6

5.3.2 Initiating Automatic Operations [TABLET, CURSOR]

The Graphics Software Package provides the facility to have certain operations occur automatically. These operations are:

1. The updating of the tablet position and status of the pen.
2. The display of a cursor within an initial viewport.

These automatic operations can be used independently or together to provide dynamic pointing capabilities without programming effort.

The automatic operations occur at the rate specified as the refresh rate in the call to PSINIT. After each frame refresh has been initiated the automatic operations that have been "turned on" are performed.

5.3.2.1 Automatic Tablet Update

The automatic tablet update is initiated by a call to the TABLET subroutine specifying that the tablet is to be used in automatic mode as shown in Example 5.3-7.

```
CALL   TABLET(1,IX,IY,IPEN)
```

Example 5.3-7

In this example, the parameters IX,IY and IPEN are variables which are to be automatically updated with the x-pen position (IX), the y-pen position (IY) and the pen status information (IPEN). This information may be used to determine menu selections and in conjunction with the CURSOR subroutine to display the current pen position.

5.3.2.2 Automatic Cursor Display

Automatic cursor display is initiated by a call to the CURSOR subroutine specifying that the cursor is to be displayed in automatic mode as shown in Example 5.3-8.

```
CALL  CURSOR(IX,IY,1)
```

Example 5.3-8

In this example, the parameters IX and IY are the variables which contain the x,y position at which the cursor is to be displayed. These parameters are usually the values which indicate the x,y position of the pen, but need not be the tablet values and may indicate any information.

5.3.2.3 Use of Automatic Tablet and Cursor Modes

Initiation of automatic tablet and cursor modes should proceed in the following order:

1. CALL PSINIT to initialize THE PICTURE SYSTEM.
2. CALL VWPORT to specify the viewport boundaries[1] within which the cursor is to appear, if other than the default boundaries are required.
3. CALL TABLET to initiate automatic tablet update.
4. CALL CURSOR to initiate automatic cursor display.

Examples 5.3-9 and 5.3-10 show the use of automatic tablet and cursor modes.

---

[1]The boundary variables used in the VWPORT call may be modified thereafter and the cursor will continue to appear within the dynamically changing viewport.

```
C
C   INITIALIZE THE PICTURE SYSTEM
C
          CALL  PSINIT(3,0,,,,)
C
C   USE DEFAULT VIEWPORT OF ENTIRE SCREEN INITIALIZED BY
C   PSINIT FOR CURSOR DISPLAY
C
          CALL  TABLET(1,IX,IY,IPEN)
          CALL  CURSOR(IX,IY,1)
C
C   BEGIN DISPLAY LOOP
C
          .
          .
          .
```

Example 5.3-9

```
          DATA  IVL,IVR,IVB,IVT/-2047,0,0,2047/
C
C   INITIALIZE THE PICTURE SYSTEM
C
          CALL  PSINIT(3,0,,,,)
C
C   SET UP THE VIEWPORT FOR CURSOR DISPLAY
C
          CALL  VWPORT(IVL,IVR,IVB,IVT,255,255)
          CALL  TABLET(1,IX,IY,IPEN)
          CALL  CURSOR(IX,IY,1)
C
C   BEGIN DISPLAY LOOP
C
          .
          .
          .
```

Example 5.3-10

5.3.3 Initialization of User Variables

Variables are usually used in an applications program to retain values which are passed to the graphics subroutines to indicate angles of rotation, translation values, etc. Upon initial loading of the program, these variables will contain their initial values. However, if the program is re-started or has a programmed re-start facility, these variables will contain values which may not be the initial values required. For this reason, it is suggested that all user variables be initialized before the display loop is begun. Figure 5.3-1 illustrates a suggested placement of the user variable initialization process.

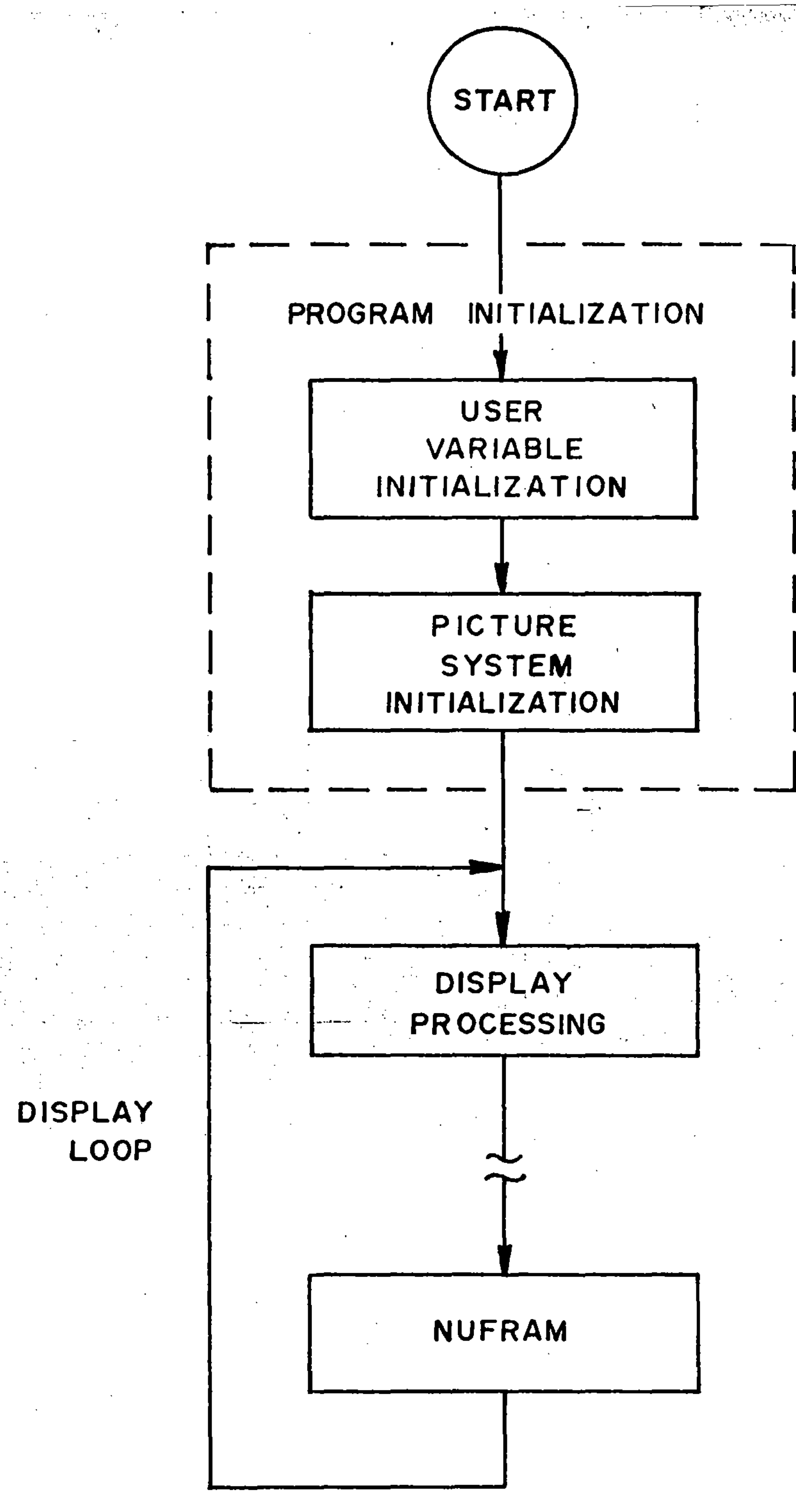


Figure 5.3-1

Suggested Program Initialization Structure

5.4 VIEWPORTS [VWPORT]

A viewport is a program-specified rectangular region of an output device within which the windowed data is mapped for display. Typically, the output device is the Picture Display of THE PICTURE SYSTEM.

Figures 5.4-1a and b illustrate the two- and three-dimensional display of data which is mapped into the viewport. A viewport is specified for THE PICTURE SYSTEM by calling the VWPORT subroutine. The following is the VWPORT calling sequence specification of Section 4.1:

CALL VWPORT (IVL,IVR,IVB,IVT,IHI,IYI).

The parameters passed to the subroutine specify the boundaries of the viewport in the coordinate system of the output device; viewport left boundary (IVL), viewport right boundary (IVR), viewport bottom boundary (IVB) and viewport top boundary (IVT). The subroutine also provides the ability to specify the intensity at which data will be displayed at the hither and yon clipping planes; hither intensity (IHI) and yon intensity (IYI).

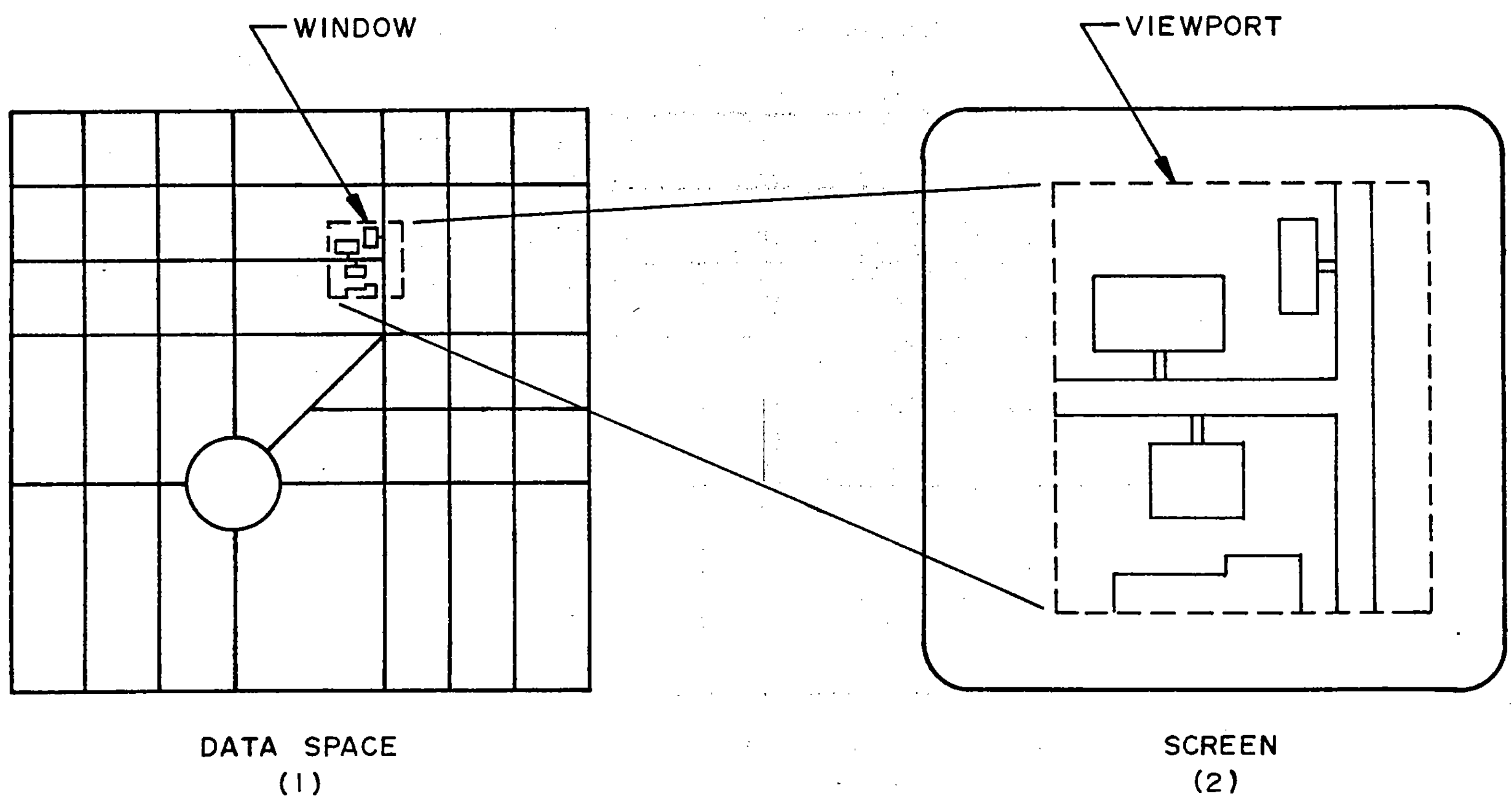
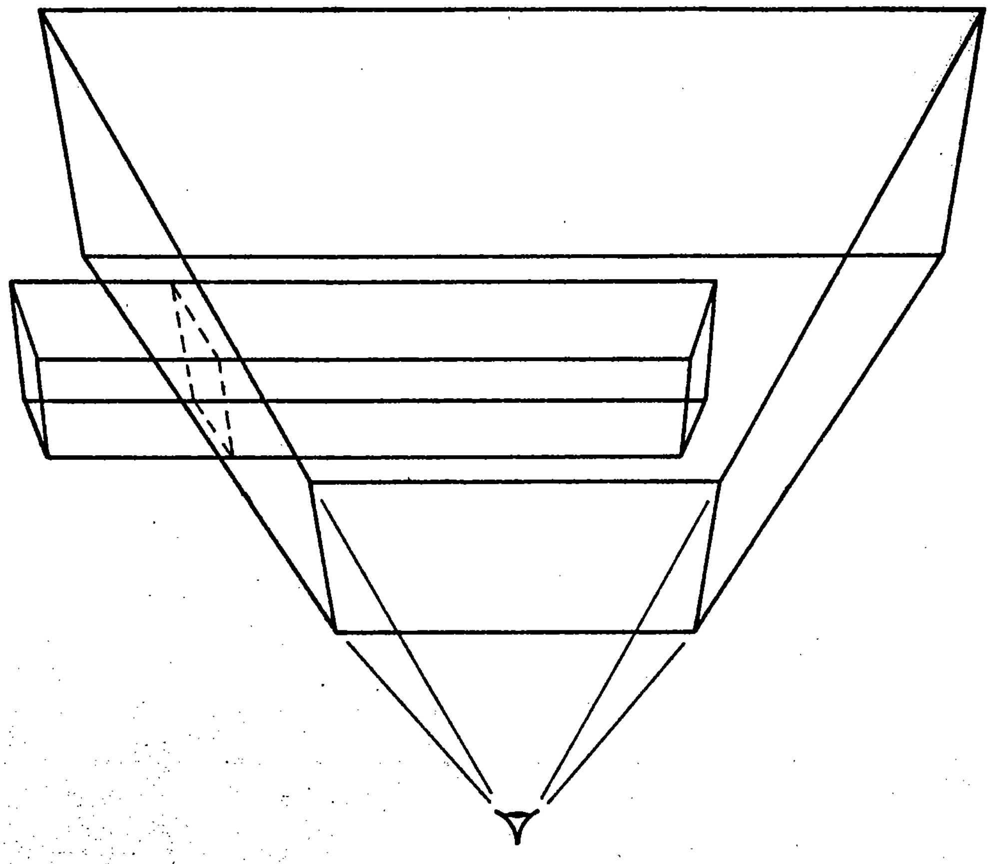
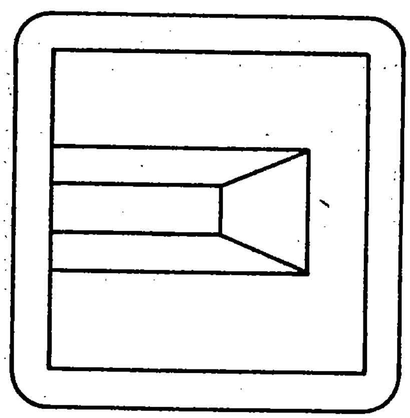


Figure 5.4-1a

Two-dimensional clipping and viewport mapping showing (1) the two-dimensional window and data and (2) the picture as it would appear on the Picture Display.



(1)



(2)

Figure 5.4-1b

Three-Dimensional clipping and viewport mapping showing (1) the three-dimensional perspective window and data and (2) the picture as it would appear on the Picture Display.

5.4.1 Full Screen Viewport

The entire Picture Display may be selected as a viewport by specifying the maximum coordinate range for the viewport boundaries as shown in Example 5.4-1.

```
CALL VWPORT (-2048,2047,-2048,2047,255,255)
```

Example 5.4-1

A call with these parameters specified would result in all data subsequently drawn being displayed within a viewport the size of the entire Picture Display. Viewports may be specified to be non-square, but this causes distortion of the data to be displayed as illustrated in Figure 5.4-2. This distortion, caused by the linear mapping of the data space into viewport coordinates, may be compensated by an appropriate windowing transformation as described in Section 5.5.

5.4.2 Multiple Viewports

The Picture Display may be used for simultaneous display of different pictures. For example, the screen could be used to show the entire street map of Figure 5.4-1a and the magnified portion of the map simultaneously as illustrated in Figure 5.4-3. The statements used to accomplish this are shown in Example 5.4-2. The use of multiple viewports on one display is a powerful feature of THE PICTURE SYSTEM. The ability of THE PICTURE SYSTEM to display data on up to four displays allows programs written using multiple viewports on one display to be upgraded at a later date using several displays to produce pictures of full screen size.

```
      INTEGER IP1(2),IP2(2)
      DATA IP1/1000,16384/
      DATA IP2/8192,0/
C
C   INITIALIZE THE PICTURE SYSTEM
C
      CALL PSINIT(3,0,,,,)
C
C   SAVE THE INITIAL TRANSFORMATION
C
100   CALL PUSH
C
C   SET THE VIEWPORT FOR THE TEXT AND ANOTATE THE DISPLAY
C
      CALL VWPORT(-2048,2047,-2040,2047,255,255)
      CALL DRAW2D(IP1,1,2,2,0)
      CALL TEXT(17,'ENTIRE STREET MAP')
      CALL DRAW2D(IP2,1,2,2,0)
      CALL TEXT(13,'MAGNIFICATION')
C
C   DISPLAY THE ENTIRE MAP
C
      CALL VWPORT (-2048,0,0,2047,255,255)
      CALL WINDOW (-32767,32767,-32767,32767)
      CALL MAP
      CALL POP
      CALL PUSH
C
C   DISPLAY THE MAGNIFIED PORTION OF THE MAP
C
C   NOTE THE NON-SQUARE VIEWPORT AND COMPENSATING
C   NON-SQUARE WINDOW
C
      CALL VWPORT(-2048,2047,-2048,0,255,255)
      CALL WINDOW(2000,10192,12288,16384)
      CALL MAP
       .
       .
       .
      CALL POP
      CALL NUFRAM
      GO TO 100
C
      END
```

Example 5.4-2

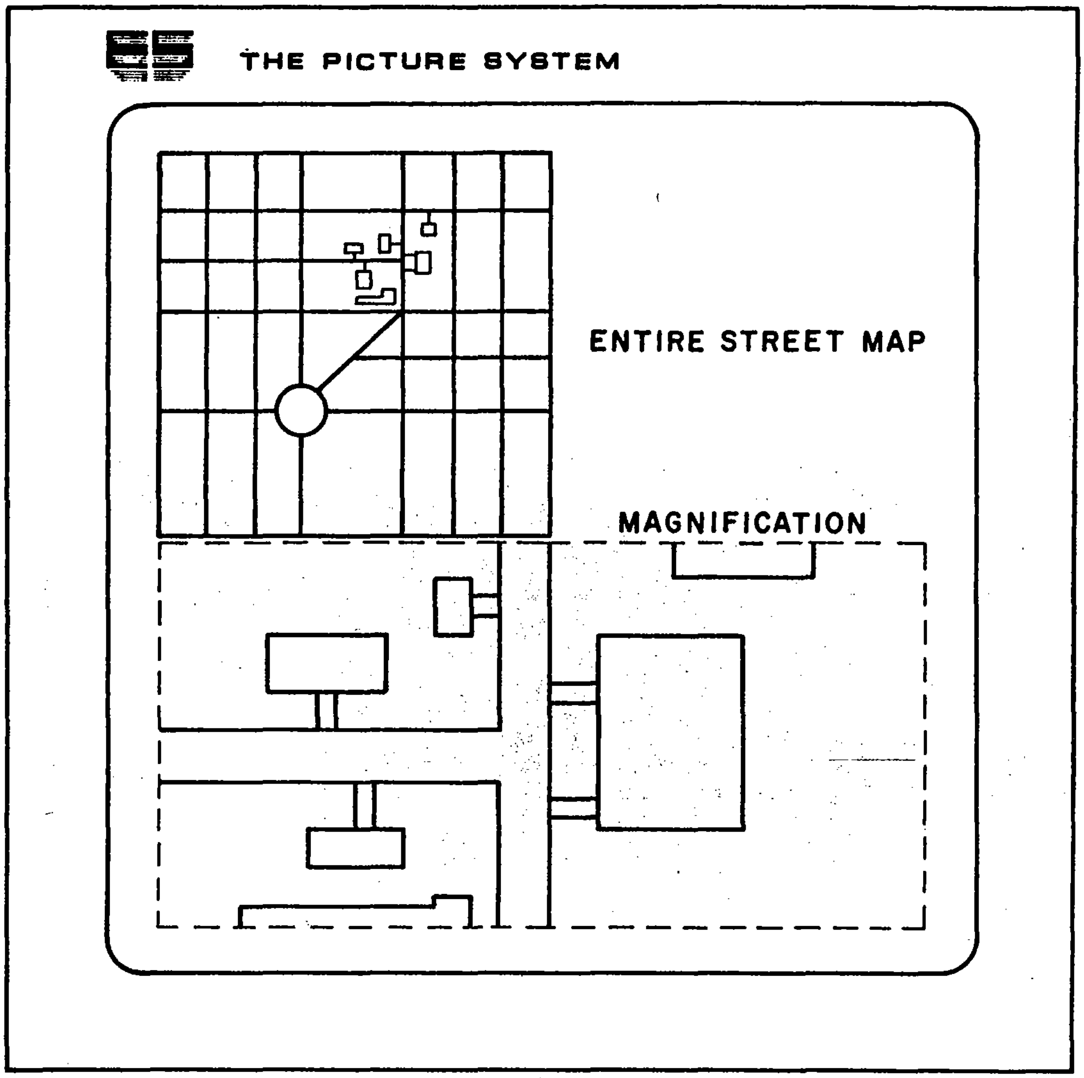


Figure 5.4-3

Simultaneous Use of the Screen to Display an Entire Street Map, a Portion of the Map Magnified and Text Anotation using Three Viewports to Specify the Portion of the Screen to be Used.

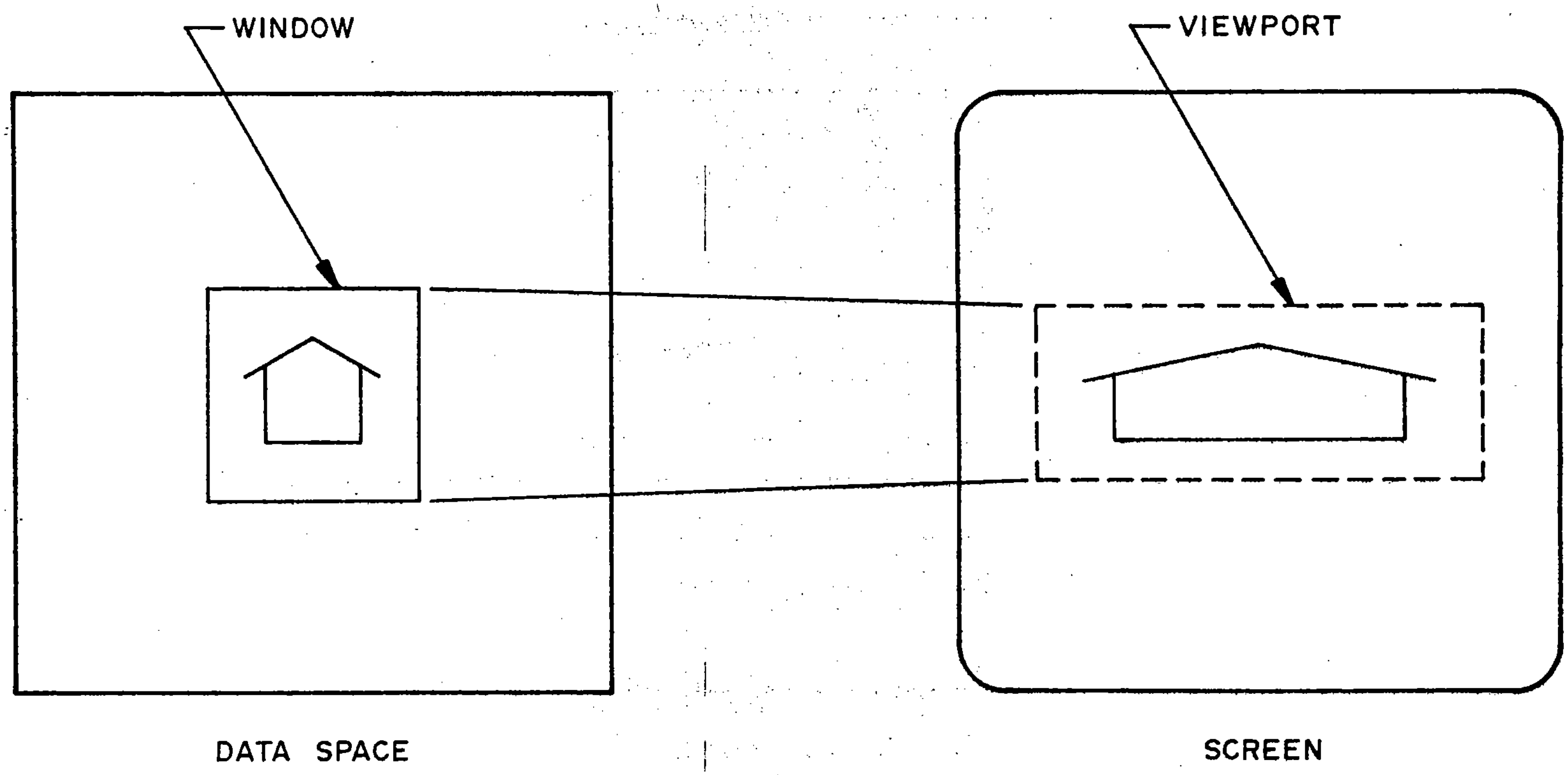


Figure 5.4-2

A non-square viewport which illustrates the data distortion which can occur if a corresponding non-square window is not specified.

5.4.3 Depth-cueing

A heightened sense of perspective may be imparted to three-dimensional objects by specifying that depth cueing be performed. This feature provides the ability to vary the intensity of lines as the lines become "further away" by specifying differing hither and yon intensities when calling the VWPORT subroutine. The maximum depth-cueing effect is obtained by specifying a maximum hither intensity (255) and a minimum yon intensity (0). Example 5.4-3 shows the use of the VWPORT subroutine to specify depth-cueing.

```
CALL VWPORT (-2048,2047,-2048,2047,255,0)
```

Example 5.4-3

NOTE: The specification of viewport boundaries larger than the capability of the output device will cause lines to wrap-around the device. The maximum viewport boundaries for the Picture Display are:

IVL, IVR, IVB, IVT: -2048 to +2047

IHI, IYI: 255 to 0.

5.5 WINDOWING [WINDOW]

A window is a two- or three-dimensional framework or enclosure in the data space. All lines which fall within the window boundaries appear on the Picture Display while those portions of the lines falling outside the boundaries are not displayed. Should a line extend from within this region to someplace outside it, only that portion of the line falling inside the boundaries will be displayed (lines are clipped to the window boundaries). This process of windowing includes the definition of both an enclosure and a point-of-view, that is, the position of the observer as shown in Figure 5.5-1a, b and c. This is in contrast to the positioning (i.e. rotating, translating, etc.) and displaying of the data (i.e. line and text output) which may be considered to set a scene to be viewed.

The following are the WINDOW calling sequence specifications of Section 4.1:

```
CALL WINDOW (IWL,IWR,IWB,IWT[,IW])

CALL WINDOW (IWL,IWR,IWB,IWT,IWH,IWY[,IE[,IW]])
```

These calling sequences allow the user to view a scene from any number of different positions, and in several different ways. For example, a scene may be viewed in perspective as an orthographic projection or even as a two-dimensional picture with no implication of apparent depth. The following sections describe how the WINDOW subroutine may be used to view a scene in these different ways.

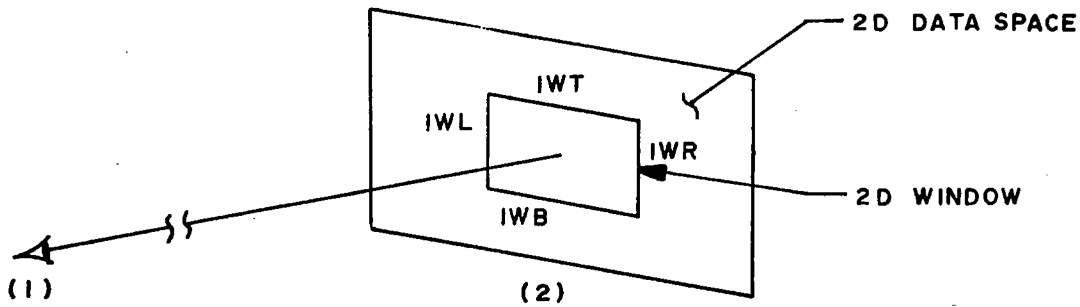


Figure 5.5-1a

Two-dimensional windowing showing (1) the eye whose x,y position is at the center of the window and whose position is at negative infinity and (2) the window whose x,y position is determined by the left, right, bottom, top parameters (IWL,IWR,IWB,IWT) and whose z position is at 0.

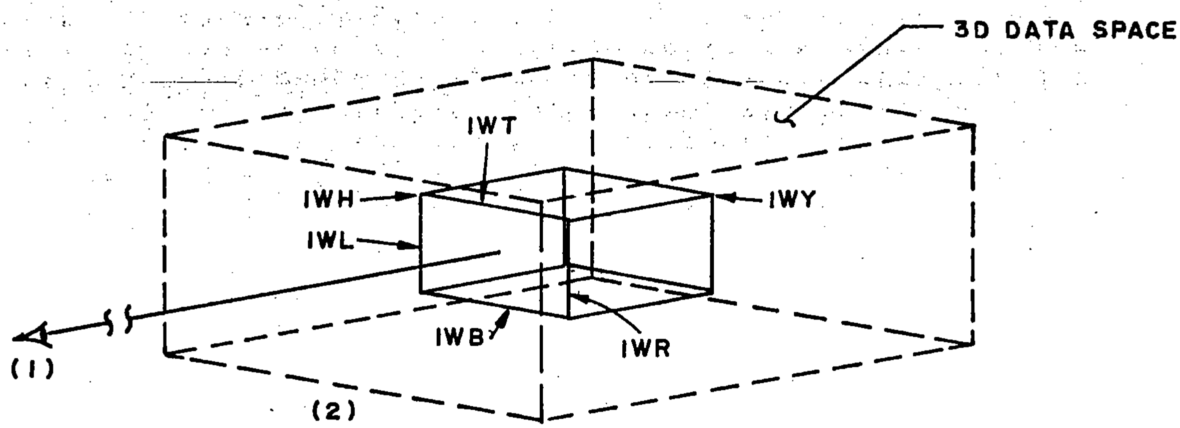


Figure 5.5-1b

Three-dimensional orthographic windowing showing (1) the eye whose x, y position is at the center of the window and whose z position is at negative infinity and (2) the 3D orthographic window whose x,y position is determined by the left, right, bottom, top parameters (IWL,IWR,IWB,IWT) and whose z position is determined by the hither and yon parameters (IH,IY).

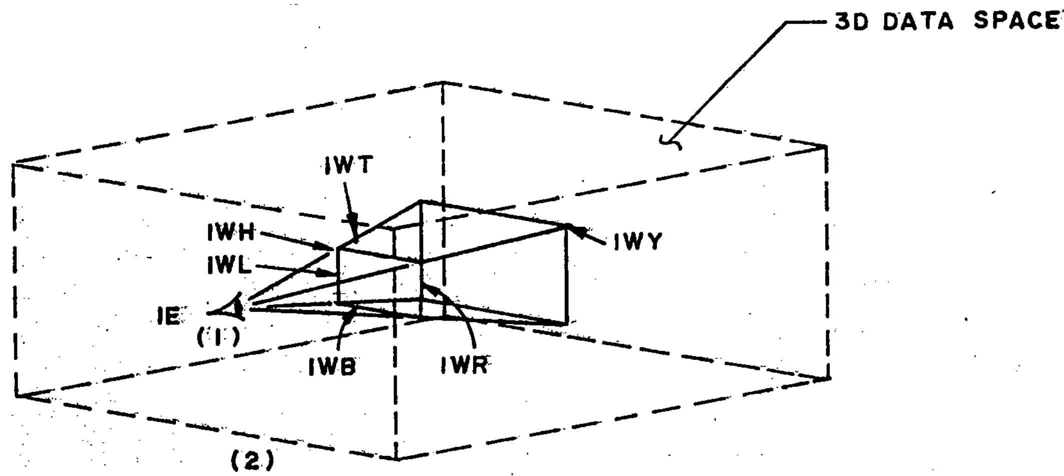


Figure 5.5-1c

Three-dimensional perspective windowing showing (1) the eye whose x,y position is at the center of the window and whose z position is determined by the eye position parameter (IE) and (2) the 3D perspective window whose position at the hither clipping plane (IH) is determined by the left,right,bottom,top parameters (IWL,IWR,IWB,IWT) and whose yon position is determined by the yon parameter (IWY).

5.5.1 Two-Dimensional Views

A two-dimensional view is established by allowing the X-Y boundaries of the window to be specified, thus permitting the side of the WINDOW facing the viewer to be shaped into any sort of rectangle and placed at the viewer's convenience anywhere on the X-Y plane. This is done by a two-dimensional (four-or five-parameter) call to the WINDOW subroutine, specifying the parameters IWL, IWR, IWB , IWT, and the optional scaling parameter IW. The first four define, respectively, the left, right, bottom and top boundaries of the WINDOW. The hither and yon boundaries remain fixed; that is, the hither boundary is set at zero, while the yon is set equal to the homogeneous coordinate IW if specified and 32767 otherwise.

This transformation is generally used in connection with the display of two-dimensional data, since the z-coordinate has no effect on the placement of the lines in the picture, except to control their intensity.

Example 5.5-1 shows a call to the WINDOW subroutine to set a two-dimensional windowing transformation.

```
           .
           .
           .
C
C   SET THE 2D WINDOWING TRANSFORMATION
C
        CALL WINDOW(-20000,-5000,-20000,-5000)
           .
           .
           .
```

Example 5.5-1

5.5.2 Three-Dimensional Orthographic Views

An extension of the two-dimensional WINDOW call to six parameters provides for the definition of a rectangular parallelepiped (i.e. box-shaped) enclosure for the WINDOW, the six boundaries of which are directly specifiable, thus allowing the user visual access to any portion of the data definition space. This is done by adding parameters which specify the position of the hither and yon boundaries (IWH,IWY) to the WINDOW left, right, bottom and top parameters. When called in this manner, a WINDOW is defined which is then viewed as if from an infinite distance away. The pictures which result are analogous to photographs of objects taken at great distances through a telescopic lens of extremely high magnification; the picture may appear clear and sharp, but evidence of perspective is lost. By setting the eye position at negative infinity, this same effect is obtained, wherein only the x and y coordinates of the displayed lines and dots affect the picture, with the z coordinate having no effect except perhaps in the intensity of the data displayed. This type of view, known as orthographic projection, is specified by a call to the WINDOW subroutine as illustrated by Example 5.5-2.

```
          .
          .
          .
C
C    SET THE ORTHOGRAPHIC WINDOWING TRANSFORMATION
C
           CALL WINDOW(-2000,-1000,-2000,-1000,-5000,-5000)
          .
          .
          .
```

Example 5.5-2

If the scaling parameter, IW, is required, the IE and IW parameters must both be specified to distinguish this calling sequence from the standard perspective view specification. The IE parameter should then be specified as equal to the IWH value, the convention chosen to specify that the eye be positioned at negative infinity as shown in Example 5.5-3.

```
C
C   SET SCALED ORTHOGRAPHIC WINDOWING TRANSFORMATION
C
C   THIS IS EQUIVALENT TO:
C
C   CALL WINDOW(-40000,-20000,-40000,-20000,-10000,10000)
C
C   (NOTE:  IE=IWH)
C
         CALL WINDOW (-20000,-10000,-10000,-10000,-5000,
        1 5000,-5000,16384)
```

Example 5.5-3

5.5.3 Perspective Views

When three-dimensional objects are viewed, the viewer infers depth from the fact that distant objects appear smaller and that parallel lines extending away from the viewer appear to come together in the distance. This effect may be invoked for three-dimensional data (and even for two-dimensional data where the z-coordinate is specified as a constant) by calling the WINDOW subroutine with the IE parameter not equal to IWH. The effect of this subroutine call is to modify the shape and position of the six-sided, three-dimensional orthographic window so as to produce a "frustrum of vision"; that is, a right rectangular pyramid, with the top sliced off by a cut parallel to the base. If the eye is placed at the position previously occupied by the apex of the pyramid, then the edges of the rectangular cut will define the hither boundaries of the four side walls of the frustrum. Anything lying within the frustrum will appear to be framed in the rectangle, and will thus be viewed when displayed on the Picture Display.

Seven parameters are supplied in a perspective call to the WINDOW subroutine. These parameters completely specify the shape and position of the enclosure, with the one restriction that the direction of view be always along a line parallel to the Z axis. The effect of rotational changes to the direction of view must be explicitly accomplished by calls to the ROT subroutine to perform opposite rotations to the coordinate data. The position and size of the rectangular side of the frustrum closest to the eye (known as the hither clipping plane) is uniquely determined by the five parameters IWL,IR,IWB,IWT and IWH. These specify its left, right, bottom and top boundaries, as well as the Z-position of the plane of the rectangle. The position of the back plane of the frustrum (called the yon clipping plane) is specified by the parameter IWY, while the Z-position of the eye (centered in front of the hither plane) is specified by IE, as shown in Figure 5.5-2. An optional eighth parameter, IW, may also be supplied when one or more of the other parameters is too large to be expressed directly. These parameters not only specify the shape and position of the window enclosure, but also implicitly define the angle of view (θ) as follows:[1]

$$\mathrm{Tan}\ \frac{\theta}{2} = \frac{IWR-IWL}{2(IWH-IE)}$$

The angle of view may be varied by adjusting the WINDOW parameters to provide an effect similar to a telephoto camera lens (viewing angle < 20°) or a fish eye camera lens (viewing angle > 40°).

---

[1]This defines the X-viewing angle only. The Y viewing angle is inferred automatically by the aspect ratio as described in the next section.

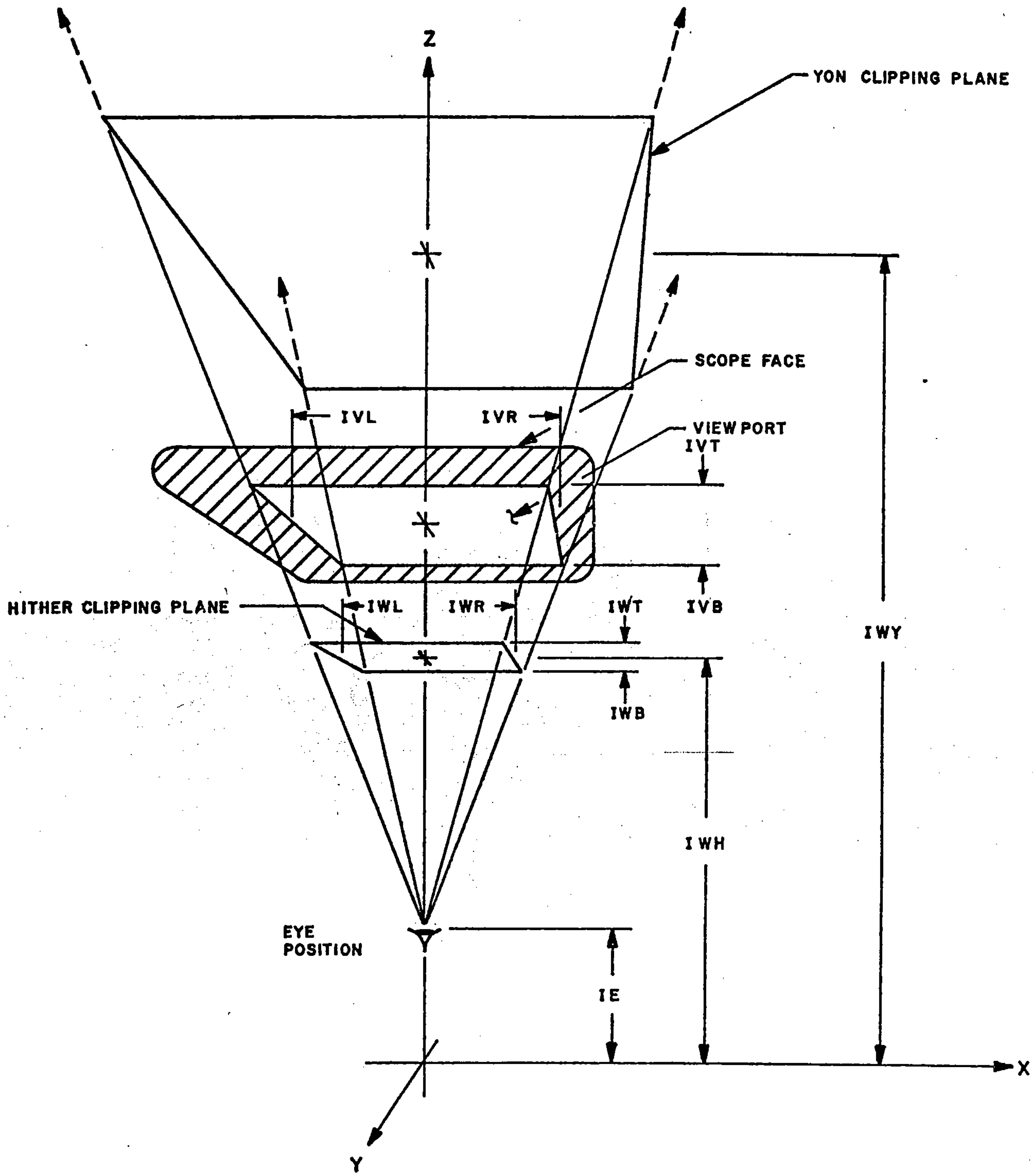


Figure 5.5-2

The Frustram of Vision as defined by the WINDOW subroutine

5.5.4 Non-Square Windows and Viewports

In specifying a WINDOW (square or non-square) the user should display the data within a viewport of a corresponding shape to ensure that the data is displayed without distortion. This may be stated more explicitly by defining the term "aspect ratio". The "aspect ratio" of the window is simply the ratio of the horizontal width of the window to its vertical height, or:

$$\text{Window Aspect Ratio} = \frac{IWR-IWL}{IWT-IWB}$$

In order for data to be displayed without distortion, the aspect ratio of the window must be equal to the aspect ratio of the viewport. This may be expressed in terms of the parameters as:

$$\frac{IWR-IWL}{IWT-IWB} = \frac{IVR-IVL}{IVT-IVB}$$

The user should maintain this equality for all types of windowing; two-dimensional, three-dimensional orthographic and three-dimensional perspective views. The user who desires to view a three-dimensional picture in proper perspective has the additional constraint that the angular width of the frustrum of vision be approximately equal to the angle through which the viewport is observed by the user. This means that the user should specify the WINDOW parameters such that the frustrum of vision assumes a shape which is proportional to that which exists when the user actually views the Picture Display as shown in Figure 5.5-3. In this figure the user is shown viewing the Picture Display from a distance of approximately 20 inches with a viewport width specified as the entire Picture Display (10 inches). From this it may be seen that the user should specify a window which has a responding ratio:

$$\frac{IWR-IWL}{IWH-IE} = \frac{\text{actual viewport width}}{\text{distance of viewer from Picture Display}}$$

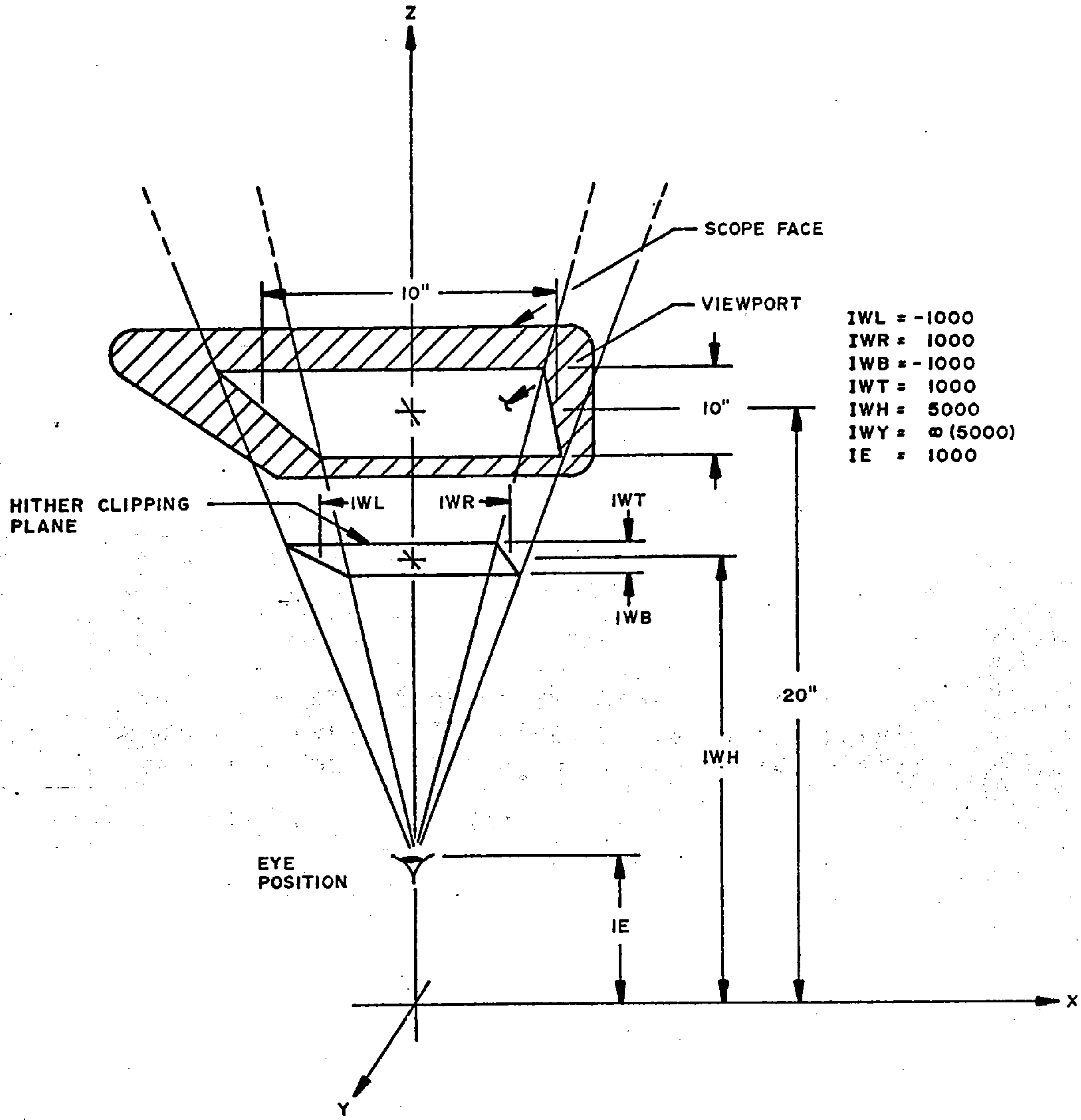


Figure 5.5-3

Window Specification which creates a "proper" perspective for the actual position of the viewer

$$\frac{IWR-IWL}{IWH-IE} = \frac{IWT-IWB}{IWH-IE} = \frac{1}{2} = \frac{10"}{20"}$$

The statements used to specify such a window are shown in Example 5.5-4.

```
C
C   ASSUME A VIEWPORT OF 10 INCHES IN WIDTH AND
C   HEIGHT, WITH THE USER EYE POSITION AT APPROXIMATELY
C   20 INCHES FROM THE DISPLAY.  THIS PRODUCES A VIEWING
C   ANGLE OF ABOUT 28 DEGREES, AN ANGLE COMPARABLE TO
C   THAT OF A CAMERA.  SPECIFY THE WINDOW SO THAT:
C
C       IWR-IWL    IWT-IWB    1
C       ------- =  ------ =   -
C       IWH-IE     IWH-IE     2
C
        CALL VWPORT (-2048,2047,-2048,2047,255,0)
        CALL WINDOW (-1000,1000,-1000,1000,5000,5000,1000)
C
C   NOW PERFORM THE TRANSFORMATIONS
C
        CALL PUSH
        CALL TRAN (ITX,ITY,ITZ)
          .
          .
          .
```

Example 5.5-4

5.5.5 Sectioning

For most applications it is desirable not to have a rear limit to the enclosure (i.e. it is desirable to have the yon clipping plane at infinity). Since infinity is not an expressable value, a convention has been adopted which entails setting IWY equal to IWH, as in Example 5.5-4, to achieve the effect of a yon clipping plane at infinity.

However, in some applications it is desirable to present the data to the viewer in a thin slice seen face-on. This is known as sectioning and is achieved simply by setting IWY to a value slightly beyond IWH. The section thickness may be gradually increased or decreased by advancing IWY steadily away from or toward IWH from frame to frame. It should be noted, however, that when IWY actually reaches IWH, the condition mentioned above will have occurred and because IWH = IWY, the yon clipping plane will be at infinity. This visually annoying situation may be easily avoided by choosing an increment for IWY which is not an even divisor of the section width (e.g. IWY-IWH=250, but IWY=-20). Then the section width, as it decreases, will pass in steps through zero without actually landing on it, and the above difficulty is thus avoided.

5.5.6 Depth-cueing

To complete the illusion of perspective, the intensity of the lines drawn may be diminished with distance from the eye. This feature is known as depth-cueing. This may be accomplished by setting the viewport hither and yon intensities at high and low values, respectively. The maximum depth-cue values are shown in the viewport specification of Example 5.5-5, as full intensity for IHI and no intensity (or black) for IYI.

```
C
C   SPECIFY MAXIMUM DEPTH-CUEING
C
          CALL VWPORT (-2000,2000,-2000,2000,255,0)
```

Example 5.5-5

For some viewers, however, these values tend to be a little harsh and a small but non-zero value for IY, permitting objects at apparently great distances to remain slightly visable, may be used.

Both sectioning an d depth-cueing are permissible in orthographic as well as perspective views. However, when using sectioning and depth-cueing together in an orthographic view, it should be noted that line intensity decreases linearly through the section; whereas, in a perspective view intensity is adjusted such that the total light emitted by a given line varies with apparent distance according to the inverse-square law of optics. This PICTURE SYSTEM feature allows data to be displayed as it would appear when illuminated by a light source; thereby allowing data to decrease rapidly in intensity with increase in apparent distance.

5.5.7 Rear-facing Views

For the sake of simplicity, all perspective and orthographic views produced by the WINDOW subroutine are oriented so that the viewer looks in the direction of positive Z-values. To alter this view, the user merely has to provide the appropriate rotation and translation transformations by making appropriate calls to the ROT and TRAN subroutines. Assuming that north lies along the Z-axis with the Y-axis pointing up , a perspective view of the world looking northeast from a point 100 units east along the X-axis is generated by the statements shown in Example 5.5-6.

However, due to the fact that values of IE may be specified which are greater than corresponding values of IWH, perspective views may be produced, without the aid of transformations, which look "south". In these views, the parameters IWL and IWR are automatically interchanged, so that the view appears as though the viewer had actually turned around and looked "south", rather than having obtained a "southern" view by looking "northward" through a mirror, as shown in Example 5.5-7. Thus, the effect obtained is exactly the same as if a northern view had been rotated 180 degrees by a call to the ROT subroutine shown in Example 5.5-8.

```
     .
     .
     .
C
C    VIEW LOOKING NORTHEAST FROM A POINT 100 UNITS
C    EAST OF THE ORIGIN.  HITHER CLIPPING PLANE IS
C    400 UNITS AWAY, YON PLANE IS 5000 UNITS AWAY.
C    (THE VIEWPORT IS MODIFIED BY ROTATING AND
C    TRANSLATING THE DATA IN THE OPPOSITE DIRECTION.)
C
      CALL WINDOW(-100,100,-100,100,400,5000,0)
      IROT45=-8192
      IYAXIS=2
      CALL ROT(IROT45,IYAXIS)
      CALL TRAN(-100,0,0)
       .
       .
       .
```

Example 5.5-6

```
     .
     .
     .
C
C    VIEW WITH SOUTHERN EXPOSURE, WITHOUT TRANSFORMATIONS
C
      CALL WINDOW(-100,100,-100,100,-400,-5000,0)
       .
       .
       .
```

Example 5.5-7

```
C
C    VIEW WITH "SOUTHERN" EXPOSURE, BY A ROTATION
C    TRANSFORMATION
C
      CALL WINDOW(-100,100,-100,100,400,5000,0)
      IR180=-32767
      IYAXIS=2
      CALL ROT(IR180,IYAXIS)
       .
       .
       .
```

Example 5.5-8

5.5.8 Placement of the Hither and Yon Planes

For two-dimensional windowing, the hither plane is placed at z=0 and the yon plane at IW (normally 32767). Thus transformed data with a negative z coordinate will be clipped at the hither clipping plane. For three-dimensional windowing (orthographic or perspective), the placement of the hither and yon planes is explicitly specified by the arguments in the window call. By convention if IWH=IWY, then the yon plane will be placed at infinity on the side of hither plane opposite the eye position. If IWH≠IWY then the hither and yon planes will be placed at the positions specified. However, to maintain utmost precision of transformed data, the hither and yon planes should not be given unnecessarily extreme positions; e.g., the hither plan e should ordinarily not be placed immediately in front of the eye. Maximal precision is maintained if the distance between the hither and yon planes is in the same order of magnitude as the width and height of the hither plane.

5.6 ROTATION [ROT]

A rotation transformation is applied to coordinate data using the ROT subroutine to cause a rotation of subsequent data drawn about an axis through the origin of the data space. Thus, if an object is described about the origin of the data space, a rotation transformation will rotate the object about its origin. However, if an object is not described about the origin of its data space, then a rotation transformation will rotate the object about the origin of the data space. The effect would be that of swinging the object on a string rather than tumbling it. In order to rotate such as object about its own origin, it would first need to be translated to the origin of the data space then rotated and finally translated back to the position it occupied in the data space.

The following is the ROT calling sequence specification of Section 4.1:

```
CALL ROT(IANGLE,IAXIS)
```

The parameters passed to this subroutine specify the angle of rotation (IANGLE) to be applied and the axis (IAXIS) about which the rotation will be performed. The angle of rotation is given by dividing a circle into 2^{16} equal parts, with zero being equal to zero degree and -2^{15} equaling 180 degrees. This method allows a greater amount of precision for rotational values since:

32767/180 = 182.04 = 182 increments/degree
182.04/60 = 3.03 = 3 increments/minute

This allows rotations to be performed to a greater precision without the need for special floating-point hardware or increased execution time due to software floating-point calculations.

Table 5.6-1 shows some common angles and their corresponding IANGLE values. Example 5.6-1 illustrates the continuous rotation of an object about all three axes. It should be noted that a new rotation transformation for each axis is computed for each frame update and these new rotation transformations represent the entire rotation about each axis rather than an incremental rotation for each axis. This technique prevents rotational roundoff error due to sequential matrix concatenations.

Note: Rotation of data through a positive angle appears counter-clockwise when viewed along the specified axis in the positive direction in the left-handed coordinate system.

Table 5.6-1

Angle	IANGLE
30°	5461
45°	8192
60°	10922
90°	16384
180°	32767 or -32767
270° (-90°)	-16384
315° (-45°)	-8192
360° (0°)	0

```
C    ONE DEGREE
     DATA I/182/
C
C    INITIALIZE THE PICTURE SYSTEM
C
       CALL PSINIT(3,0,,,,)
       CALL SETERR(3,-1)
       IANGLX = 0
       IANGLY = 0
       IANGLZ = 0
C
C    PERFORM THE PERSPECTIVE TRANSFORMATION
C
       CALL WINDOW(-10000,10000,-10000,-10000,-10000,-10000)
C
C    BEGIN THE DISPLAY LOOP BY UPDATING THE "ANGLES"
C
100    IANGLX = IANGLX + I
       IANGLY = IANGLY + I
       IANGLZ = IANGLZ + I
C
C    SAVE THE ORIGINAL TRANSFORMATION
C
       CALL PUSH
C
C    ROTATE ABOUT THE Z AXIS
C
        CALL ROT(IANGLZ,3)
C
C    ROTATE ABOUT Y AXIS
C
        CALL ROT(IANGLY,2)
C
C    ROTATE ABOUT THE X AXIS
C
        CALL ROT(IANGLX,1)
C
C    CALL A SUBROUTINE TO DISPLAY THE OBJECT
C
        CALL OBJECT
C
C    RESTORE THE ORIGINAL TRANSFORMATION
C
       CALL POP
       CALL NUFRAM
       GO TO 100
        END
```

Example 5.6-1

Note: The SETERR subroutine is used to avoid FORTRAN error detection of the integer overflow caused by this example. The call of the example is for DOS/BATCH FORTRAN.

5.7 TRANSLATION [TRAN]

A translation transformation is applied to coordinate data, using the TRAN subroutine, to cause a translation of all subsequent data drawn in the X, Y and Z directions of the data space. The following is the TRAN calling sequence specification of Section 4.1:

```
CALL TRAN(ITX,ITY,ITZ[,IW])
```

The parameters passed to this subroutine specify the X (ITX), Y (ITY) and Z (ITZ) translational values.

Translation is often performed after an object has been rotated about its origin. However, in terms of coding an applications program, this means that the TRAN subroutine should be called <u>before</u> the ROT subroutine[1]. This order is illustrated by Example 5.7-1

```
          .
          .
          .
C
C   NOW PERFORM THE TRANSFORMATIONS
C
        CALL  TRAN(ITX,ITY,ITZ)
        CALL  ROT(IANGLZ,3)
        CALL  ROT(IANGLY,2)
        CALL  ROT(IANGLX,1)
C
C   AND DISPLAY THE OBJECT
C
        CALL DRAW3D,(IDATA,INUM,IF1,IF2)
          .
          .
          .
```

Example 5.7-1.

---
[1]See Section 5.2.3 for a further discussion of the placement of CALLs.

If it is necessary to translate an object to a position in the data space which is outside the range of values which can be expressed by a 16-bit number ($\pm 2^{15}-1$), the optional argument [IW] may be used. This argument may be used to increase the effective range of the translational values to $\pm 2^{30}$.

Example 5.7-2 illustrates the calling sequence required to translate an object by 100000 in the X, Y and Z directions.

```
          .
          .
          .
C
C   TRANSLATE THE OBJECT BY 100000 IN X,Y AND Z
C
C   EFFECTIVELY:  CALL TRAN(100000,100000,100000)
C
C   25000 = 100000/4
C    8192 =  32767/4
C
          CALL  TRAN(25000,25000,25000,8192)
          .
          .
          .
```

Example 5.7-2

5.8 SCALING [SCALE]

A scaling transformation is applied to coordinate data, using the SCALE subroutine, to cause an increase or decrease in the size of subsequent data drawn. The following is the SCALE calling sequence specification of Section 4.1:

```
CALL SCALE(ISX,ISY,ISZ[,IW])
```

The parameters passed to the subroutine specify the X (ISX), Y (ISY) and Z (ISZ) scaling values. The scaling values are integers which specify the number of 1/32767 by which coordinate data is to scaled. For example if an object were to be decreased in size by 1/2 in the X, Y and Z axes then the appropriate scaling values would be:

ISX = ISY = ISZ = 1/2•32767 = 16384

and the following calling sequence would be used:

```
CALL SCALE (16384,16384,16384)
```

If ISX = ISY = ISZ = 32767 then the coordinate data would remain unscaled.

If an object is to be increased in size larger than its definition in the data space, the homogeneous coordinate IW is used as described in Section 5.8-3.

5.8.1 Data Distortion

If the scaling values ISX, ISY and ISZ are not equal, they have the effect of distorting pictures by elongating or shrinking them along the directions parallel to the coordinate axes. This may be used to emphasize certain structural characteristics of the data displayed. It should be noted that if the scaling is to be always parallel to the X, Y and Z axes <u>of the object</u>, the scaling should be performed before the object has been rotated about its origin. This means that the SCALE subroutine should be called <u>after</u> the ROT subroutine[1]. This order is illustrated by Example 5.8-1.

---
[1]See Section 5.2.3 for a further discussion of the placement of CALLs.

```
              .
              .
              .
C
C   NOW PERFORM THE TRANSFORMATIONS
C
          CALL TRAN(ITX,ITY,ITZ)
          CALL ROT(IANGLZ,3)
          CALL ROT(IANGLY,2)
          CALL ROT(IANGLX,1)
          CALL SCALE(ISX,ISY,ISZ)
C
C   NOW DISPLAY THE OBJECTS
C
          CALL DRAW3D(IDATA,INUM,IF1,IF2)
              .
              .
              .
```

Example 5.8-1

5.8.2 Mirroring

The mirror image of an object may be generated by using negative values for ISX, ISY or ISZ. With this ability, an object which is symetrical along an axis or axes may be described as a half or quarter image and mirrored to produce a full image for display. Typical mirroring calling sequences are shown in Example 5.8-2.

```
          .
          .
          .
C
C   MIRROR DATA ABOUT THE X-AXIS
C
          CALL PUSH
          CALL SCALE(-32767,32767,32767)
          .
          .
          .
          CALL POP
C
C   MIRROR DATA ABOUT THE Y-AXIS
C
          CALL PUSH
          CALL SCALE(32767,-32767,32767)
          .
          .
          .
          CALL POP
C
C   MIRROR DATA ABOUT THE Z-AXIS
C
          CALL PUSH
          CALL SCALE(32767,32767,-32767)
          .
          .
          .
          CALL POP
          .
          .
          .
```

Example 5.8-2

5.8.3 Scaling Using the Homogeneous Coordinate, IW

Coordinate data may be decreased in size by specifying scaling values for ISX, ISY and ISZ less than 32767 as described in Section 5.8. However, a corresponding increase in size may not be done if ISX=ISY=ISZ=32767 unless the homogeneous coordinate, IW is utilized. As IW is decreased from the value 32767, the effective range of the scaling values ISX, ISY and ISZ is increased to $\pm 2^{30}$.

Example 5.8-4 illustrates the calling sequence required to scale data to twice its size.

```
          .
          .
          .
C
C   NOW SCALE THE DATA TO TWICE ITS SIZE
C   EFFECTIVELY:  CALL SCALE(65534,65534,65534)
C    32767 = 65534/2
C    16384 = 32767/2
C
        CALL SCALE(32767,32767,32767,16384)
          .
          .
          .
```

Example 5.8-4

5.9 DATA DISPLAY

Data that is displayed on THE PICTURE SYSTEM may consist of three data types:

1. Lines and dots
2. Characters
3. Instances

Of these three data types, the first two may be considered the primitives from which the third is constructed. The user is free to utilize each of the data types available without regard for the mixing of the data types and constrained only by the length of the Refresh Buffer and the frame update rate required to provide the dynamic motion of the data displayed. The following sections describe the use of the subroutines contained in the Graphics Software Package which allow the display of each of these data types.

5.9.1 Display of Lines and Dots [DRAW2D,DRAW3D]

The display of two- or three-dimensional data sets as lines or dots is accomplished by calling the DRAW2D or DRAW3D subroutines. The following are the DRAW2D and DRAW3D calling sequence specifications of Section 4.1:

```
CALL DRAW2D(IDATA,NUM,IF1,IF2,IZ[,IW])

CALL DRAW3D(IDATA,NUM,IF1,IF2[,IW])
```

These subroutines are very general; the user specifies using the IF1 parameter, the type of draw function to be performed (i.e. disjoint lines, connected lines, dots) and using IF2 parameter, the mode in which the x,y or x,y,z coordinates are to be interpreted (i.e. absolute, relative, absolute-relative). The valid values for IF1 are:

0 = Disjoint lines from new position.
1 = Disjoint lines from current position.
2 = Connected lines from new position.
3 = Connected lines from current position.
4 = Dot at each point.

The valid values for IF2 are:

0 = absolute-relative-relative-relative-etc.
1 = relative always.
2 = absolute always.

The DRAW2D and DRAW3D subroutines may display the same set in a variety of ways dependent upon the values of the IF1 and IF2 parameters at the time of the call. To illustrate this, Figure 5.9-1 shows the simplistic data set:

```
X1,Y1 = 1,0
X2,X2 = -1,-2
X3,Y3 =  1,-2
X4,Y4 =  1,0
X5,Y5 =  3,2
```

as it would be displayed for each of the valid values of IF1 and IF2 on a grid which ranges from — 4 to 4.

It is assumed for each of these drawings that the current position before the draw begins is at the x,y = 0,1 position. The user is free to utilize these drawing functions in whatever manner is required by his particular application. The decision to use a two-dimensional or three-dimensional data set is dependent upon the data, but the ability to display a two-dimensional data set within a three-dimensional environment is available to the user.

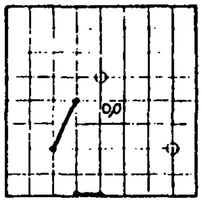
5.9.1.1 Drawing Two-Dimensional Data

Two-dimensional data is defined within a data set as a series of x,y coordinates with constant z and w coordinates for the entire data set. Thus, two-dimensional data is really three-dimensional data which resides in a constant plane and may therefore be ROTated, TRANslated, etc. as a three-dimensional data set. Example 5.9-1 shows a typical call to the DRAW2D subroutine to draw the data set IDATA, which contains five data points, as connected lines from the first data point with all coordinate data interpreted as absolute coordinates. The entire data set will be drawn as if it resides in the Z = 16384 plane.

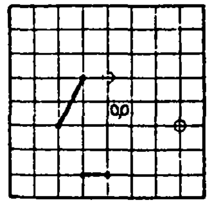
```
      INTEGER IDATA(2,5)
      DATA IDATA/10000,10000,-10000,10000,-10000,-1000(
     1 10000,-10000,10000,10000/
       .
       .
       .
C
C     NOW DRAW THE 2D DATA
C
      CALL DRAW2D(IDATA,5,2,2,16384)
       .
       .
       .
```

Example 5.9-1

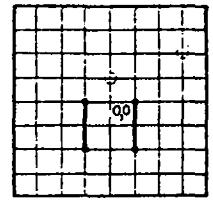
IF1=Ø
DISJOINT LINES FROM NEW POSITION
IF2=Ø
ABSOLUTE-RELATIVE-RELATIVE-ECT.



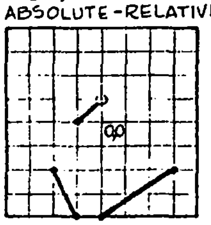
IF1=Ø
DISJOINT LINES FROM NEW POSITION
IF2=1
RELATIVE ALWAYS



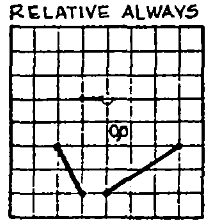
IF1=Ø
DISJOINT LINES FROM NEW POSITION
IF2=2
ABSOLUTE ALWAYS



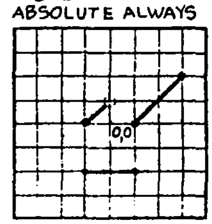
IF1=1
DISJOINT LINES FROM CURRENT POSITION
IF2=Ø
ABSOLUTE-RELATIVE-RELATIVE-ECT.



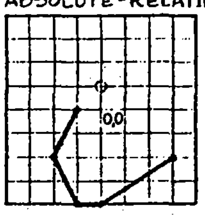
IF1=1
DISJOINT LINES FROM CURRENT POSITION
IF2=1
RELATIVE ALWAYS



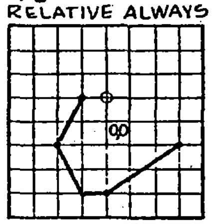
IF1=1
DISJOINT LINES FROM CURRENT POSITION
IF2=2
ABSOLUTE ALWAYS



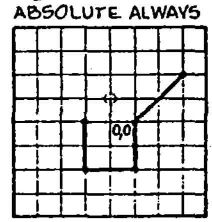
IF1=2
CONNECTED LINES FROM NEW POSITION
IF2=Ø
ABSOLUTE-RELATIVE-RELATIVE-ECT.



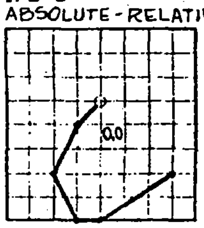
IF1=2
CONNECTED LINES FROM NEW POSITION
IF2=1
RELATIVE ALWAYS



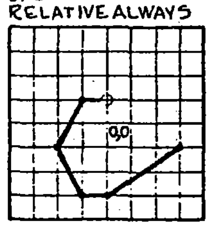
IF1=2
CONNECTED LINES FROM NEW POSITION
IF2=2
ABSOLUTE ALWAYS



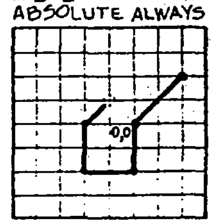
IF1=3
CONNECTED LINES FROM CURRENT POSITION
IF2=O
ABSOLUTE-RELATIVE-RELATIVE-ECT.



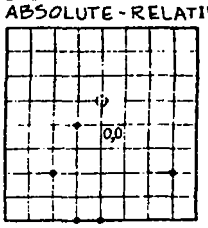
IF1=3
CONNECTED LINES FROM CURRENT POSITION
IF2=1
RELATIVE ALWAYS



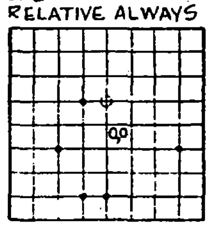
IF1=3
CONNECTED LINES FROM CURRENT POSITION
IF2=2
ABSOLUTE ALWAYS



IF1=4
DOTS AT SPECIFIED POINTS
IF2=O
ABSOLUTE-RELATIVE-RELATIVE-ECT.



IF1=4
DOTS AT SPECIFIED POINTS
IF2=1
RELATIVE ALWAYS



IF1=4
DOTS AT SPECIFIED POINTS
IF2=2
ABSOLUTE ALWAYS

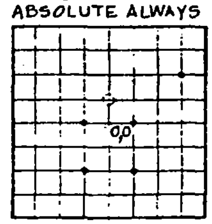


FIGURE 5.9-1
THE DRAWING FUNCTIONS OF THE DRAW2D AND DRAW3D SUBROUTINES
NOTE: A SETPOINT IS INDICATED BY THE O SYMBOL.

When the DRAW2D subroutine is used to draw two-dimensional data, the constant z coordinate (IZ) is used to specify the intensity at which the image is to be displayed[1]. When viewed through a two-dimensional window, IZ = 0 will cause the data to be displayed at the maximum intensity[2] as specified in the call to the VWPORT subroutine. To decrease the intensity of the data displayed through such a window, the user need only adjust the value of IZ to be more positive. In two-dimensional windowing, the intensity of the data displayed decreases linearly as IZ = 0→32767 with 256 distinct levels of intensity available for user specification. Once the intensity level which is required by the user is determined, the value of IZ may be directly computed by the ratio:

$$\frac{IH - IL}{IH - IY} = \frac{IZ}{32767}$$

where:

IH is the hither intensity in the viewport specification.
IY is the yon intensity in the viewport specification.
IL is the intensity level required by the user. $(IH \geq IL \geq IY)$

From this ratio, it can be seen that to display two-dimensional data at an intensity level which is one half the maximum (128), a call such as that shown in Example 5.9-2 would be required.

---

[1]This assumes that a viewport was specified which allows intensity variation (i.e. depth-cueing).

[2]This is because the data is drawn in the same Z plane as the hither clipping plane, which is positioned at Z = 0 for two-dimensional windowing.

```
         .
         .
         .
C
C   SET FOR DEPTH-CUEING AND 2D WINDOWING
C
      CALL VWPORT(-2047,2047,-2047,2047,255,0)
      CALL WINDOW(-10000,10000,-10000,10000)
C
C   NOW DRAW THE DATA AT HALF INTENSITY (LEVEL 128)
C
C   255 - 128     127   1      IZ
C   ---------  =  --- = - =  ----- THEREFORE IZ = 16384
C   255  - 0      255   2    32767
C
      CALL DRAW2D(IDATA,N,2,2,16384)
         .
         .
         .
```

Example 5.9-2

5.9.1.2 Drawing Three-Dimensional Data

Three-dimensional data is defined within a data set as a series of x,y coordinates with a constant (or default) w coordinate. Example 5.9-3 shows a typical call to the DRAW3D subroutine to draw the data set IDATA, which contains five data points, as connected lines from the first data point with all coordinate data interpreted as absolute coordinates.

```
      INTEGER IDATA(3,5)
      DATA IDATA/10000,10000,16384,-10000,10000,16384,
     1 -10000,-10000,16384,10000,-10000,16384,10000,10000,
     1 16384/
          .
          .
          .
C
      CALL DRAW3D(IDATA,5,2,2)
```

Example 5.9-3[1]

---

[1]This example is equivalent to the two-dimensional case of Example 5.9- and would produce the same image if displayed.

When the DRAW3D subroutine is used to draw three-dimensional data, the z-position of the transformed data in relation to the hither and yon clipping planes determines the intensity at which the data is displayed.[1] When viewed orthographically, the intensity at which the data is displayed varies linearly from the hither to yon clipping planes. When viewed in perspective, however, the intensity at which the data is displayed varies reciprocally from the hither to yon clipping planes.

5.9.1.3 Specific Drawing Functions

The DRAW2D and DRAW3D subroutines allow the user to draw data in many modes. Often however, the user needs only a specific drawing mode (i.e. needs to draw only one line or may only need to position to a given point). In cases such as these, the four, five or six parameters of these subroutines calls seem overly complicated. In these cases the user may create subroutines of his own, which in turn call the DRAW2D and DRAW3D subroutines, to perform a specific type of draw function. Examples 5.9-4 and 5.9-5 show how this may be done to provide the two-dimensional "move to" (absolute) or "move" (relative) functions.

---

[1]This assumes that a viewport was specified which allows intensity variation (i.e. depth-cueing).

```
      SUBROUTINE MOVETO(IX,IY)
C
C   THIS SUBROUTINE PROVIDES THE ABILITY TO "MOVE TO" A GIVEN X,Y
C   POSITION BY SPECIFYING ONLY THE X,Y COORDINATES.
C
C   CALLING SEQUENCE:
C   - - - - - - - - - -
C   CALL MOVETO(IX,IY)
C
C   WHERE:
C     IX IS AN INTEGER WHICH SPECIFIES THE X COORDINATE
C     IY IS AN INTEGER WHICH SPECIFIES THE Y COORDINATE
C
      INTEGER IDATA(2)
      IDATA(1)=IX
      IDATA(2)=IY
      CALL DRAW2D(IDATA,1,2,2,0)
      RETURN
      END
```

Example 5.9-4

```
      SUBROUTINE MOVE(IX,IY)
C
C   THIS SUBROUTINE PROVIDES THE ABILITY TO "MOVE" A GIVEN DELTA X
C   AND DELTA Y BY SPECIFYING ONLY THE X AND Y RELATIVE VALUES.
C
C
C   CALLING SEQUENCE:
C   - - - - - - - - - -
C   CALL MOVE(IDX,IDY)
C
C   WHERE:
C      IDX IS AN INTEGER WHICH SPECIFIES THE DELTA X VALUE.
C      IDY IS AN INTEGER WHICH SPECIFIES THE DELTA Y VALUE.
C
     INTEGER IDATA(2)
     IDATA(1)=IDX
     IDATA(2)=IDY
     CALL DRAW2D(IDATA,1,2,1,0)
     RETURN
     END
```

Example 5.9-5

This technique, of course, may be used in conjunction with any of the general purpose subroutines of the Graphics Software Package to provide for PICTURE SYSTEM compatibility with existing graphics applications or to facilitate the development of a device-independent graphics "language".

5.9.2 Display of Characters

The display of characters, represented within the Picture Controller as an ASCII text string, is accomplished by:

1. Calling the CHAR subroutine to specify the size and orientation in which the characters are to appear.
2. Calling the DRAW2D and DRAW3D subroutine to move to the position to where the text is to be displayed.
3. Calling the TEXT subroutine to output the characters to be displayed.

The following are the CHAR and TEXT calling sequence specifications of Section 4.1:

```
CALL CHAR(IXSIZE,IYSIZE,ITILT)

CALL TEXT (NCHARS, ITEXT)
```

5.9.2.1 Character Size and Orientation [CHAR]

The CHAR subroutine may specify a total of 64 character sizes and 2 orientations for text display. Typically though, the X and Y character sizes are equal (or nearly equal; ±1 or 2) so that the characters do not appear extremely flat or thin. The character sizes available are shown in Figure 5.9-2. As this figure illustrates, the sizes may range from 0.07 inches to 0.56 inches. The characters which are displayed may be oriented either horizontally or vertically depending on the value of the ITILT parameter as shown in Figure 5.9-3a and b. The CHAR subroutine may be called at any time during the execution of the user's program and the character size and orientation will remain in effect throughout the duration of the program or until a subsequent call to the CHAR subroutine. Therefore, if the default character specification (horizontal 0.28 characters) is sufficient for the user's application, the CHAR subroutine need not be called at all. It should be noted, however, that character size and orientation changes applied halfway through a given frame will still be in effect at the beginning of the next frame, and thus the default values may no longer be relied on, but must be explicitly specified.

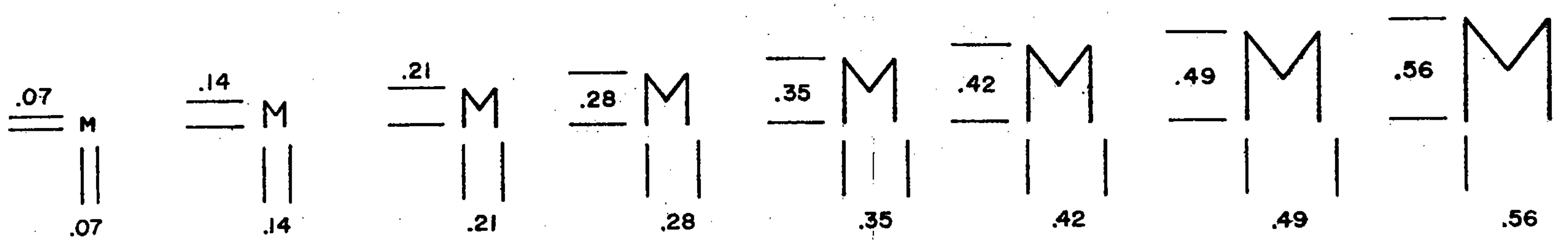


Figure 5.9-2

Standard Character Sizes in Inches.

HORIZONTAL

Figure 5.9-3a

Horizontal Character Orientation (ITILT=0)

VERTICAL

Figure 5.9-3b

Vertical Character Orientation (ITILT≠0)

5.9.2.2 Positioning for Text Display

Text is positioned for display within a two- or three-dimensional environment by calling the DRAW2D or DRAW3D subroutine to perform a move to an x,y (or x,y,z) position or to draw a data set whose last point is the position from which the text string is to be displayed. This position will be at the lower left corner of the first character drawn. The intensity of the move (or of the last line drawn) determines the intensity at which all of the subsequent characters drawn are displayed. Hence, characters may be displayed at any of the 256 levels of intensity. Typically though, characters are displayed at the maximum intensity available and a two-dimensional window is used to make positioning and intensity specification simple. A natural two-dimensional window to use is the one which is initialized when PSINIT is called. This window[1] is one whose boundaries are:

window left boundary:	-32767
window right boundary:	32767
window bottom boundary:	-32767
window top boundary:	32767
hither boundary:	0
yon boundary:	32767

The user then, is free to position anywhere within this window and to define the intensity at which the text string will be displayed (IZ=0 for maximum intensity). To ensure that this window is always available to be used for text positioning, the user should PUSH it before any transformations are performed and POP back to it before the next frame is to be created. Example 5.9-6 shows how this may be done.

The user should note that the viewport in effect at the time the text is positioned for display will determine the position on the screen where the text will be displayed. For example, if the viewport in

---

[1]This two-dimensional window is merely an identity matrix.

```
      INTEGER IDATA(2)
      DATA IDATA/-32767,0/
C
C   INITIALIZE THE PICTURE SYSTEM
C
      CALL PSINIT(3,0,,,,)
C
C   BEGIN THE DISPLAY LOOP BY SAVING THE IDENTITY MATRIX
C
100   CALL PUSH
C
C
      CALL VWPORT(-2048,2047,-2048,2047,255,255)
      CALL DRAW2D(IDATA,1,2,2,0)
      CALL TEXT(18,'THE PICTUE SYSTEM')
C
C   SET THE VIEWPORT AND POINT OF VIEW
C
      CALL VWPORT(0,2047,-2048,0,255,50)
      CALL WINDOW(-1000,1000,-1000,1000,-1000,5000,-5000)
C
C   AND SAVE THE POINT OF VIEW
C
      CALL PUSH
C
C   NOW THE TRANSFORMATIONS
C
      CALL ROT(16384,1)
        .
        .
        .
      CALL POP
C
C   DISPLAY THE NEW FRAME
C
      CALL NUFRAM
C
C   RESTORE THE IDENTITY AND CONTINUE
C
      CALL POP
      GO TO 100
```

Example 5.9-6

effect at the time the text is positioned is the lower right quadrant of the screen, then no x,y coordinate pair could position text for display in any of the other quadrants of the screen. For this reason, text is typically displayed within a viewport whose boundaries are the maximum boundaries of the screen. The user should also note that since the characters are stored in the Refresh Buffer as packed ASCII codes and generated relative to the last character displayed, they are not passed through the clipping process of the Picture Processor and hence, are not clipped at viewport or screen boundaries. If the user attempts to display more character s than may fit within a viewport, the string will extend out into the neighboring area and if the text string extends out to (and past) the screen boundary, the characters will "wrap-around" to the opposite side of the screen where they will continue to be displayed. A similar warning should be issued for the positioning of the text. If the point or line which positions the text is clipped by the Picture Processor, the text string will be displayed positioned from the last x,y coordinates which were placed into the Refresh Buffer. This may lead to confusion for users who are unaware of the cause.

5.9.2.3 Text Output [TEXT]

Text is output for display on THE PICTURE SYSTEM by calling the TEXT subroutine specifying the number of characters to be displayed and an ASCII text string which contains the characters to be displayed, as shown in Example 5.9-7.

```
CALL TEXT (18,'THE PICTURE SYSTEM')
```

Example 5.9-7

This will cause the ASCII character string "THE PICTURE SYSTEM" to be displayed at the position and intensity last specified by a call to the DRAW2D or DRAW3D subroutine (or positioned by previous text display) and at the size last specified in a call to the CHAR subroutine or initialized by PSINIT.

All text is output to the Refresh Buffer as ASCII character codes. The 96 characters which may be displayed are shown, along their ASCII codes, in Table 5.9-1. When these codes are encountered in the Refresh Buffer during the refresh cycle, the Character Generator is called to stroke the character encountered relative to the current position of the beam on the scope.

A character is always stroked (in horizontal or vertical mode) relative to the lower left position of the are a in which any qiven character is defined. Figures 5.9-4a and b show the area in which a character is displayed by the Character Generator.

Table 5.9-1

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
040	space	120	P
041	!	121	Q
042	"	122	R
043	#	123	S
044	$	124	T
045	%	125	U
046	&	126	V
047	'	127	W
050	(	130	X
051	)	131	Y
052	*	132	Z
053	+	133	[
054	,	134	\
055	-	135	]
056	.	136	↑
057	/	137	←
060	0	140	`
061	1	141	a
062	2	142	b
063	3	143	c
064	4	144	d
065	5	145	e
066	6	146	f
067	7	147	g
070	8	150	h
071	9	151	i
072	:	152	j
073	;	153	k
074	<	154	l
075	=	155	m
076	>	156	n
077	?	157	o
100	@	160	p
101	A	161	q
102	B	162	r
103	C	163	s
104	D	164	t
105	E	165	u
106	F	166	v
107	G	167	w
110	H	170	x
111	I	171	y
112	J	172	z
113	K	173	{
114	L	174	|
115	M	175	}
116	N	176	¬
117	O	177	null

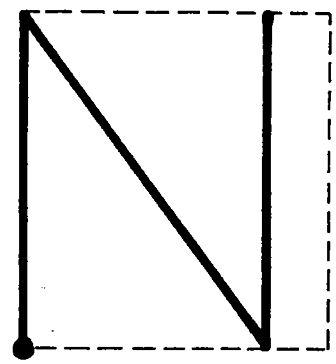


Figure 5.9-4a
Horizontal Character Stroking Relative to the Current Position of the Beam.

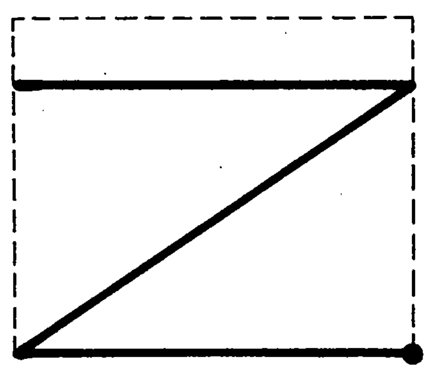


Figure 5.9-4b
Vertical Character Stroking Relative to the Current Position of the Beam.

5.9.3 Instancing [INST,MASTER]

As was pointed out previously, the data which comprises a scene is made up of a series of primitives, i.e. lines, dots and strings of text. These primitives are used, either singly or repeatedly, to generate objects which comprise a scene to be viewed. The ability to create and easily position more complex primitives greatly simplifies construction of a scene. For example, suppose a transistor symbol is defined as a primitive. Then, in order to construct a complicated schematic diagram containing several transistors, the user need only specify the position, size and orientation of every occurrence of the transistor symbol and display of the symbol is then automatic for each occurrence. The technique of constructing and positioning complex primitives such as these, and generating repeated copies thereof, is called instancing. It constitutes one of the more powerful toools of computer graphics.

An instance of a given primitive is invoked using the graphics software by:

1. Calling the INST subroutine to define the boundaries within which the instance is to be placed. These boundaries define the position of the instance within the data space and, typically, also within the WINDOW. If the instance boundaries are outside of the window boundaries, the instance will be totally clipped; if the instance boundaries are partially inside the window boundaries, only that portion of the instance which lies within the window will be displayed; if the instance boundaries are within the window, the entire instance will be displayed.

?

2. Calling the ROT subroutine one or more times if the instance is to appear in an orientation which is different from that in which the original primitive was defined (i.e. if a symbol is to appear rotated 90° from its original position, etc.). This call maybe omitted otherwise.

3. Calling the subroutine which defines and outputs the given primitive.

are the individual lines still clipped?

The following is the INST calling sequence specification of Section 4.1:

```
      CALL INST(INL,INR,INB,INT[,IW])
                       or
      CALL INST(INL,INR,INB,INT,INH,INY[,IW])
```

As described above, the INST parameters define the boundaries within which the instance is to be placed. If the instance is two-dimensional, INST is called with a four[1] argument parameter list such as that shown in Example 5.9-8.

```
C
C    SET A TWO-DIMENSIONAL WINDOW
C
        CALL WINDOW(0,16000,0,16000)
          .
          .
          .
C
C    DISPLAY A TWO-DIMENSIONAL INSTANCE OF A HOUSE
C
        CALL INST(4000,8000,4000,8000)
        CALL HOUSE2
          .
          .
          .
```

Example 5.9-8

If the instance is three-dimensional, INST is called with a Six[2] argument parameter list such as that shown in Example 5.9-9.

---

[1]The four argument parameter list may be extended to five arguments with the inclusion of the scale factor, IW.

[2]The six argument parameter list may be extended to seven arguments with the inclusion of the scale factor, IW.

```
C
C    SET A THREE-DIMENSIONAL WINDOW
C
           CALL WINDOW(0,16000,0,16000,0,0,32000)
             .
             .
             .
C
C    DISPLAY A THREE-DIMENSIONAL INSTANCE OF A HOUSE
C
           CALL INST(4000,8000,4000,8000,2000,6000)
           CALL HOUSE3
             .
             .
             .
```

Example 5.9-9

In the two previous examples, HOUSE2 and HOUSE3 are subroutines which define a graphic primitive or "master copy". A "master copy" defines an object or symbol which is to be instanced and takes the form of a FORTRAN subroutine. A subroutine of this type always contains four parts which must be executed in the following order:

a. A call to the graphics subroutine MASTER to set the boundaries of the data space within which the master copy will be defined.
b. Calls to the various transformation and data output subroutines (ROT,TRAN, DRAW2D, etc.) which define the primitive.
c. A call to the graphics subroutine POP.
d. A FORTRAN RETURN statement.

Example 5.9-10 shows the FORTRAN subroutine, HOUSE2 referenced previously in Example 5.9-8, which contains these four parts.

```
      SUBROUTINE HOUSE2
      INTEGER HOUSE (2,6), DOOR (2,5)
C
C   DEFINE THE TWO-DIMENSIONAL HOUSE DATA
C
      DATA HOUSE/0,32000,32000,16000,32000,-32000,-32000,-32000,
     1 -32000,16000,0,32000/
C
C   AND A DOOR
C
      DATA DOOR/4000,46000,4000,-32000,-4000,-32000,
     1 -4000,-16000,4000,-16000/
C
C   BEGIN THE TWO-DIMENSIONAL MASTER COPY
C
      CALL MASTER(-32767,32767,-32767,32767)
C
C   DRAW THE HOUSE
C
      CALL DRAW2D(HOUSE,6,22,0)
C
C   AND THE DOOR
C
      CALL DRAW2D(DOOR,5,2,2,0)
C
C   NOW RESTORE THE TRANSFORMATION MATRIX AND RETURN
C
      CALL POP
      RETURN
      END
```

Example 5.9-10

The following discussion describes each of the parts, illustrated by Example 5.9-10 in more detail:

a. A call to the MASTER subroutine generates a six-sided, box-shaped enclosure, similar to that produced by an orthographic call to the WINDOW subroutine (in fact the transformations produced by the two routines are exactly identical). This enclosure is used as the definition space for the primitive. As each instance of the primitive is invoked, the "master copy" is mapped (subject to rotation) onto the instance enclosure (defined hereafter). Since all four (or six) boundaries of both enclosures are individually specifiable, the instance may therefore differ in size, shape and location from the master; however, the basic primitives comprising the instance bear the same spatial relationship to each other as do those of the master--in other words, a transistor still looks like a transistor, although its size and shape may be modified. The following are the MASTER subroutine calling sequence specifications of Section 4.1:

```
CALL MASTER (IML,IMR,IMB,IMT[,IW])
              or
CALL MASTER (IML,IMR,IMB,IMT,IMH,IMY[,IW])
```

The parameters define the left, right, bottom, top, hither and yon boundaries of the master enclosure in data space coordinates. For two-dimensional calls, the IMH and IMY parameters are omitted. The origin (and thus the center of rotation) of the master copy is at center of the front boundary of the master enclosure. NOTE: Each instance invoked produces two transformations, the master and the instance which are concatenated. Because instances tend to be small compared to the window in which they are viewed, this concatenated transformation may suffer a loss of precision if the <u>MASTER enclosure is not defined as large as possible</u>. Therefore, a MASTER enclosure should not be defined more than an order of magnitude smaller than the scaling parameter, IW (normally 32767). The data should also be defined so that it extends the full range of the master enclosure.

b. The output comprising the primitive may consist of any executable FORTRAN statements and graphics subroutine calls, normally calls to the subroutines DRAW2D, DRAW3D, TEXT and the transformation subroutines ROT, TRAN and SCALE; as well as calls to other instancing subroutines. Thus, nested instances and even recursive calls to the same primitive definition subroutine are permitted, so long as a conditional exit is provided to prevent infinite recursion, and that sufficient Matrix Stack space is allocated when calling the PSINIT Subroutine, since each instance call results in an implied PUSH. However, the loss of precision previously mentioned is compounded with each level of nesting.

So Master's really specifying an extent!!

While the MASTER call transformation is identical to that of a WINDOW call, data is not clipped at the master boundaries as it is at the boundaries of a normal window; therefore data which extends beyond the master boundaries will be displayed extending beyond the bounds of each specified instance, provided that it does not also extend beyond the bounds of the window.

c. Each call to a master copy subroutine is preceeded by a call to the INST subroutine which contains an implicit PUSH. In order to restore the original transformation for use following the instance, a call to the POP subroutine to restore that transformation must be performed.

d. The subroutine RETURN statement should immediately follow the POP call. Note that the master copy subroutine may just as easily be coded in assembly language, provided that it meets the above specifications and that it is FORTRAN-callable (see Appendix C).

Using this technique, two- and three-dimensional primitives may be defined and libraries of these "master copies" maintained. Example 5.9-11 shows a simple program which uses the primitive defined by Example 5.9-10. Figure 5.9-5 shows the mapping performed by THE PICTURE SYSTEM during instancing.

```
C
C    INITIALIZE THE PICTURE SYSTEM
C
         CALL PSINIT(3,0,,,,)
C
C    SET THE TWO-DIMENSIONAL WINDOW
C
         CALL WINDOW(0,16000,0,16000)
C
C    DISPLAY THE FIRST INSTANCE OF THE HOUSE
C
         CALL INST(4000,8000,4000,8000)
         CALL HOUSE2
C
C    DISPLAY THE SECOND INSTANCE OF THE HOUSE
C    (ROTATED -90°)
C
         CALL INST(14000,18000,10000,14000)
         CALL ROT(-16384,3)
         CALL HOUSE2
C
C    DISPLAY THE DATA
C
         CALL NUFRAM
         PAUSE
C
         STOP
         END
```

Example 5.9-4

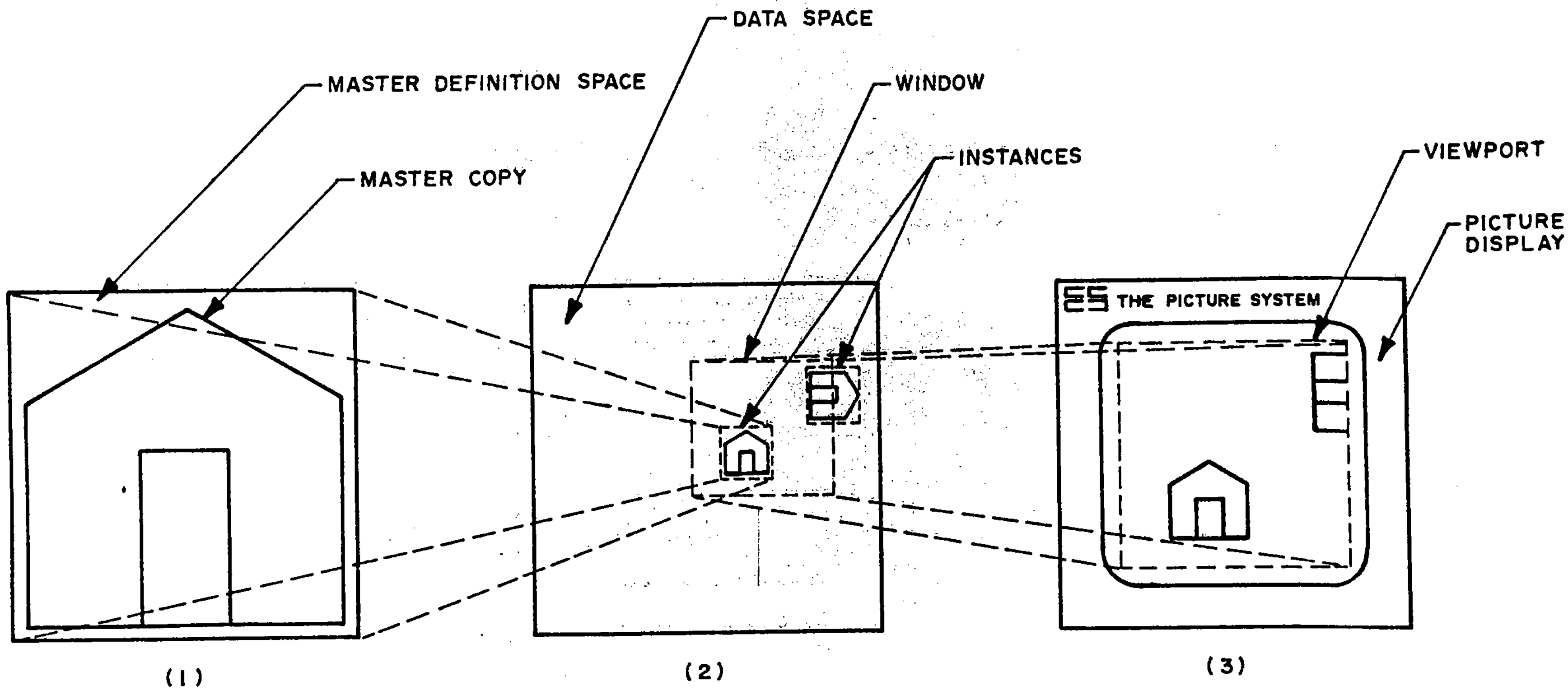


Figure 5.9-5
Illustration of the Mapping of the Instances to the Window and the Window to the Viewport of Example 5.9-11 Showing (1) the master copy definition of Example 5.9-10, (2) the mapping of the master copy into two instance regions of the data space and (3) the Mapping of the Window to the Viewport.

5.9.4 Display Modes

All lines, dots, characters and instances may be displayed in dashed and/or blinking display mode on one to four Picture Displays (or any combination thereof) simultaneously. The following sections describe how each of the display modes are initiated and the manner in which Picture Displays may be selected for output. It should be noted that once set, these display modes remain in effect until the mode is reset with a corresponding subroutine call. Thus, a display may remain in effect for subsequent frames overriding the default setting and even affecting the cursor (which, if used, is output to the beginning of the Refresh Buffer). Therefore, if a user was employing blink mode, it should be reset before calling NUFRAM unless the cursor is intended to blink.

5.9.4.1 Dashed Display Mode [DASH]

When PSINIT is called to initialize THE PICTURE SYSTEM, the display mode is set so that all subsequent data output will be drawn in solid line mode. This mode will remain in effect until the user calls the DASH subroutine to initiate dashed line mode. The following is the DASH calling sequence specification of Section 4.1:

```
CALL DASH(ISTAT)
```

The parameter passed to this subroutine specifies the mode which all subsequent lines will be drawn in, until PSINIT is called to re-initialize the system or DASH is called to reset the line status. If DASH is called with ISTAT≠0, all subsequent data output will be drawn in dashed line mode. Characters may also be displayed in dash modes but may appear indistinguishable because of the dashed lines. A call to DASH with ISTAT=0 resets solid line mode. Dots which are drawn in dashed display mode appear as dots.

5.9.4.2 Blink Display Mode [BLINK]

When PSINIT is called to initialize THE PICTURE SYSTEM the display mode is set so that all subsequent data output will be drawn in non-blink mode. This mode will remain in effect until the user calls the BLINK subroutine to initiate blink display mode. The following is the BLINK calling sequence specification of Section 4.1:

CALL BLINK(ISTAT)

The parameter passed to this subroutine specifies the mode which all subsequent lines will be drawn in, until PSINIT is called to re-initialize the system or BLINK is called to reset the display mode. If BLINK is called with ISTAT=0, all subsequent data output will be displayed non-blinking. If BLINK is called with ISTAT≠0, all subsequent data output will be displayed blinking at the approximate rate of 90 blinks per minute. Example 5.9-10 shows how blink mode may be used to cause a blinking cursor to be displayed.

```
C
C   INITIALIZE THE PICTURE SYSTEM
C
          CALL PSINIT(3,0,,,,)
          CALL TABLET(1,IX,IY,IPEN)
          CALL CURSOR(IX,IY,1,IPEN)
C
C   BEGIN THE DISPLAY LOOP, RESET THE BLINK MODE LEFT
C   FROM THE END OF THE DISPLAY LOOP.
C
100       CALL BLINK(0)
             .
             .
             .
C
C   SET THE BLINK MODE FOR THE CURSOR
C
          CAll BLINK(1)
C
C   NEW FRAME
C
          CALL NUFRAM
          GO TO 100
```

Example 5.9-10

5.9.4.3 Scope Selection [SCOPE]

When PSINIT is called to initialize THE PICTURE SYSTEM, all Picture Displays (Scopes 1-4) are selected for output. This selection will remain in effect until the SCOPE subroutine is called to de-select the Picture Displays to which output is directed. The following is the SCOPE calling sequence specification of Section 4.1:

```
CALL SCOPE(INUM)
```

The parameter passed to this subroutine specifies the Picture Display to be selected (1-4). If INUM is less than 1 or greater than 4, all scopes will be de-selected and all subsequent data output will not be displayed until the SCOPE subroutine is called again to select a Picture Display.

If the user's particular PICTURE SYSTEM configuration has less than four Picture Displays, the selection of all scopes for output incurs no additional overhead, but insures that output will be directed to the Picture Display(s), regardless of the actual configuration of the components of the Picture Generator.

5.10 INITIATING THE DISPLAY OF DATA [NUFRAM,SETBUF]

The manner in which frames are displayed on THE PICTURE SYSTEM is dependent upon the environment in which the control program executes. The environments that are available are:

1. The Refresh Buffer absent from the system configuration.
2. The Refresh Buffer used in single-buffer mode.
3. The Refresh Buffer used in double-buffer mode.

In each of these environments, the basic program structure is the same, and the only difference is the way in which the display of data is initiated. The following sections describe the display of data within each of these environments.

5.10.1 Display of Data Without a Refresh Buffer

THE PICTURE SYSTEM may be configured in what is known as a Starter Configuration. This minimal configuration, shown in Figure 5.10-1, has all of the hardware processing features of the standard PICTURE SYSTEM, but does not include a Refresh Buffer. The absence of the Refresh Buffer requires that as data is transformed, clipped and viewport mapped it be sent directly to the Picture Generator rather than being deposited in the Refresh Buffer for display on the screen. This means that a new frame must be generated by the control program for each refresh cycle. To avoid flicker, the control program is therefore constrained to update the display at least 30 times per second. This limits, to a certain extent, the applications in which THE PICTURE SYSTEM can be used, but provides for programs and applications written for this minimal configuration to be easily upgraded to the more flexible standard PICTURE SYSTEM configuration.

The Graphics Software Package is used as in the standard PICTURE SYSTEM configuration with the exception of the NUFRAM subroutine. This subroutine, usually called to initiate the display of a new frame, is not required in programming a non-Refresh Buffer configuration. This is because all data is displayed as it is transformed. For this reason, the user simply restarts the display loop rather than call NUFRAM[1]. The user variables ICLOCK and IFMCNT are available to the user for display synchronization with the line frequency.

[1]The NUFRAM call may be included but will function as a no-operation call.

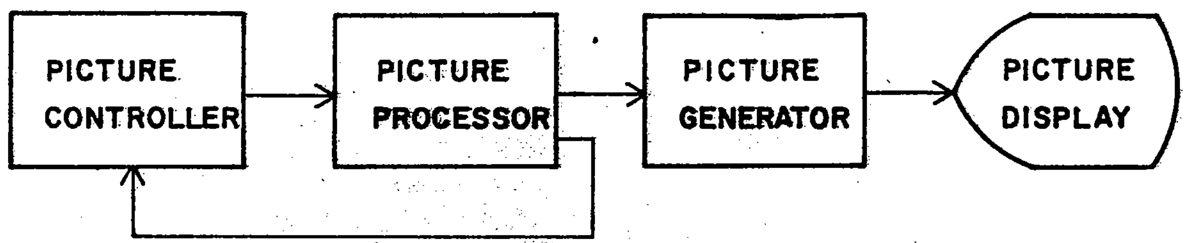


Figure 5.10-1

THE PICTURE SYSTEM Starter Configuration

5.10.2 Display of Data in Single Buffer Mode

THE PICTURE SYSTEM may be used in single-buffer mode when the user's display requirements exceed the capacity of half of the Refresh Buffer. This condition may be diagnosed by the absence of most of the expected data from the picture. (The Refresh Buffer Address Register wraps around, leaving only the last data drawn available for display.) The user selects the single-buffer mode of the Refresh Buffer by calling the SETBUF subroutine as illustrated in Example 5.10-1.

```
C
C   INITIALIZE THE PICTURE SYSTEM
C
       CALL PSINIT(3,0,,,,)
C
C   AND SET THE REFRESH BUFFER TO SINGLE BUFFER MODE
C
       CALL SETBUF(1)
C
C   BEGIN THE DISPLAY LOOP
C
          .
          .
          .
```

Example 5.10-1

The user may select single- or double-buffer modes at any time during the execution of any given program. The user should, however, be aware of the subtle difference between the use of the Refresh Buffer in single- and double-buffer modes. As Figure 5.10-2 illustrates, in single buffer mode, the data displayed on the Picture Display is the same data which is being updated by the user program. This may result in a refresh cycle which displays a portion of the user's old data (old frame). In cases where the data may not change drastically from frame to frame or where there are many refresh cycles between frame updates, this

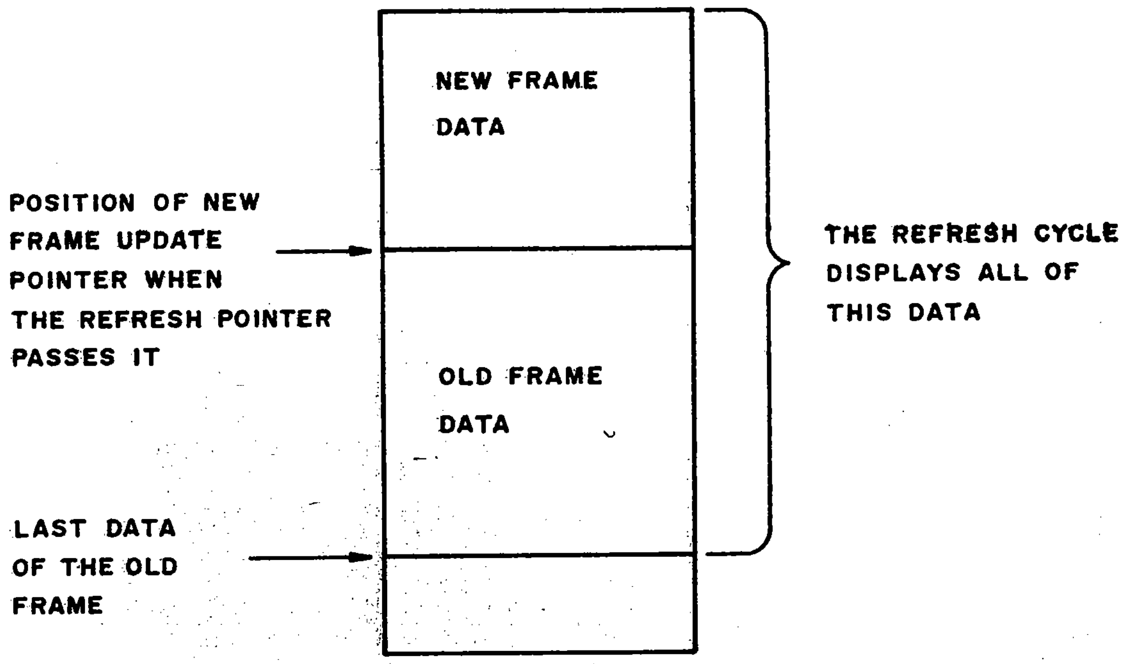


Figure 5.10-2

A Single Buffered Refresh Buffer

be of little consequence. The structure of a program which uses single-buffer mode is, nonetheless, the same as if the Refresh Buffer were double-buffered. In either case the user draws[1] all of the data that is to be displayed and then calls the NUFRAM subroutine so that the next data drawn will be stored at the beginning of the Refresh Buffer. This is shown in Example 5.10-2. If no subsequent data is drawn, the picture will appear static on the screen. Alternately, the single-buffered Refresh Buffer may be used in a manner similar to a storage tube display. In this manner, the user fills the Refresh Buffer with the data which is to be viewed. This data will continue to be refresed until an "erase" of the Refresh Buffer is initiated by the user. The equivalent of an "erase" is provided by calling the NUFRAM subroutine twice in succession. Therefore, a user could write an ERASE subroutine. Example 5.10-3, which could then be called to "erase" the Picture Display.

---

[1]The term draws here means that the data will be transformed, clipped, viewport mapped and stored into the Refresh Buffer where it will be displayed (or drawn) upon the next refresh cycle.

```
C
C    INITIALIZE THE PICTURE SYSTEM
C
          CALL PSINIT(3,0,,,,)
C
C    SET THE REFRESH BUFFER TO SINGLE BUFFER MODE
C
          CALL SETBUF(1)
C
C    SET THE WINDOWING TRANSFORMATION
C
          CALL WINDOW(-4000,4000,-4000,4000,-4000,4000,8000)
C
C    SAVE WINDOWING TRANSFORMATION AND BEGIN THE DISPLAY LOOP
C
100       CALL PUSH
C
C    MODIFY OR OBTAIN NEW TRANSFORMATION PARAMETERS
C
             .
             .
             .
C
C    CONCATENATE THE TRANSFORMATIONS
C
          CALL TRAN(ITX,ITY,ITZ)
          CALL ROT(IANGLZ,3)
          CALL ROT(IANGLY,2)
          CALL ROT(IANGLX,1)
          CALL SCALE(ISX,ISY,ISZ)
C
C    NOW TRANSFORM THE DATA BY THE COMPOUND TRANSFORMATION
C
          CALL DRAW3D(IDATA,N,IF1,IF2)
C
C    AND DISPLAY THE DATA AND LOOP AGAIN
C
          CALL NUFRAM
          CALL POP
          GO TO 100
             .
             .
             .
```

Example 5.10-2

```
      SUBROUTINE ERASE
C
C   THIS WILL ESSENTIALLY ERASE THE CONTENTS OF THE
C   REFRESH BUFFER ALLOWING THE PICTURE SYSTEM TO BE
C   USED AS IF IT WERE A STORAGE TUBE DISPLAY.
C
C   NOTE:  THE "ERASURE" WILL TAKE ONE REFRESH CYCLE
C          AS DEFINED IN THE CALL TO PSINIT.
C
      CALL NUFRAM
      CALL NUFRAM
      RETURN
      END
```

Example 5.10-3

The user is free to use the single buffered Refresh Buffer in either of the previous ways described, or in some combination thereof. Example 5.10-4 illustrates this with the use of the ERASE subroutine of Example 5.10-3 only between major frame changes.

```
      SUBROUTINE MFRAM2
C
C   THIS SUBROUTINE DISPLAYS THE 2ND MAJOR FRAME OF
C   THIS PROGRAM.  IT IS ASSUMED THAT AN ERASE WAS
C   PERFORMED IMMEDIATELY BEFORE THIS MODULE WAS
C   CALLED (WITH THE REFRESH BUFFER SET TO SINGLE
C   BUFFER MODE).
C
C   BEGIN DISPLAY LOOP
C
          .
          .
          .
C
C   EXIT FROM THIS MODULE YET ?   GO TO 1000 IF SO
C
      IF (IDONE.NE.0) GO TO 1000
C
      CALL NUFRAM
      GO TO 100
C
C   POP THE ORIGINAL MATRIX, ERASE THE DISPLAY AND EXIT
C
1000  CALL POP
      CALL ERASE
      RETURN
      END
```

Example 5.10-4

5.10-3 Display of Data in Double-Buffer Mode

The Refresh Buffer is typically used in what is termed double-buffer mode where the Refresh Buffer is divided into two separate buffers. For this reason, the default usage of the Refresh Buffer is double-buffered, initialized to that state by PSINIT when called by user program. In this mode, the user fills a buffer with data to be displayed, calls the NUFRAM subroutine to initiate its display and then may proceed to fill the other buffer with new frame data. This is illustrated by Figure 5.10-3. This method of frame display frees the user to create a new frame at his leisure without worry of degradation of his picture. Example 5.10-5 shows the structure of the display loop of a typical applications program which utilizes the double-buffer mode. If the user's application requires that the Refresh Buffer be used in single-buffer mode for a given set of frames and then returned to double-buffer mode, it would be done as shown in Example 5.10-6.

```
C   INITIALIZE THE PICTURE SYSTEM
C
        CALL PSINIT(3,0,,,,)
C
C   SET THE WINDOWING TRANSFORMATION
C
        CALL WINDOW(-4000,4000,-4000,4000,-4000,4000,8000)
C
C   SAVE THE WINDOWING TRANSFORMATION
C
100      CALL PUSH
C
C   MODIFY OR OBTAIN NEW TRANSFORMATION PARAMETERS
C
          .
          .
          .
C
C   CONCATENATE THE TRANSFORMATIONS AND DISPLAY THE DATA
C   TWICE

         CALL TRAN(ITX,ITY,ITZ)
         CALL ROT(IANGLZ,3)
         CALL ROT(IANGLY,2)
         CALL ROT(IANGLX,1)
         CALL PUSH
         CALL SCALE(ISX1,ISY1,ISZ1)
         CALL DRAW3D(IDATA,N,IF1,IF2)
         CALL POP
         CALL SCALE(ISX2,ISY2,ISX2)
         CALL DRAW3D(IDATA,N,IF2,IF2)
C
C   RESTORE THE ORIGINAL WINDOW THAT WAS SAVED
C
         CALL POP
C
C   AND DISPLAY THE DATA AND LOOP AGAIN
C
         CALL NUFRAM
         GO TO 100
```

Example 5.10-5

```
C
C    ENTER THE DISPLAY SEGMENT WHICH MUST USE SINGLE BUFFER
C
         CALL SETBUF(1)
C
C    BEGIN DISPLAY OF THE SINGLE BUFFERED DATA
C
500             .
                .
                .
C
C    EXIT SINGLE BUFFER MODE?  (LOOP IF NOT)
C
         IF (IDONE.EQ.0) GO TO 500
C
C    RESET BUFFER MODE TO DOUBLE BUFFERED
C
         CALL SETBUF(2)
                .
                .
                .
```

Example 5.10-6

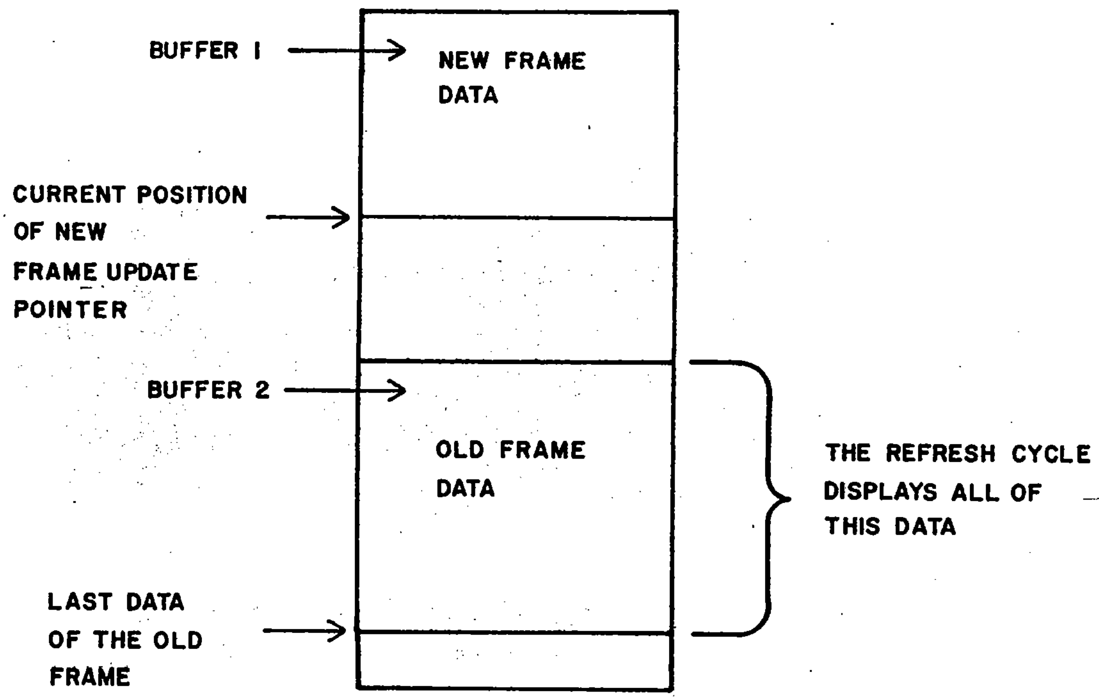


Figure 5.10-3

A Double-Buffered Refresh Buffer.

5.11 INTERACTION USING THE TABLET

Data may be input to THE PICTURE SYSTEM by any of the various standard DEC peripherals available for the PDP-11, or by any of the standard Evans & Sutherland graphical input devices. The Tablet, however, serves as the standard, general purpose graphic input device for THE PICTURE SYSTEM, performing those interactive functions usually reserved for such graphic input devices as light pens, joy sticks and function switches. This section discusses in detail how the Tablet may be used to perform pointing, positioning, and other miscellaneous functions required for flexible interactive data input.

5.11.1 Tablet and Cursor Use [TABLET,CURSOR,ISPDWN]

Data is input from the Tablet within a user application program by calling the TABLET subroutine. The following is the TABLET calling sequence specification of Section 4.1:

CALL TABLET (ISTAT[,IX,IY,IPEN])

This subroutine is called with ISTAT=0 to read the current pen x,y coordinates and status. The pen x,y coordinates which are returned in the IX and IY parameters are scaled[1] integer values whose approximate range is ±32700. The Tablet is considered to be a two-dimensional input device whose coordinate system origin is at the center of the tablet, as shown in Figure 5.11-1.

This coordinate system was chosen for the tablet so that the values returned to the user could be used directly for pointing, positioning and tracking. The pen status is returned to the user in the parameter IPEN, so that the pen information may be determined by the user. The status that is returned is the information as read directly from the tablet. The status information returned is shown in Figure 5.11-2.

---

[1] The x,y coordinate values are scaled from the actual tablet coordinate range $(0\text{-}1777_8)$ to the approximate data space range ±32700 $(\pm77700_8)$.

As Figure 5.11-2 shows, the user may determine when the pen is down (i.e. pressed against the surface of the tablet) by testing bit 1 of the pen status word. Since bit testing capabilities are not provided directly by FORTRAN, the integer function subroutine ISPDWN is available to the FORTRAN programmer to determine whether or not the pen is down. This function subroutine returns the value 0 if the pen is not down or 1 if the pen is down. This is illustrated by Example 5.11-1.

```
            .
            .
            .
C
C   READ THE TABLET VALUES AND PEN STATUS
C
         CALL TABLET (0,IX,IY,IPEN)
C
C   IF THE PEN IS DOWN, GO TO 100
C
         IF(ISPDWN(IPEN).NE.0) GO TO 100
C
C   THE PEN IS NOT DOWN, SO CONTINUE
C
            .
            .
            .
```

Example 5.11-1

The user may choose to utilize the tablet in what is termed automatic mode by setting the ISTAT parameter to a non-zero value. In this mode, the user "turns on" the TABLET subroutine and the pen x,y coordinate and status are then updated automatically upon each refresh interrupt. In this way, the user constantly has avilable the most recent tablet values without explicitly calling the TABLET subroutine. Example 5.11-2 shows how the tablet may be used in automatic mode.

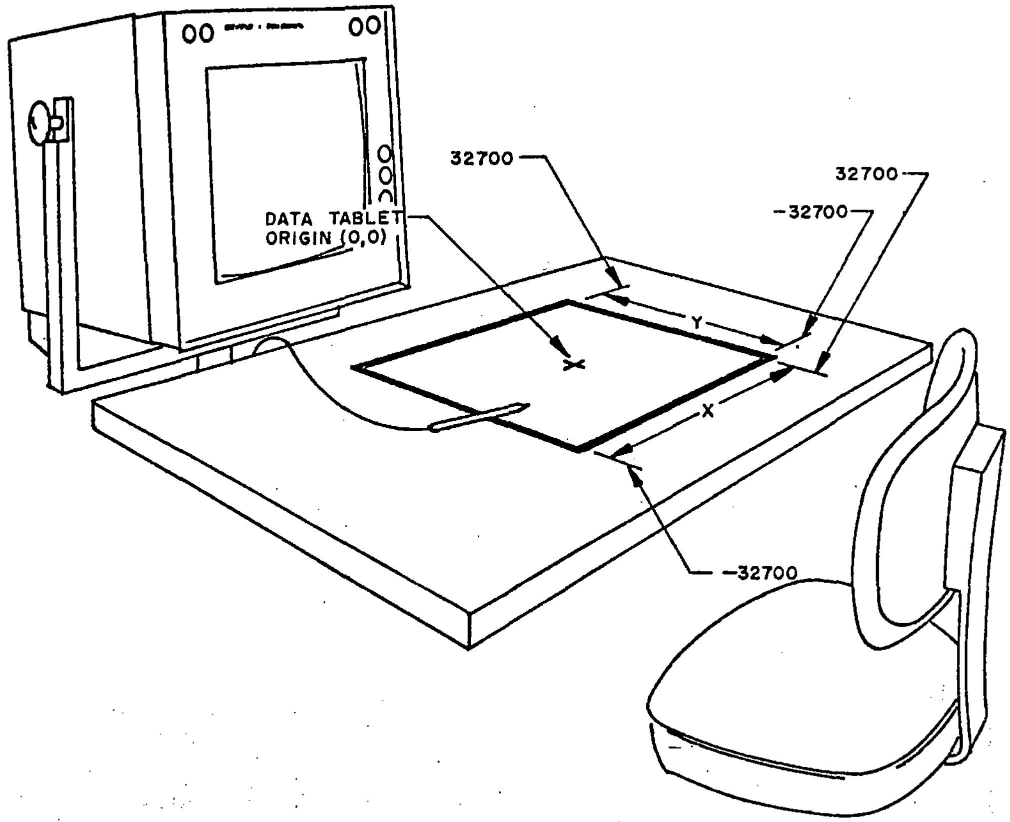


Figure 5-11-1

The Two-dimensional Coordinate System of the Tablet

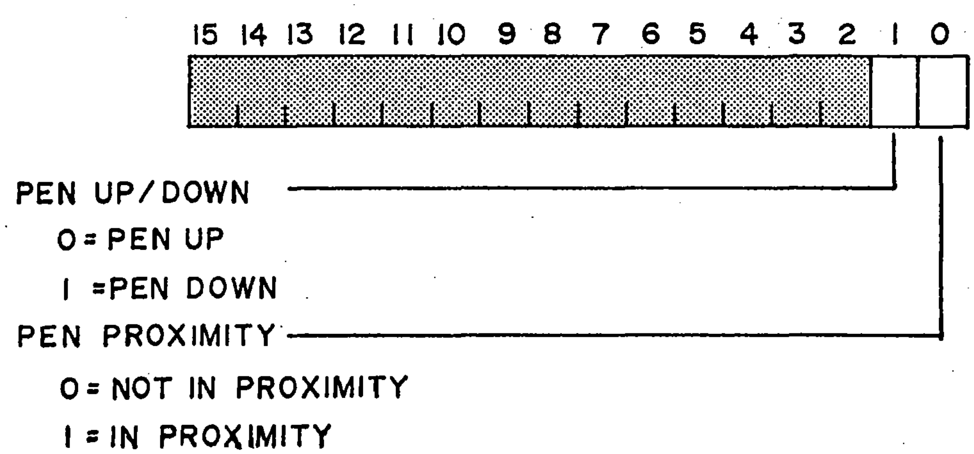


Figure 5.11-2

The Pen Status as Returned by the TABLET Subroutine

```
          INTEGER IX,IY,IPEN
            .
            .
            .
C
C    INITIALIZE THE PICTURE SYSTEM
C
          CALL PSINIT(3,0,,,,)
C
C    AND TURN ON THE TABLET FOR AUTOMATIC MODE
C
          CALL TABLET (1,IX,IY,IPEN)
            .
            .
            .
C
C    BEGIN THE DISPLAY LOOP BY SEEING IF THE PEN IS DOWN
C
100       IF(ISPDWN(IPEN).EQ.0)GO TO 200
C
C    THE PEN WAS DOWN...
C
            .
            .
            .
```

Example 5.11-2

When used in either automatic or non-automatic mode, the TABLET subroutine requires the user to acknowledge that the pen information has been read by clearing the IPEN parameter. If this parameter is not zero when the tablet values are to be updated, then the x, y and pen status will not be updated unless the pen is down. This requirement ensures that the user will not "miss" an occasion when the pen has been set down and always has the most recent position (x,y) where the pen was set down. Example 5.11-3 illustrates the clearing of the IPEN parameter after the pen position has been determined.

```
              .
              .
              .
C
C    IF THE PEN IS NOT DOWN, BRANCH TO 200
C
100        IF (ISPDWN(IPEN).EQ.0) GO TO 200
C
C    THE PEN IS DOWN...DETERMINE THE MENU SELECTION
C
              .
              .
              .
C
C    MENU SELECTION DETERMINED...INDICATE PEN POSITION
C    READ
C
           IPEN=0
C
C    CONTINUE DISPLAY LOOP
C
200        CONTINUE
              .
              .
              .
```

Example 5.11-3

It is often convenient to provide a visual feed back of the current pen position in relation to the tablet. For this purpose, a "cursor" may be drawn on the Picture Display at a position which corresponds to the x,y position of the pen on the tablet by calling the CURSOR subroutine. The following is the CURSOR calling sequence specification of Section 4.1:

```
CALL CURSOR(IX,IY,ISTAT[,IPEN])
```

This calling sequence allows a cursor symbol to be displayed at the position specified by the IX and IY parameters. The cursor which is displayed is a simple cross which is centered at the x,y coordinate specified. The pen status (IPEN) is an optional argument which, if specified, provides visual feedback of the pen status by displaying the cursor brighter whenever the pen is down, and also provides the information so that the cursor will not be displayed when the pen is not in the proximity of the tablet. If the argument is not specified, the cursor will always be displayed at maximum intensity. Example 5.11-4 shows the use of the CURSOR subroutine with the optional argument.

As with the TABLET subroutine, the user may optionally choose to display a cursor in what is termed automatic mode, by setting the ISTAT parameter to a non-zero value. In this mode, the user "turns on" the CURSOR subroutine and a cursor will then be displayed automatically upon each refresh interrupt. In this way, the user constantly has displayed the current pen position without explicitly calling the CURSOR subroutine. When used in automatic mode in conjunction with the TABLET subroutine, the user will always have the current position of the pen displayed regardless of the frame update rate of a particular applications program. Example 5.11-5 shows how the automatic mode of the TABLET and CURSOR subroutine may be specified.

```
      INTEGER IX,IY,IPEN
C
C   INITIALIZE THE PICTURE SYSTEM
C
      CALL PSINIT(3,0,,,,)
C
C   "TURN ON" AUTOMATIC MODE FOR THE TABLET AND CURSOR
C
      CALL TABLET (1,IX,IY,IPEN)
      CALL CURSOR (IX,IY,1,IPEN)
C
C   NOW BEGIN THE DISPLAY LOOP WITHOUT WORRYING
C   ABOUT TABLET UPDATE AND CURSOR DISPLAY.
C
```

Example 5.11-5

It should be noted that the tablet x,y coordinates need not be used to position the cursor. If the cursor is to be positioned by other means (i.e. control dials, arithmetic computations, etc.), the variable which will contain the x or y positioning information should be specified rather than the pen coordinate variables. The CURSOR subroutine, however, expects an x,y position value in the range of approximately ±32700 to be specified. There is, of course, no restriction on the use of values to specify that the cursor always be displayed at a given x or y position, as shown by Example 5.11-6.

```
      INTEGER IX,IY,IPEN,ZERO
      DATA ZERO/0/
C
C   INITIALIZE THE PICTURE SYSTEM
C
      CALL PSINIT(3,0,,,,)
C
C   SET TABLET, CURSOR AUTOMATIC MODE(CURSOR ALWAYS AT
C   Y=0)
C
      CALL TABLET (1,IX,IY,IPEN)
      CALL CURSOR (IX,ZERO,1)
      .
      .
      .
```

Example 5.11-6

The cursor is defined in a window running from -32767 to +32767 in x and y, which coincides appoximately with the range of tablet values.

The cursor will always be displayed within a viewport which is specified by the variables which defined the viewport in effect when the CURSOR subroutine was called. In Examples 5.11-5 and 5.11-6, this means that the cursor will always be displayed in a viewport which is the entire screen (since PSINIT last specified a viewport). However, as Example 5.11-7 shows, a cursor can be displayed within a dynamically changing viewport merely by calling the VWPORT subroutine before initiating automatic TABLET and CURSOR modes and then modifying the variables which defined the viewport. This feature proves useful in menu and data pointing functions.

```
      INTEGER IVL,IVR,IVB,IVT,IH,IY
      DATA IVL,IVR,IVB /-2048,2047,-2048/
      DATA IVT,IVH,IVY/2047,255,0/
C
C   INITIALIZE THE PICTURE SYSTEM
C
      CALL PSINIT(3,0,,,,)
C
C   SET THE  INITIAL VIEWPORT
      CALL VWPORT(IVL,IVR,IVB,IVT,IH,IY)
C
C   SET TABLET, CURSOR AUTOMATIC MODE
C
      CALL TABLET (1,IX,IY,IPEN)
      CALL CURSOR (IX,IY,1,IPEN)
      IPOINT=0
C
C   BEGIN THE DISPLAY LOOP...
C
        .
        .
        .
C
C   IF IPOINT=0 THEN RESET THE MAX VIEWPORT FOR
C   CURSOR DISPLAY, OTHERWISE SET ANOTHER VIEWPORT
C
      IF(IPOINT.EQ.0) GO TO 200
      IVL=-1024
      IVR= 1024
      IVB=-1024
      IVT= 1024
      GO TO 210
C
C   RESET THE VIEWPORT VARIABLES FOR MAX SIZE VIEWPORT
C
200   IVL=-2048
      IVR= 2047
      IVB=-2048
      IVT= 2047
210   CONTINUE
        .
        .
        .
```

Example 5.11-7

5.11.1 Pointinq

The user may input data interactively with the tablet by:

1. selecting a menu item which specifies a command to be performed.
2. identifying a data element with which the user wishes to interact.

Both of these functions may be considered to be pointing functions; i.e. the user points to a menu item or points to a particular data element. However, the implementations of the two pointing functions are typically different. The following two sections describe the use of the Tablet to perform these two pointing functions.

5.11.2.1 Pointing at Menu Items

A menu item is a symbol, usually text, which when selected by the user issues a command to the user's program. Figure 5.11-3 shows a menu which includes both text and other symbols as menu items. In this program, the user would position the pen on the tablet to the location which would correspond to the menu item to be selected, and press the pen down indicating to the program that the particular menu item, whose boundaries contain that x,y position, is being selected. The program would then initiate the action required for the particular menu item selected. The user may programmably determine which (if any) menu item is being selected by comparing the x,y coordinates of the pen with the boundaries defined for each of the menu items. However this comparison need be performed only if the pen is down. Example 5.11-8 illustrates how this may be done.

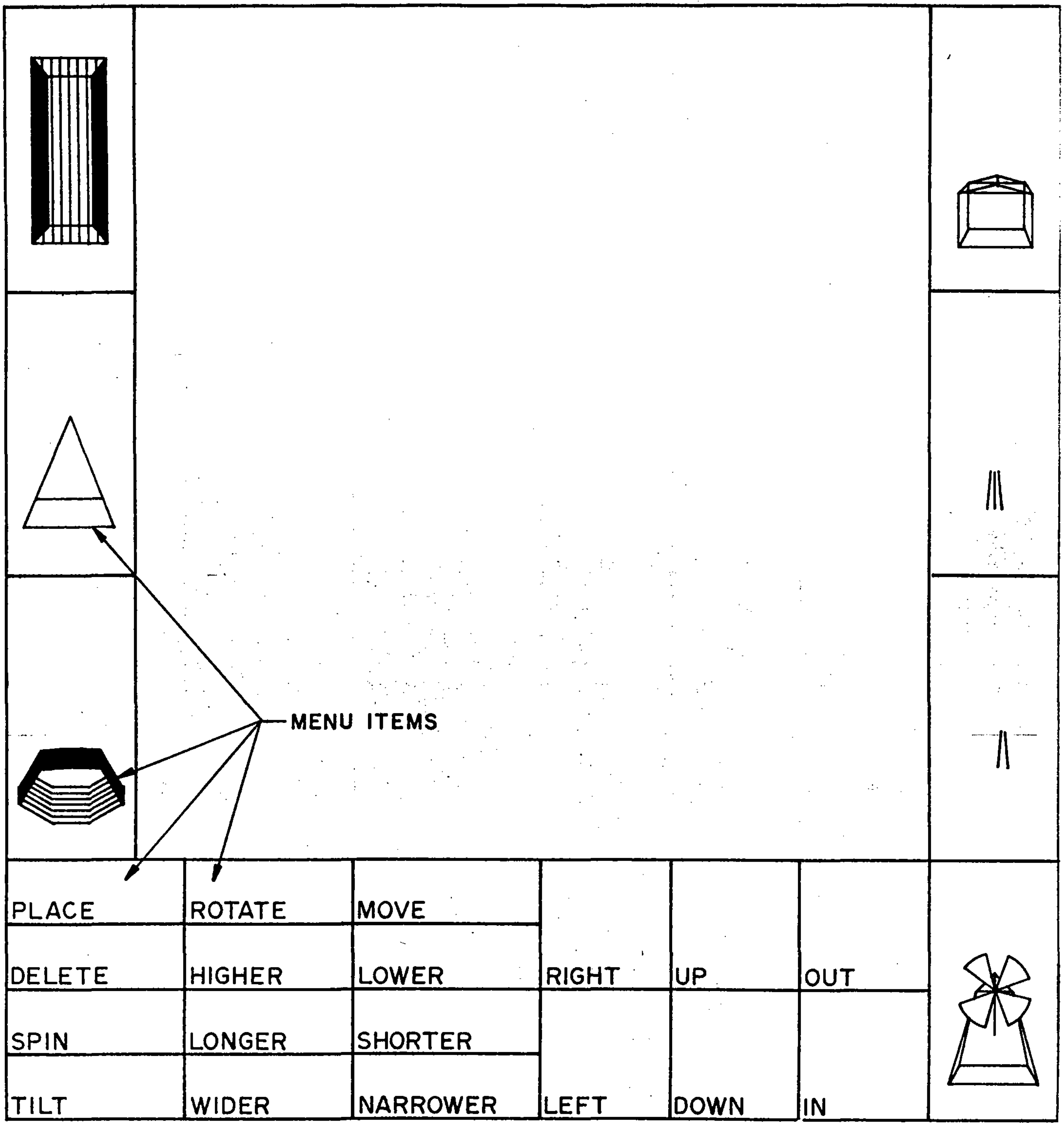


Figure 5.11-3

A Menu which Includes both Text and other Symbols

```
          .
          .
          .
C
C   IF THE PEN IS DOWN, ENTER THE MENU SELECTION CODE,
C   OTHERWISE, BRANCH TO 300
C
          IF(ISPDWN(IPEN).EQ.) GO TO 300
C
C   MENU SELECTION...COMPARE THE MENU AREAS WITH
C   THE TABLET X,Y COORDINATES
          IF(IY.LE.0) GO TO 250
C
C   UPPER PORTION OF MENU AREA, LEFT OR RIGHT SIDE?
C
          IF(IX.LE.24576) GO TO 230
C
C   UPPER RIGHT SIDE...UPPER OR LOWER MENU ITEM?
C
          IF(IY.LE.16384) GO TO 210
C
C   UPPER,UPPER RIGHT MENU ITEM...PERFORM SELECTED
C   FUNCTION
C
          .
          .
          .
C
C   MENU ITEM SELECTION COMPLETED, INDICATE PEN
C   POSITION READ
C
          IPEN=0
300       CONTINUE
          .
          .
          .
```

Example 5.11-8

A feature of the tablet is that the menu may be only a paper overlay which is placed on the tablet, or a menu may be displayed on the screen which corresponds to the menu areas on the tablet, enabling the user to point with the cursor to the menu area on the screen to select a menu item. An additional feature is that a viewport may be defined within which the non-menu data may be mapped and displayed without extending into the menu areas, as shown in Figure 5.11-4.

5.11.2.2 Pointing at Data Elements [HITWIN,HITEST]

A data element is typically pointed at by the user to indicate that a particular function is to be performed upon, or in relation to, the data element pointed at. Such functions might be the deletion of the data element, the stress computation on the element in relation to its neighboring elements, or any other function which may be programmed as a particular application. This pointing function, often erroneously considered to be strictly a light pen operation, is performed with THE PICTURE SYSTEM by what is known as "hit testing". This function, performed by the Picture Processor's clipping process, allows a "hit window" to be defined through which all data in question may be processed to determine whether any of the data was "hit"; i.e. whether any point (visible or not), or any part of any line, fell within the "hit window".

This process is superior to the analogous function of the light pen in several ways:

1. The hit testing feature, when coupled with the TABLET and CURSOR subroutines, allows the user the ability to point at and identify data elements, with the added flexibility that the size of the "hit window" or region of interest described about the pen position may be varied dynamically to allow a wide range of pointing resolution upon user demand.
2. The "hit window", while usually positioned by the x,y coordinates of the pen on the tablet, may be specified arithmetically, allowing data which is not even displayed to be "hit".
3. The "hit testing" technique requires no trace back within the display file to determine which data element was "hit" since the user programmal controls the level to which "hit testing" is performed.

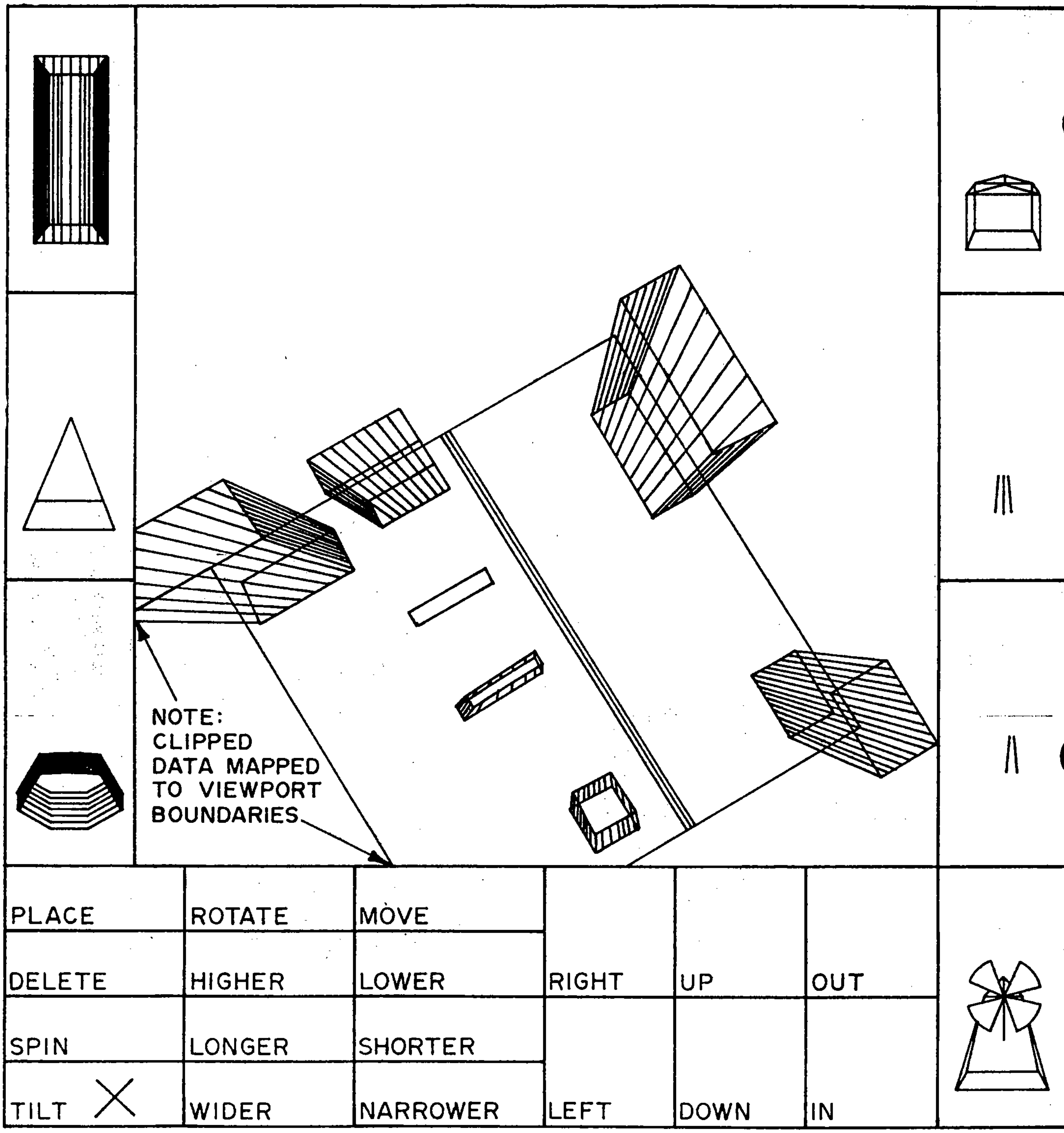


Figure 5.11-4

A Displayed Menu Illustrating Pointing at Menu Item with the Cursor and Data which had been Clipped, Mapped to the Viewport Boundaries.

The "hit testing" capability is provided within the Graphics Software Package by the HITWIN and HITEST subroutines. The following are the HITWIN and HITEST calling sequence specifications of Section 4.1:

```
CALL HITWIN(IX,IY,ISIZE[,IW])
CALL HITEST (IHIT,ISTAT)
```

The HITWIN subroutine is called to specify a "hit window" through which data may be processed to determine whether any of the data was "hit". HITWIN also suspends output to the Refresh Buffer, since "hit" testing uses the transformation and clipping facilities of the Picture Processor in a way which would result in misplaced picture elements if they were allowed to be displayed. The "hit window" is centered at the x,y coordinates specified by the IX and IY parameters and whose half-width and half-height is specified by the ISIZE parameter. All three parameters will be scaled by the homogeneous coordinate, IW, if it is specified. Such a "hit window" is considered to have finite boundaries in x and y directions (determined by the ISIZE parameter) and to extend from 0 to IW in the Z direction as shown in Figure 5.11-5. The size of the "hit window" may be varied by modifying the value of the ISIZE parameter. The actual size of the hit window (in inches) may be determined by the following ratio:

$$\frac{\text{ISIZE}}{\text{IW}} = \frac{\text{actual "hit window" width}}{\text{actual "hit testing" viewport width}}$$

With a "hit testing" viewport which is the entire screen (10 inches_and IW its typical default value (IW=32767), this ratio would reduce to:

$$\frac{\text{ISIZE}}{32767} = \frac{\text{actual "hit window" width}}{10}$$

In this case, to achieve a "hit window" which is 1 inch in width:

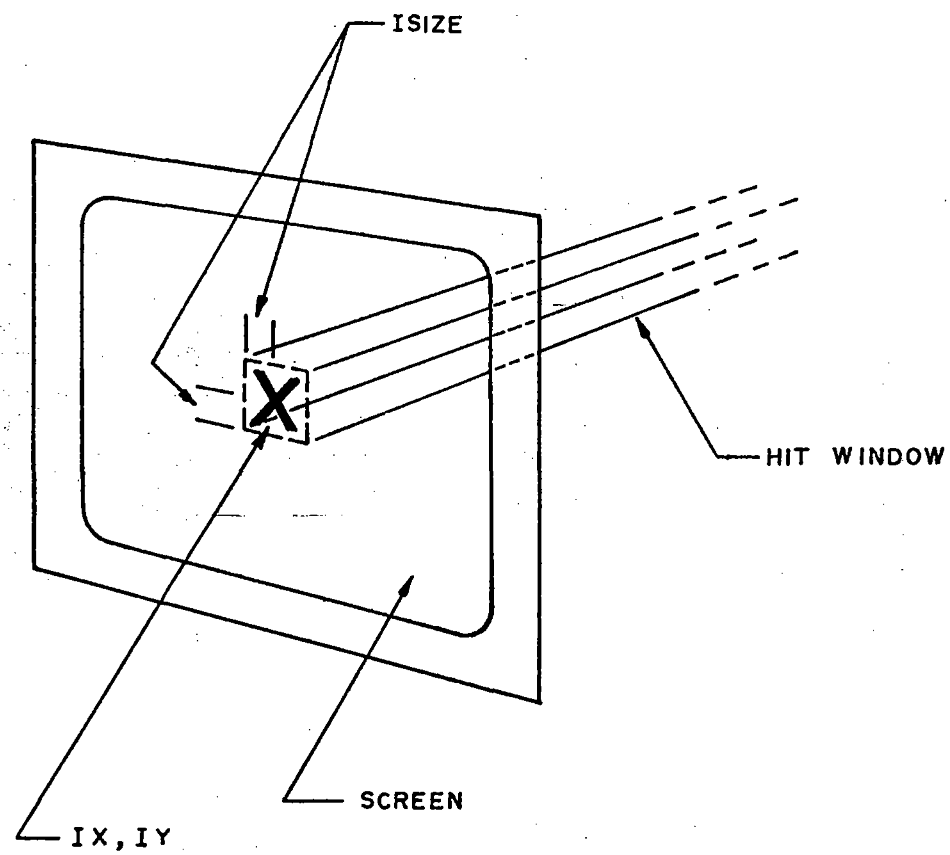


Figure 5.11-5

The "Hit Window" as Specified by the HITWIN Subroutine Illustrating its Boundary = IW.

Left Blank Intentionally.

```
ISIZE     1
----- = --   or ISIZE = 3276
32767    10
```

These subroutines allow the tablet to be used in a manner similar to a light pen, i.e. any data element which appears behind the "hit window" will be "hit" if tested during the "hit testing" process. Hit testing is performed at a stage in the Picture Processor's operation where pictorial data has been completely transformed, put in perspective and mapped onto a region running from -32767 to +32767 in both x and y, which is identical to the region in which the cursor is defined. Hence, if the picture's viewport is identical to that of the cursor (namely, the viewport in effect when the CURSOR subroutine was called - usually the full screen), then a picture element which appears near the cursor will be hit. If the picture occupies a viewport other than the full scope, the cursor should also be confined to that viewport if hit testing is to be performed.

Hit testing may be performed on lines, dots, or the origin of a character string, but not the characters themselves, since they are generated by the Character Generator after the clipping process.

The HITEST subroutine is called (normally with ISTAT=0) to determine whether any data has been "hit" since the "hit window" has been specified or since the last call to the HITEST subroutine. This allows the user control over the level to which hit testing is performed; i.e. groups of data sets may be tested at once, or a single data element can be individually tested merely by the placement of the call to the HITEST subroutine. This subroutine is also called at the completion of the "hit testing" process with ISTAT≠0 to restore the transformation which was in effect at the initiation of "hit testing" and to reset the Picture Processor so that all subsequent data drawn will be output to the Refresh Buffer.

The HITWIN and HITEST subroutines should be called in the following manner:

1. CALL the HITWIN subroutine to set the desired "hit window". This window will be centered at the x,y coordinate and of the size specified by the user. Typically, the x,y coordinates are

those returned by the TABLET subroutine but may correspond to any values, dynamic or otherwise.

2. Draw each data element, or data set, for which "hit testinq" is to be performed (the data is not actually drawn but is processed for hit testinq purposes only).
3. Determine whether a "hit" has been made upon the data element or data set by callinq the HITEST subroutine and testinq the IHIT parameter whose value will be returned:
   =0 if no hit occurred.
   =1 if a hit has occurred since the initial call to HITWIN or the last call to HITEST.
4. Steps 2 and 3 may be repeated as required to determine the data element or data set which was "hit". The final call to HITEST should have the second arqument, ISTAT, set to a non-zero value to restore the transformation in effect when the HITWIN subroutine was called and to allow subsequent data drawn to be written into the Refresh Buffer.

The previous steps (1-4) specify the manner in which hit testinq should be performed using the HITWIN and HITEST subroutines. The user should note that the transformations performed upon the data when it is displayed must also be performed upon the data when "hit testinq". These transformations include WINDOWinq, ROTation, TRANslation, SCAL(E)ing and INSTancing. This simplifies the "hit testinq" process since it may be done within the logical flow of the program, while (or as if) a new frame is beinq created. Example 5.11-9 illustrates how "hit testinq" may be used to determine if an object, in this case a "HOUSE" has been hit.

```
C
C    INITIALIZE THE PICTURE SYSTEM AND TURN ON
C    AUTOMATIC TABLET AND CURSOR
C
         CALL PSINIT(3,0,,,,)
         CALL TABLET (1,IX,IY,IPEN)
         CALL CURSOR (IX,IY,1,IPEN)
C
C    SET THE PERSPECTIVE WINDOW
C
         CALL WINDOW(-5000,5000,0,10000,0,10000,-20000)
C
C    BEGIN THE DISPLAY LOOP BY PERFORMING THE
C    TRANSFORMATIONS
C
100      CALL PUSH
         CALL TRAN (ITX,ITY,ITZ)
         CALL ROT  (IANGLZ,3)
         CALL ROT  (IANGLY,2)
         CALL ROT  (IANGLX,1)
C
C    IS THE PEN DOWN?  IF-SO BEGIN HIT TESTING
C
         I=ISPDWN(IPEN)
         IF (I.NE.0) CALL HITWIN (IX,IY,1000)
C
C    CALL THE SUBROUTINE WHICH DRAWS THE OBJECT
C
         CALL HOUSE
C
C    IF NOT HIT TESTING PROCEED WITH THE DISPLAY
C    LOOP, OTHERWISE...
C
         IF(I.EQ.0) GO TO 200
C
C    HIT TESTING... WAS BEGIN PERFORMED
C
         CALL HITEST (J,1)
         IF(J.EQ.0) GO TO 200
C
C    IT WAS HIT, UPDATE THE VALUES ACCORDINGLY
           .
           .
           .
         IPEN=0
C
C    CONTINUE THE DISPLAY LOOP
C
200      CONTINUE
           .
           .
           .
```

Example 5.11-9

5.11.3 Positioning

The Tablet is a natural positioning device, since the current x,y coordinates of the pen may be read at any time, and when the TABLET subroutine is used in automatic mode the most recently read pen coordinates are available at all times without specifically calling the TABLET subroutine. The x,y coordinates of the pen which are returned by the TABLET subroutine are in the range ±32700, a direct relation with the range of the data space values. This allows the user to directly use the pen coordinates to position data elements within the data space performing such functions as; line endpoint positioning, dragging, inking and rubber-band lines, with a minimal amount of software effort.

REFERENCES

1. "Principles of Interactive Computer Graphics"
Newman and Sproull, McGraw-Hill, 1973

2. "Matrices"
F. Ayers, McGraw-Hill, 1967

3. "The DOS/Batch Handbook", DEC-11-ODBHA-A-D
Digital Equipment Corporation, 1974

4. "RT-11 FORTRAN Compiler and Object Time System
User's Manual", DEC-11-LRFPA-A-D
Digital Equipment Corporation, 1974

5. "RT-11 System Reference Manual", DEC-11-ORUGA-A-D
Digital Equipment Corporation, 1973

APPENDIX A

SPECIFICATIONS OF THE PICTURE SYSTEM

This appendix includes the Functional Specifications for THE PICTURE SYSTEM as well as the detailed Hardware Specifications for the Picture Processor. The Functional Specifications provide the performance statistics describing the capabilities of THE PICTURE SYSTEM as a general purpose graphics system. The Picture Processor Hardware Specifications provide the interfacing, command and data details required to utilize the hardware at a systems level.

A.1 THE PICTURE SYSTEM FUNCTIONAL SPECIFICATIONS

The following describes the functional specifications of THE PICTURE SYSTEM. These specifications detail the capabilities of each of the components of the system: the Picture Controller, Picture Processor, Refresh Buffer, Character Generator, Picture Generator, Picture Display and Tablet.

A.1.1 Picture Controller

General Functions	- Contains the data base. - Executes the display programs. - Performs input/output operations.
Computer[1]	- Any DEC PDP-11 Family Computer.
Word Size	- 16 Bit.
Dimension Modes	- THE PICTURE SYSTEM displays two- and three-dimensional objects. - Two-dimensional data requires two words of Picture Controller memory to store the x and y coordinate values of a point. - Three-dimensional data requires three words of Picture Controller memory to store the x, y, and z coordinate values of a point. - Homogeneous coordinate data representation can be used with THE PICTURE SYSTEM in order to provide a much larger effective dynamic range by scaling the normal two- and three-dimensional data.
Coordinate Specification Modes	- Absolute coordinates (A) used to define points which are a given displacement from the origin of the data space. - Relative coordinates (R) used to define points which are a given displacement from the previous set of coordinates. - Picture elements may be specified in any of the following sequences of coordinate point definitions: • A,A,A,A,... • A,R,R,R,... • R,R,R,R,...
Drawing Modes	- The Move mode (M) moves the beam position to a specified location with the beam intensity off.

---

[1] THE PICTURE SYSTEM may be interfaced to any PDP-11 Family Computer. PICTURE SYSTEMS have been interfaced to PDP-11/05, PDP-11/35 and PDP-11/45 computers with various standard DEC peripherals including disks, DECtapes, magtapes, printers, etc.

- The Draw To mode (DT) draws a straight line from the current beam position to a new specified location and leaves the beam position at a new location.
- The Dot mode (D) moves the beam position to a specified location with the beam intensity off and then intensifies the beam at that specified location. The beam position remains at the dot location.
- The Character mode (C) draws the specified character beginning at the current beam position and then moves the beam position with intensity off, to the position where the next character in a string begins.
- Picture elements may be drawn using any of the above modes one by one or they may be drawn using any of the following sequences of the above modes:
  - M,DT,M,DT,... (unconnected lines)
  - M,DT,DT,DT,... (lines connected end-to-end)
  - DT,M,DT,M,... (another mode sequence for unconnected lines)
  - DT,DT,DT,DT,... (another mode sequence for lines connected end-to-end)
  - D,D,D,D,... (a series of dots)
  - C,C,C,C,... (a string of characters)

Instancing

- A method of defining in the data base a two- or three-dimensional structure once and replicating it several times in a picture in different positions, sizes and orientations.
- Instancing may be performed to any level.

Parameter Load/Store

- The Picture Controller can load and store all control registers, status registers and matrix registers that reside in the other components of THE PICTURE SYSTEM.

A.1.2 Picture Processor

General

- The Picture Processor operations are implemented in digital hardware.

Transformations

- Translates objects in any direction in three space.
- Rotates objects about any axis in three space.
- Scales objects with respect to any of the dimensions in three space.
- Perspective transformations can be performed on data passed to the Picture Processor.
- The Transformation Matrix is expressed in homogeneous coordinates which allows much larger translational values than would otherwise be possible.
- Creates mirror images of objects about a plane.

Compound Transformations

- Multiplies transformation matrices together while maintaining full-word accuracy.
- The Transformation Matrix may be loaded from the data base or stored into the data base residing in the Picture Controller memory.
- There is a push-down stack for storing four full transformation matrices with provision for continuing the stack in the Picture Controller memory.

Clipping

- Extracts the portions of the objects, defined in the data base, that are within a program-specified field of view.
- In two dimensions, the field of view is a program-specified rectangular region of the data space.
- In three dimensions, the field of view is a pyramid or frustrum (truncated pyramid) in the data space whose apex is at the eye.
- Clipping is performed with respect to the program-controlled six surfaces of the frustrum.

Perspective

- Displays realistic line representations of three-dimensional objects as they appear to the eye with reference to relative distance or depth.

Viewport	- The viewport specification is under program control and defines a six surface region of the Picture Display where the picture is to appear. Data which has been transformed, clipped, and put in perspective is linearly mapped into the viewport which allows complete separation of the coordinate systems of the drawing space and the Picture Display. - The resolution of the data mapped into the viewport is 16 bits, which allows this data to be used for precision plots. - Multiple viewports may be defined for a given frame to give simultaneous use of several areas of the screen. - Specification of viewport front and back provides the intensity bounds for depth-cueing.
Zooming	- The Picture Processor allows for moving smoothly and quickly into (or out of) a complex data structure in order to obtain a more detailed (or wide angle) view of a chosen region in the drawing space.
Hit Test	- THE PICTURE SYSTEM can detect whether any part of a given picture element is within a program-specified region in the data space or on the Picture Display. Hit Test is used for implementing the pointing function with a data tablet, eliminating the need for a light pen.
Memory Write Back	- Under program control, transformed digital data can be written back into the Picture Controller's memory to drive a hard copy plotter, for example, or as data for further computation.

A.1.3 Refresh Buffer

General Function	- The Refresh Buffer is for storing processed digital frame data allowing complete separation of Picture Display refresh requirements from the dynamic picture update requirements.
Data Content	- Dots and line endpoint data for use by the Picture Generator (one complete dot or line endpoint defination per buffer entry containing 12 bits for each of the x and y coordinate values and 8 bits for the intensity value). - Packed character codes for use by the Character Generator (up to three codes per buffer entry). - Status information used to control the displaying of the data.
Buffering	- Program-selectable single or double buffering is standard.
Cursor	- A dynamic cursor can be maintained regardless of the frame update rate.
Size[1]	- In single buffer mode, up to 8188 dots, line endpoints, or character code entries can be stored in the buffer in any combination. - In double buffer mode, up to 4092 dots, line endpoints, or character code entries can be stored in the buffer in any combination.

[1]The Standard Refresh Buffer is 8K, 36 bit words. An additional 8K of Refresh Memory may be obtained to provide a 16K Refresh Buffer.

A.1.4 Character Generator

General Function	- Accepts character codes and produces properly sized digital character stroking data for the Picture Generator.
Character Set	- Ninety-six character extended ASCII character set.
Sizes	- There are 8 character sizes available under program control ranging from 0.07 inches high in increments of 0.07 inches to 0.56 inches high on the Picture Display. The character width is also under program control with 8 different widths selectable for each size.
Character Orientation	- Horizontal 90° counter-clockwise orientation.
Capacity	- A maximum of 1725 characters can be displayed at a refresh rate of 30 frames per second.

A.1.5 Picture Generator and Picture Display

General Function	- Converts digital coordinate and intensity information to analog voltages to drive an electron beam across a phosphor-coated surface.
Line Modes	- <u>Solid</u>. - <u>Blink</u> mode allows selected picture elements to blink on and off. - <u>Dash</u> mode allows selected lines of a picture to be dashed.
Intensity Modes	- <u>Constant</u> intensity of program-selected picture elements may be chosen from 256 levels. Lines are drawn at a constant rate which assures uniform brightness for the chosen intensity level. - <u>Depth-cueing</u> allows the intensity of lines to vary continuously with depth (i.e., the z coordinate of the display).
Intensity and Contrast Controls	- In order to present a uniform variation in brightness, the intensity control of the Picture Display treats the z coordinate data as the logarithm of the intensity to be shown on the display. - The contrast control of the Picture Display is completely independent of the intensity control.
Refresh Control	- The refresh cycle is controlled by synchronization with the power line.
Display Rates	- <u>Move time</u> (for an n" move) $\leq .48 \times n + 2.0$ usec for $n \geq 2"$ < 3.0 usec for $n < 1/2"$ - <u>Draw Time</u> (for an n" line) $\leq 1.85 \times n + 2.0$ usec for $n \geq 1/2"$ < 3.0 usec for $n < 1/2"$ - <u>Dot Time</u> (for dots spaced n" apart) $\leq .6 \times n + 4.85$ usec for $n \geq 1/2"$ < 5.15 usec for $< 1/2"$ - Appoximate display capacities at 30 frames per second refresh rate: • 11500 connected 1/2" lines • 1625 connected 10" lines • 6650 dots 1/4" apart • 1725 characters .14" high (average) • 1500 characters .56" high (average)
Display Type	- Calligraphic.

Deflection Type	- Electromagnetic
Spot Size	- 0.020 inch.
Addressable Locations	- 4096 x 4096.
Endpoint Matching	- 0.020 inch.
CRT Size	- 21" rectangular, 10" x 10" quality viewing area.
Phosphor	- P4.

A.1.6 Tablet

General	- General purpose interactive input device.
Output	- 11 bits of x, 11 bits of y, and pen up/down status.
Resolution	- Digital: 11 bits for both x and y. - Graphic: 100 lines per inch.
Sampling Rate	- Variable up to 200 samples per second.
Size	- 11" x 11" useful area.
Cursor	- The cursor location on the Picture Display may be made to correspond to the stylus pen position on the tablet.

A.1.7 PDP-11 UNIBUS Addresses Reserved for the Picture System

The Standard PICTURE SYSTEM reserves the PDP-11 UNIBUS addresses summarized by Table A-1, for interfacing to the Picture Processor and the various PICTURE SYSTEM peripherals available.

Table A-1

Device	Reserved Unibus Addresses$_8$	Interrupt Vector$_8$
Picture Processor	767770 - 767776	300,304
Lorgnette	767760 - 767766	310
Keyboard	767750 - 767756	324
Switches & Lights #1	767740 - 767746	none
Switches & Lights #2	767730 - 767736	none
Switches & Lights #3	767720 - 767726	none
Tablet	777730 - 777736	330
Programmable Maintenance Panel	767700 - 767706	none
DR11-B (Picture Processor)	772410 - 772416	124

A.2 THE PICTURE PROCESSOR HARDWARE SPECIFICATIONS

The following describes the PDP-11/Picture Processor interface registers used to communicate the commands and data to and from the Picture Processor and the internal register structure and functions performed by the Picture Processor.

A.2.1 PDP-11 Picture Processor Interface Registers

This section describes the PDP-11 UNIBUS addressable registers that comprise the command and data interface paths between the PDP-11 and the Picture Processor. They are functionally divided into three classifications:

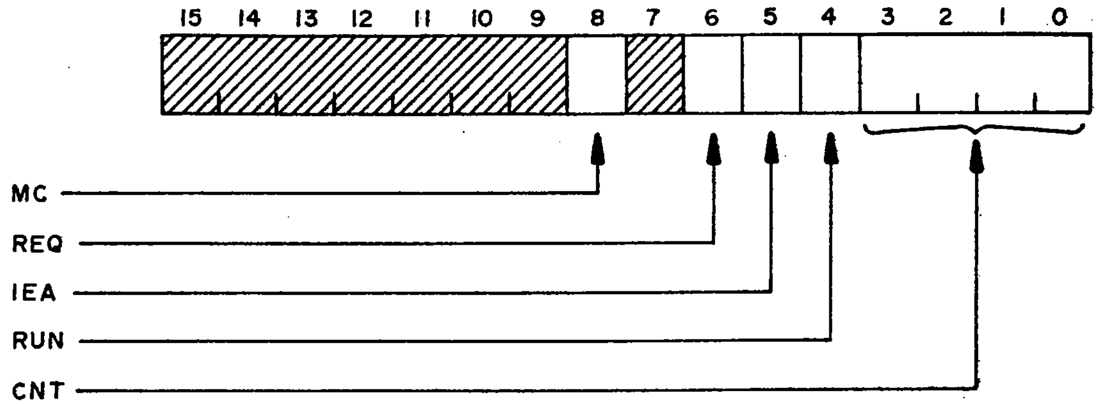
1. Refresh Timing Register
2. Command Registers
3. Data Transfer Registers

Table A-2 lists these registers and the interrupt vectors which are associated with them. The sections that follow give detailed descriptions of the functions of the registers and the bits within them.

TABLE A-2

NAME	SYMBOL	UNIBUS ADDRESS$_8$	INTERRUPT VECTOR$_8$
Real Time Clock	RTC	767770	300
Status Register	SR	767772	none
Repeat Status Register	RSR	767774	none
Word Count Register	DRWC	772410	none
Bus Address Register	DRBA	772412	none
DMA Status Register	DRST	772414	124

A.2.1.1 Refresh Timing Register (RTC): 767770_8



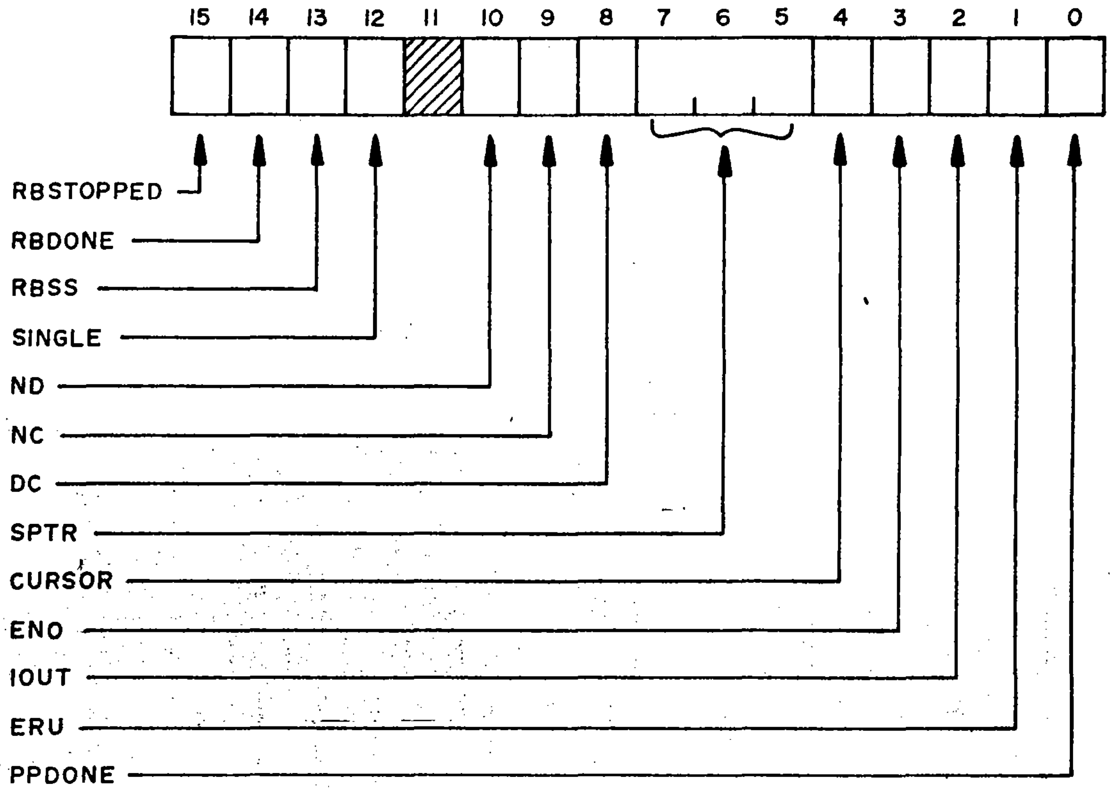
The RTC provides a mechanism for interrupting the PDP-11 at intervals which are programmable multiples of 1/120 second. (The PDP-11 Line Frequency Clock interrupts at 1/60th second).

BIT	NAME	FUNCTION
15-9	Unassigned	
8	Master Clear (MC)	When set causes a pulse that resets the Picture Processor and Picture Generator to their initial state. This provides a mechanism for initializing the Picture Processor without executing a RESET command. This bit always reads as zero.
7	Unassigned	
6	Request interrupt (REQ)	Set every n/120 seconds where n is the two's complement of the Count field (see bits 3-0), if bit 4 is set. This bit must be cleared by the interrupt routine to acknowledge interrupt service.

5	Interrupt Enable (IEA)	Set to allow REQ (bit 6) to cause an interrupt.
4	Run (RUN)	Set to allow REQ to be set.
3-0	Count 3,2,1,0	Four bit field loaded with two's complement of the number (n) of 1/120 second intervals that are desired to elapse before REQ is set. Bit 3 is the MSB. Clearing all bits results in n=16.

A.2.1.2 Command Registers

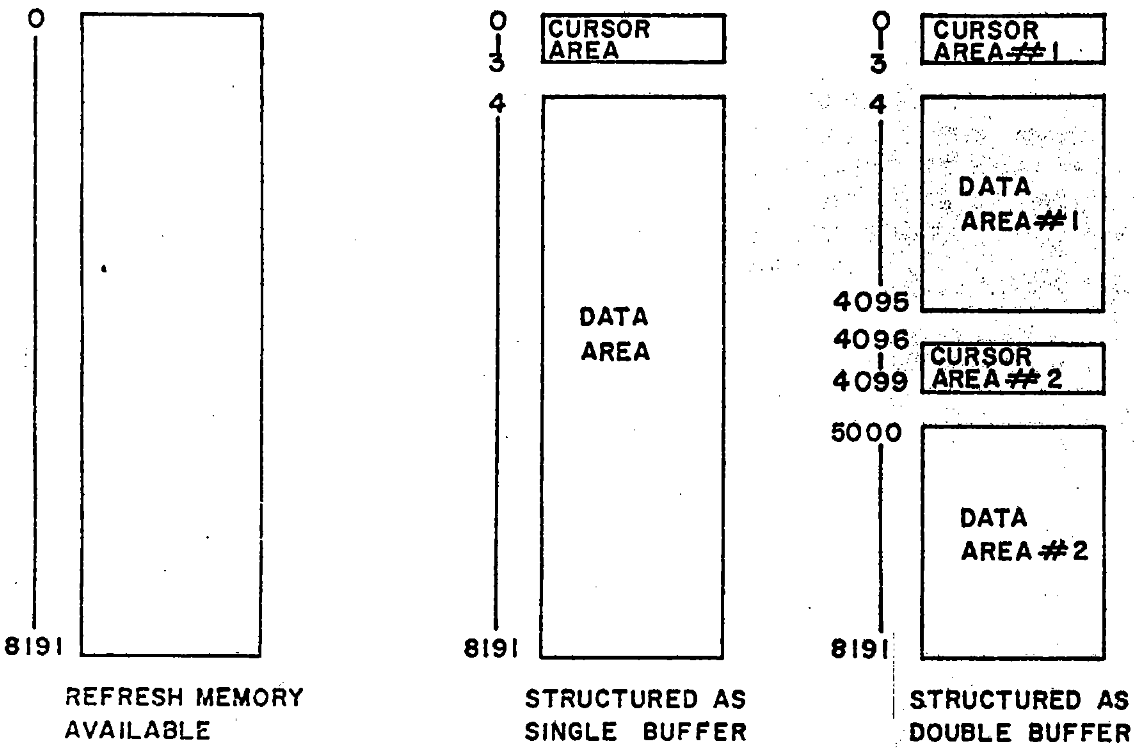
a. Status Register (SR): 767772_8



The Status Register is used to provide global operating mode information, such as single or double buffer, to the Picture Processor, and also to initiate the display refresh process in the Refresh Buffer.

BIT	NAME	FUNCTION
15	RBSTOPPED	Set by the Refresh Buffer Control when the refresh process stops. Clearing this bit causes the refresh process to start. Set by INIT.
14	RBDONE	If this bit is set when RBSTOPPED is set it indicates that the refresh process has stopped because the end of the current refresh data has

		been reached. If it is not set it means the refresh process has stopped because a "Status Halt" (see Section A.2.1.4) was encountered by the Refresh Buffer control, or that bit 13, RBSS, is set. Set by INIT. (Read only bit.)
13	RB Single Step (RBSS)	When this bit is set the Refresh Buffer control will stop the refresh process after each read access (RBSTOPPED will be set). This is a diagnostic mode that enables the data accessed by the refresh process to be read back to the PDP-11 using a STORE command. Cleared by INIT.
12	SINGLE Buffer	If this bit is set, the Refresh Buffer is configured as a single buffer with a four word cursor area and an 8188 word data area. If it is clear, the Refresh Buffer functions as a double buffer with two four-word cursor areas and two 4092 word data areas. Cleared by INIT. See Figure A-1.
11	Unassigned	
12,9,8	New Data (ND), New Cursor (NC), Display Cursor (DC)	These bits are used to indicate special actions to be taken by the Refresh Buffer control when it begins a refresh operation. These bits are only sampled by the Picture Processor at the beginning of the refresh cycle (i.e. whenever RBSTOPPED is cleared). Their functions are as follows:



NOTE: THE REFRESH BUFFER MAY OPTIONALLY BE EXPANDED TO TWICE ITS STANDARD SIZE TO PROVIDE FOR A REFRESH BUFFER OF 16384 WORDS.

Figure A-1

Refresh Memory Structure

In Double Buffer Mode:
ND - Causes the area of the Refresh Buffer currently assigned as the "write" data area to become the "read" data area, and assigns the current "read" data area as the new "write" area. This prepares for the display of the data which was just written into the Refresh Buffer.

NC - Causes the area of the Refresh Buffer currently assigned as the "write" cursor area to become the "read" cursor area, and assigns the current "read" cursor area to be the new "write" cursor area. Subsequent refresh cycles that display a cursor will refresh from the new "read" cursor area. The refresh Buffer "write" cursor area addressing mechanism is initialized to point to the beginning of the assigned area.

DC - If this bit is set when a refresh cycle is started, the current "read" cursor area contents will be displayed in addition to the contents of the current "read" data area.

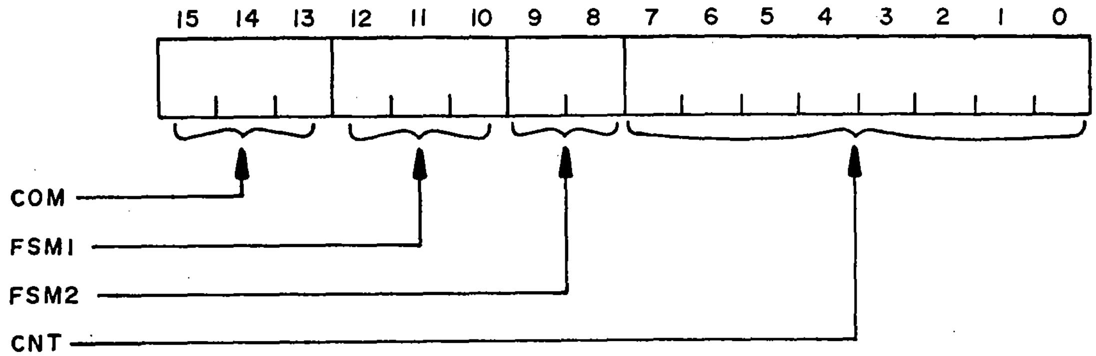
In Single Buffer:
ND - Causes the Refresh Buffer "write" data addressing mechanism to reset to the beginning of the data area, and causes the subsequent refresh cycle to start reading from the beginning of the data area.

		NC - Causes the Refresh Buffer "write" cursor area addressing mechanism to be initialized to point to the beginning of the cursor area. DC - (same as double buffer). ND, NC, and DC are cleared by INIT.
7,6,5	Stack Pointer (SPTR)	These bits are used to address the currently available matrix area on the Matrix Stack (see Section A.2.2). This field is automatically incremented after a PUSH operation and decremented before a POP operation. If the stack is (SPTR=4) and a PUSH is attempted, or empty (SPTR=0) and a POP is attempted the STACK ERROR bit of the DMA Status Register (see Section A.2.1.4) will be set and the operation will not be performed. Cleared by INIT.
4	CURSOR	When this bit is set, any data that is normally written in the Refresh Buffer will be written in the currently assigned "write" cursor area. If more than four draw commands which result in data being written into the cursor area are executed, then the previous contents of the cursor area will be overwritten.
3	Enable No Overlap (ENO)	If this bit is set, the ATTENTION bit of the DMA Status Register (see Section A.2.1.4) will be set each time a DRAW2D or DRAW3D command is executed by the Picture Processor and if the RSR Coordinate Count (see

		Section A.2.2b) is such that the command would normally be repeated, the subsequent command execution will be inhibited until bit 0 of the DMA Status Register is cleared. This bit is useful in "hit testing" to determine which draw command resulted in the "hit". Cleared by INIT.
2	Inhibit Output (IOUT)	When this bit is set, it will prevent any data from being written into the Refresh Buffer. This bit is also useful in "hit testing" where data passing through a hit window would appear misplaced if displayed. Cleared by INIT.
1	Enable RSR Update (ERU)	If this bit is set, the Coordinate Count field of the Repeat Status Register (see Section b below) equals negative-one (all 1's), the Picture Processor will automatically fetch new contents for the RSR via the DMA data path at the end of the current command execution. Once the new RSR has been fetched, it will then be treated as a new command and a new command execution will automatically take place. Cleared by INIT.
0	Picture Processor Done (PPDONE)	When this bit is set it means that the Picture Processor is waiting for input and has no processing pending (it is not an indication of the state of the Refresh Buffer). The contents of the SR and RSR registers should not be modified until this bit is set. The "hit" bits of the DRST (see Section A.2.1.4) should not be examined until this bit is set, nor should data being

written into the memory of the PDP-11 by a STORE command. Set by INIT. (Read only.)

b. Repeat Status Register (RSR): 767774_8



The Repeat Status Register is used to supply commands to the Picture Processor. It is called a "Repeat" Status Register because portions of its contents may be automatically modified in a predetermined manner after the specified command has been executed, and in certain cases the command is repeated after the modification, without required program intervention.

The bits of the RSR are divided into 3 fields:

1. Command - specifies what command type is to be executed. This field is never automatically modified.
2. Finite State Machine - these bits give further specification of the command to be executed. This field is automatically updated after each execution.
3. Coordinate Count - these bits determine how many times the command should be executed before program intervention is required. If the field contains a negative number, it is automatically incremented after each command execution.

BIT	NAME	FUNCTION
15,14,13	Command Bits (COM)	These bits define the command type to be executed. The bit combin-atiions and interpreta-tion are as follows:

000-2DDRAW	Two words are accessed from the PDP-11 via the data transfer mechanism, and are then processed

by the Picture Processor, as specified by the Finite State Machine bits. Note that if the Finite State Machine bits specify CHARACTER or STATUS, then three words are accessed from the PDP-11.

001-3DDRAW	Three words are accessed from the PDP-11 via the data transfer mechanism, and are then processed by the Picture Processor as specified by the Finite State Machine bits.
010-PUSH	The current contents of the Transformation Matrix (see Section A.2.2.1) are placed into the currently available element of the Matrix Stack and the SPTR is incremented. If the SPTR is 4, the execution does not take place and the STKRR bit of the DMA Status Register is set.
011-MATCON	Matrix Concatenation - Four words are accessed from the PDP-11 via the data transfer path. These four words are treated as a row (0-3) of a matrix that is beinq post multiplied by the Transformation Matrix in the Picture Processor. The row is specified by the Finite State Machine (FSM) bits. The resultinq row is placed in the Temporary Matrix (see Section A.2.2) of the Picture Processor. If the row specified is row 3 then at the end of the post multiplication

		process the matrix stored in the Temporary Matrix is normalized and placed in the Transformation Matrix, destroying the old contents of the Transformation Matrix.
	100-POP	The last matrix PUSHed is returned to the Transformation Matrix and the SPTR is decremented. If the SPTR is 0 the execution does not take place and the STKRR bit of the DMA Status Register is set.
	101-LOAD	Four words are accessed from the PDP-11 via the data transfer path and placed in the Picture Processor Register (see Section A.2.2) specified by the FSM bits.
	110-STORE	The four words which represent the contents of the Picture Processor Register specified by the FSM bits are sent to PDP-11 via the data transfer path.
	111-NO OP	This command is treated by the Picture Processor as a NO OP. It requires that FSM1=7.
12,11,10	Finite State Machine 1 (FSM1)	These five bits comprise the fields known as the Finite State Machines. The way they are interpreted is a function of the Command bit.
9,8	Finite State Machine 2 (FSM2)	
		<u>PUSH,POP</u> No effect on operation. <u>LOAD,STORE</u>: the given bits are interpreted as a single field

containing an octal address (0-27) specifying which Picture Processor Register is to be manipulated. The address is incremented at the completion of the command.

MATCON: The three FSM1 bits must be zeros. The FSM2 bits represent an address (0-3) that specifies which row of the matrix to be concatenated with the Transformation Matrix is currently being sent. The address is incremented at the completion of the command.

2DDRAW, 3DDRAW: The FSM1 bits are used to describe the type DRAW that is desired. At the end of the command the bits are updated. The type, update definitions and the FSM1 sequences initiated are listed below:

FSM1 OCTAL VALUE	TYPE	VALUE AFTER UPDATE	SEQUENCES
0	MOVETO(M)	1	(M,D,M,D,...)
1	DRAWTO(D)	0	(D,M,D,M,...)
2	MOVETO(M)	3	(M,D,D,D,...)
3	DRAWTO(D)	3	(D,D,D,D,...)
4	DOT(DOT)	4	(DOT,DOT,...)
5	STATUS(S)	5	(S,S,S,S,...)
6	not used	7	
7	CHARACTER(C)	7	(C,C,C,C,...)

The interpretations of the various types of FSM1 are as follows:

MOVETO: specifies a point in the coordinate system, normally used as the beginning point of a line.

DRAWTO: indicates that a line is to be drawn from the last specified point to the point being specified.

DOT: Specifies that a dot is to be drawn at the point specified.

STATUS: causes a word to be written into the Refresh Buffer that consists of parts of the three PDP-11 words that must accompany the command (see Section A.2.1.4d). These words are used to change the status of the Picture Generator, the Character Generator or Color within the definition of a picture.

CHARACTER: causes a word to be written into the Refresh Buffer that contains the three ASCII codes specified by the three PDP-11 words that must accompany the command (one ASCII code, right justified, per word). When the refresh process encounters the word in the Refresh Buffer, it accesses the Character Generator and draws the specified characters before making any subsequent accesses from the Refresh Buffer.

NOTE: See Section A.2.1.4d which follows, for the data formats of the DRAW Commands.

The FSM2 bits are used to specify whether the data accompanying DRAW commands is to be interpreted as absolute or relative coordinate data (added to previous data). Note that STATUS and CHARACTER commands always imply absolute data. At the completion of a command execution the bits are updated. The interpretation, update definitions and the FSM2 sequences initiated are listed below:

FSM2 OCTAL VALUE	INTERPRETATION	VALUE AFTER UPDATE	SEQUENCES
0	ABSOLUTE(A)	1	(A,R,R,...)
1	RELATIVE(R)	1	(R,R,R,...)
2	ABSOLUTE(A)	2	(A,A,A,...)
3	not used	3	

7-0	Coordinate Count (CNT)	This 8-bit field is used to specify how many times the command specified by the COM bits is to be repeatedly executed (with additional data each time). The field is treated as a two's complement number, with bit 7 being the sign bit.

largest count is 128

If the value of the number is positive (bit7=0) the command is executed once and the 8-bit number is not incremented at the completion of the execution. If the value of the number is negative then the command is executed repeatedly, with the number incremented at the end of each command execution, until it goes positive (all 1's to all 0's).

If, at the time the coordinate count increments from -1 to 0, the Enable RSR Update bit (bit 2) of the Status Register is set, then a word will automatically ~~be accessed from the~~ PDP-11 via ~~the data transfer~~ path and placed in ~~the~~ RSR. The new contents of the RSR will then be interpreted and a command execution initiated automatically.

new "instruction" fetched automatically

A.2.1.3 Command Execution

The registers described above merely serve to specify the command to be executed. To initiate execution a bit in one of the data transfer registers must be manipulated. This bit is bit 0 of the DRST register, as described in the following section.

A.2.1.4 Data Transfer Registers

The preceding section dealt with the UNIBUS registers of the Picture Processor that are used to pass command information. This section deals with the mechanism that is used to pass data between the PDP-11 and Picture Processor. Note that data can be transferred in either direction.

The data transfer path is a DEC DR11-B, Direct Memory Access Interface unit. To pass data to or from the Picture Processor, a block of PDP-11 memory which contains the data, or which will receive the data, is specified by loading registers in the DR11-B. The Picture Processor, then, in its normal course of executing commands specified by the RSR requests the DR11-B to access the memory locations specified.

a. Word Count Register (DRWC): 772410_8

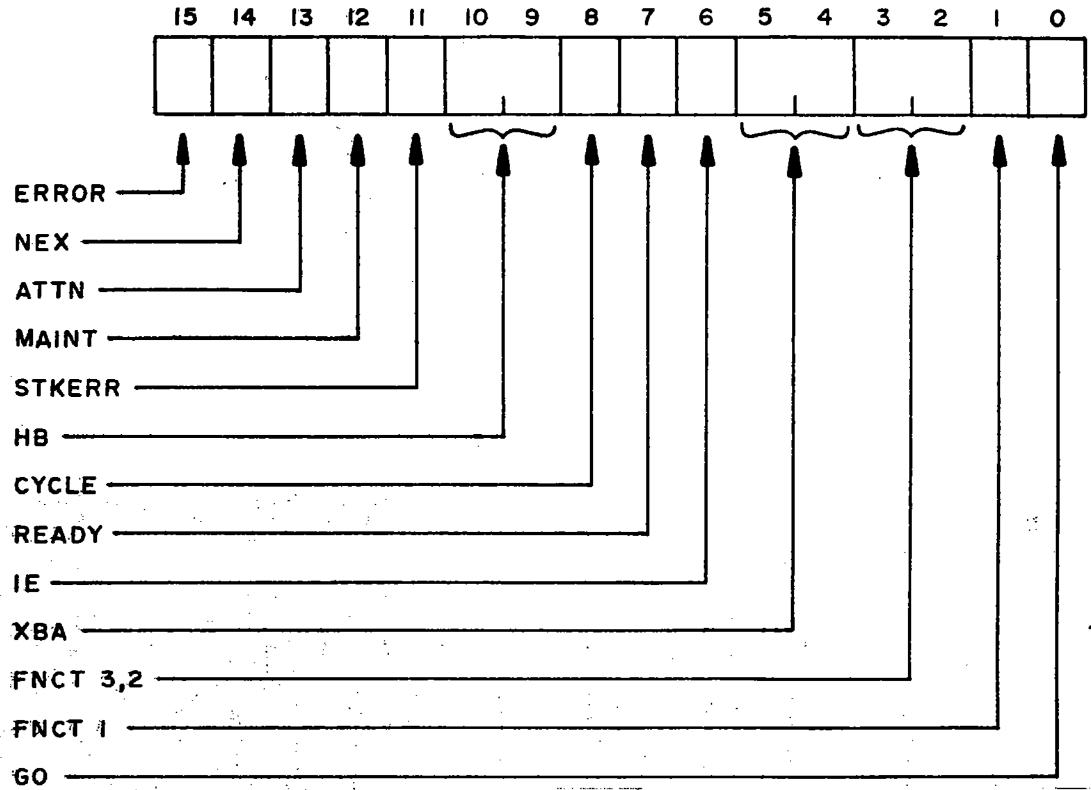
The DRWC is a 16-bit Read/Write register. It is initially loaded with the two's complement of the number of words to be transferred and increments up towards zero after each bus cycle. When overflow occurs (all 1's to all 0's), the READY bit of the DRST is set and bus cycles stop. DRWC is a word register, byte instructions should not be used when loading this register. This register is cleared by the UNIBUS INIT signal.

b. Bus Address Register (DRBA): 772412_8

The DRBA is a 15-bit R/W register. Bit 0 is always a zero, and is a read-only bit. Along with bits 5 and 4 of the DRST (XBA17 and XBA16), the DRBA is used to specify the address used when the DR11-B accesses the UNIBUS. The register is incremented (by 2) after each bus access, advancing the address to the next sequential word location on the bus. If DRBA overflows (177776 to 0) the ERROR bit in the DRST is set. This error condition is cleared by loading DRBA or by INIT. DRBA is a word register; byte instructions should not be used when loading this register. Cleared by INIT.

count of <u>all</u> data & RSR auto-load.

c. **DMA Status and Command Register (DRST): 772414_8**



This register is used to provide status indicators of the DR11-B, status indicators of the Picture Processor, and to provide a means for initiating execution of Picture Processor commands specified by the SR and RSR.

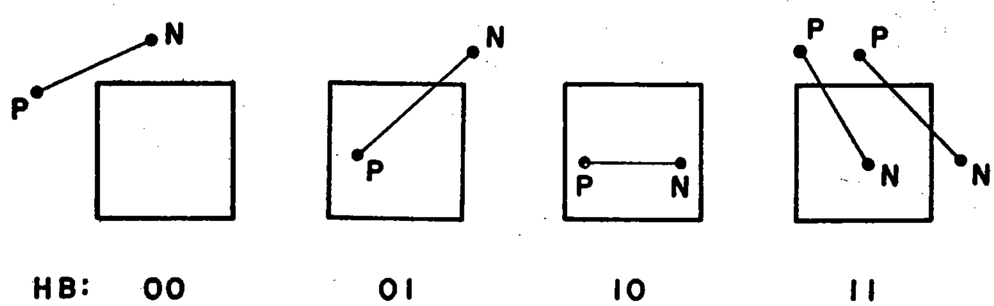
BIT	NAME	FUNCTION
15	ERROR	Set to indicate an error condition: either NEX (BIT 14), ATTN (BIT 13), interlock error (test board is neither in slots AB02 nor CD04), or bus address overflow (BAOF: DRBA incremented from 177776 to 0). Sets READY (BIT7) and causes interrupt if IE (BIT 6) is set. ERROR is cleared by removing all four possible error conditions; interlock error is removed by inserting test board in CD04 for diagnostic tests or in AB02 for normal opera-

		tion; bus address overflow is cleared by loading DRBA; NEX is cleared by loading bit 14 with a zero; ATTN is cleared by the method described below. Read only.
14	Nonexistent Memory (NEX)	Set to indicate that an UNIBUS master, the DR11-B did not receive a SSYN[1] response 20 usec after asserting MSYN[2]. Cleared by INIT or loading with a 0; cannot be loaded with a 1. Sets ERROR.
13	Attention (ATTN)	This bit is set by the picture Processor whenever PUSH or POP operation is executed or when a 2DDRAW or 3DDRAW command is executed and the ENO bit of the SR is set. (See Section A.2.1.2, bit 3.) The bit is read only, in the sense that it cannot be set or cleared by MOVing to DRST. It is cleared whenever GO (DRST bit 0) is set.
12	Maintenance (MAINT)	Maintenance bit used by diagnostic programs. Cleared by INIT, Read/Write.

----

[1]See PDP-11 Peripherals Handbook, for further UNIBUS signal details.
[2]Ibid.

11	Stack Error (STKRR)	If this bit is set when ATTN is set, it indicates that a Matrix Stack overflow or underflow has occurred. This read only bit is cleared by the same method used to clear ATTN.
10-9	Hit Bits (HB)	These bits are used to determine whether data that has just been processed during 2DDRAW or 3DDRAW command execution (except STATUS and CHARACTER) has been clipped. Once the bits have been set they can only be cleared by having FNCT1 set when GO is set. The values of the bits indicate something of the geometry of the data. The figures below show the geometries that result in the four possible combinations of these bits, where "N" is the most recent coordinate processed, and "p" is the one directly preceding it. The rectangle represents the clipping boundaries.



		For the bits to be meaningful they must be clear immediately before execution of the 2DDRAW or 3DDRAW command in question.
8	CYCLE	Cycle is used to prime bus cycles; if set when GO is issued, an immediate bus cycle occurs. Cleared when bus cycle begins; cleared by INIT.

		Read/Write.
7	READY	Set to indicate that the DR11-B is able to accept a new command. Set by INIT or ERROR; cleared by GO; set on word count overflow. Causes interrupt if bit 6 is set. Forces DR11-B to release control of the UNIBUS and prevents further DMA cycles. Read only.
6	Interrupt Enable (IE)	Set to allow ERROR or READY=1 to cause an interrupt. Cleared by INIT. Read/Write.
5-4	Extended Bus Address (XBA)	Extended bus address bit 17 and 16 that in conjunction with DRBA specify an 18 bit address to be used for direct memory transfers. Cleared by INIT. XBA17 and XBA16 do not increment when DRBA overflows; instead ERROR is set. Read/Write.
3-2	FNCT3,2	Unassigned, may be used as general Read/Write bits.
1	FNCT1	This bit is used to allow the clearing of DRST bits 10 and 9 (HB). The method for clearing these bits is to set this bit prior to setting bit 0 of the DRST. Read/Write.
0	GO	This bit is set to initiate the execution of a command by the Picture Processor. Setting this bit clears ATTN, if set, and if FNCT1 is set it also clears the HB bits. Note that the setting of this bit always causes a command execution by the Picture Processor.

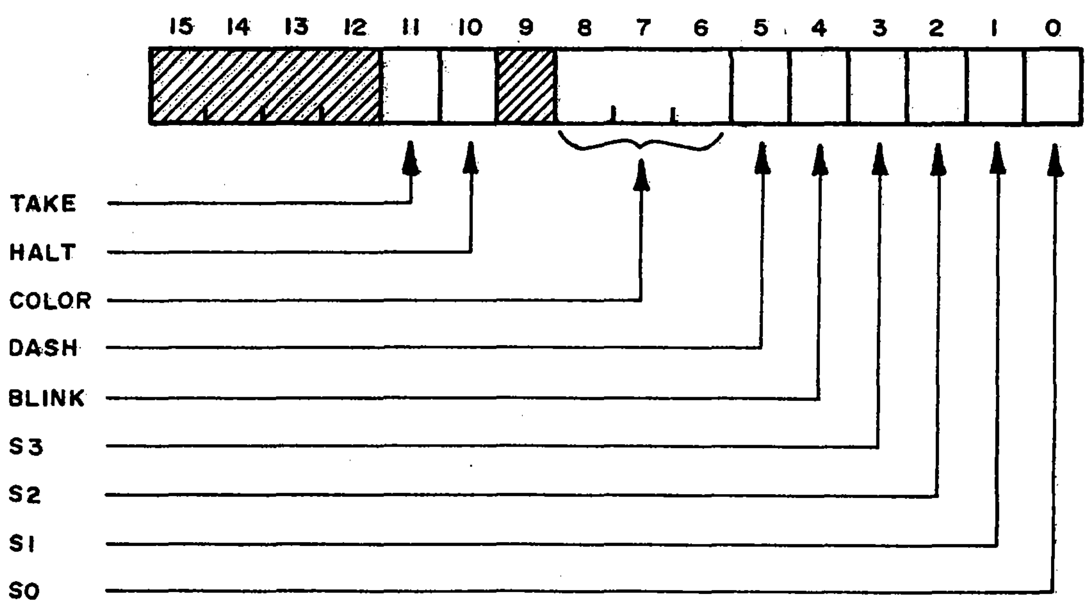
d. Data Formats

The preceding sections detailed the registers used in transferring data. This section details the formats of the data that accompany the various commands that can be specified by the RSR. For each case, the data required for one execution cycle is given.

additional data is required for each execution as specified by the CNT field of the RSR.

PUSH,POP:	No data necessary.
LOAD,STORE, MATCON:	4 PDP-11 words.
2DDRAW:	For all except STATUS AND CHARACTER, 2 PDP-11 words representing x and y coordinate values. For STATUS and CHARACTER: 3 PDP-11 words. See the data format for STATUS and CHARACTER in 3DDRAW.
3DDRAW:	For all except STATUS and CHARACTER: 3 PDP-11 words representing x, y and z coordinate values. STATUS: 3 words representing: 1. Picture Generator Status. 2. Character Generator Status. 3. Not used. CHARACTER: 3 words, each containing the right justified 7-bit ASCII code of the character desired.

Picture Generator Status Word:



This word accessed by a 2DDRAW or 3DDRAW command and a STATUS FSM1, is used to specify global information to the Picture Generator. the information specifies that color is to be displayed (for color monitor use only), whether to draw DASHed lines, put the Picture Generator in BLINK mode, what scopes should be selected, or whether to stop the Refresh process ("Status Halt").

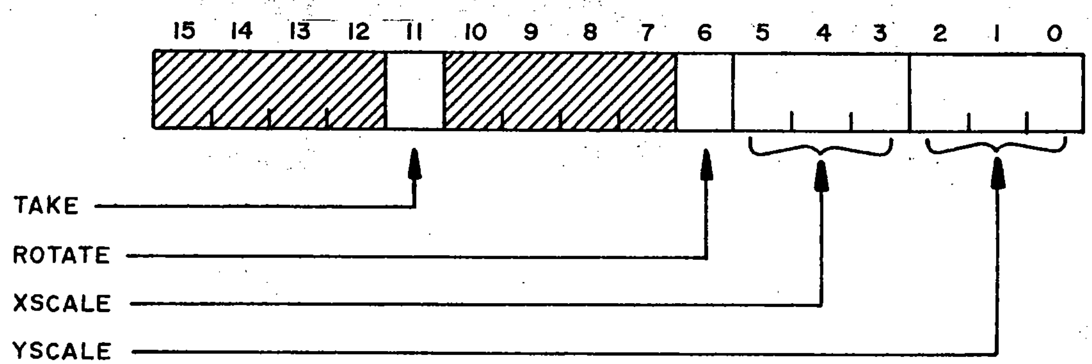
BIT	NAME	FUNCTION
15-12	Unused	
11	TAKE	This bit signals the Refresh Buffer control that bits 8-0 are valid and should be loaded in the Picture Generator Status Register. If it is not set, bits 8-0 are not interpreted.
10	HALT	If this bit is set the Refresh Buffer control will stop the refresh process (Status Halt).
9	Unused	
8-6	COLOR	These bits specify the color status for the scopes selected by bits 3-0. The octal value of these bits specifies the color of all subsequent data drawn. A value of 0, 1 or 2 must be used when a black and white display is selected. Values 3-7 are used when a beam penetration monitor is selected.

COLOR	Color Selected
0	(Black and White Display)
1	(Black and White Display)
2	(Black and White Display)
3	Red
4	Red/Orange
5	Orange
6	Yellow
7	Green

BIT	NAME	FUNCTION
5	DASH	Indicates that all succeeding

		lines and characters are to be drawn dashed.
4	BLINK	Indicates that all succeeding dots, lines, and characters are to blink on the display.
3	Scope 3 Select (S3)	Indicates that the Picture Display whose scope driver card is in Picture Generator backpanel slot 24 will display all data subsequently drawn.
2	Scope 2 Select (S2)	Same as bit 3, but for slot 22.
1	Scope 1 Select (S1)	Same as bit 3, but for slot 20.
0	Scope 0 Select (S0)	Same as bit 3, but for slot 18.

Character Generator Status Word:



This word, accessed by a 2DDRAW or 3DDRAW command and a STATUS FSM1 is used to specify rotation and scaling information to the Character Generator.

BIT	NAME	FUNCTION
15-12	Unused	
11	TAKE	This bit signals the Refresh Buffer control that bits 6-0 are valid and are to be loaded in the Character Generator Status Register. If it is not set bits 6-0 are not interpreted.

10-7	Unused	
6	ROTATE	If this bit is set, all subsequent characters drawn by the Character Generator will be rotated 90° in the counterclockwise direction.
5-3	XSCALE	This octal (0-7) number specifies the X size of all subsequent characters; 0 is the smallest, 7 the largest.

SCALE	APPROXIMATE SIZE (of Capital Letters)
0	0.07"
1	0.14"
2	0.21"
3	0.28"
4	0.35"
5	0.42"
6	0.49"
7	0.56"

2-0	YSCALE	Same as XSCALE, except specifying the Y size.

Figure A-2 illustrates the data formats for the 2DDRAW and 3DDRAW commands.

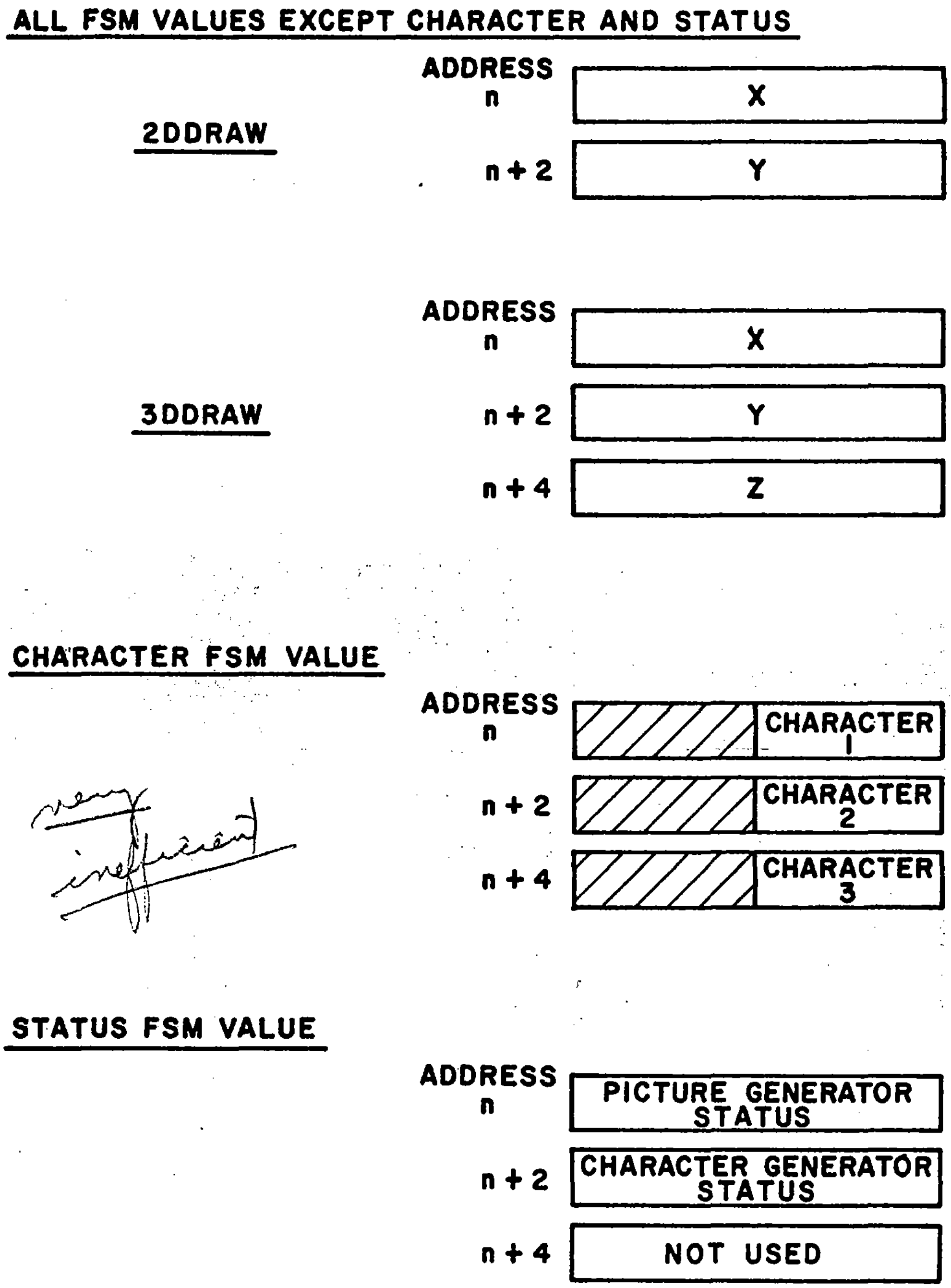


Figure A-2

Data Formats for 2DDRAW and 3DDRAW Commands

A.2.2 Picture Processor Internal Registers

The proceding sections describe the registers of the Picture Processor that may be accessed directly with PDP-11 instructions. This section describes the registers that are internal to the Picture Processor and used to contain parameters for various functions or as working storage during command execution. Figure A-3 shows these registers and the addresses assigned to them. These addresses are specified using the FSM fields for LOAD and STORE commands. Each register consists of four 16-bit elements. LOAD and STORE commands always refer to four-element registers.

A.2.2.1 Transformation Matrix (TRANMAT), Register $0\text{-}3_8$

These four registers are used to contain the 4x4 Transformation Matrix. This matrix is post-multiplied by the data processed during the execution of 2DDRAW or 3DDRAW commands (except STATUS and CHARACTER).

A.2.2.2 Temporary Matrix (TEMPMAT), Register $4\text{-}13_8$

These eight registers are used to store the temporary results during a MATCON operation. They are working registers of the Picture Processor and have addresses for diagnostic purposes only. They cannot be loaded with a LOAD command.

A.2.2.3 Refresh Buffer (REFBUF), Register 14_8

This read only register (cannot be LOADed) always contains the data last read from the Refresh Buffer. It is addressable for diagnostic purposes only.

A.2.2.4 Viewport Left, Bottom, Hither (VIEWL,VIEWB,VIEWH), Register 20_8

This register (in conjunction with register 24) is used to specify the boundaries to which data that lies within the clipping boundaries will be mapped by the viewport mapping process.

VIEWL is the left boundary
VIEWB is the bottom boundary
VIEWH is the hither boundary

The fourth component is not used and is undefined.

A.2.2.5 Save (SAVE), Register 21_8

This is a working register. At the completion of a 2DDRAW or 3DDRAW command execution (except STATUS and CHARACTER) this register contains the data as it

ADDRESS $_8$

0					TRANSFORMATION MATRIX
1					
2					
3					
4					TEMPORARY MATRIX
5					
6					
7					
10					
11					
12					
13					
14	REFBUFX	REFBUFY	REFBUFZ	REFBUFS	REFRESH BUFFER
15					NOT USED
16					
17					
20	VIEW L	VIEW B	VIEW H		VIEWPORT LEFT BOTTOM, HITHER
21	SAVE X	SAVE Y	SAVE Z	SAVE W	SAVE
22	NCX	NCY	NCZ	NCW	NEWCLIP
23	NVX	NVY	NVZ		NEWVIEW
24	VIEW R	VIEW T	VIEW Y		VIEWPORT RIGHT, TOP, YON
25	BASE X	BASE Y	BASE Z	BASE W	BASE
26	PCX	PCY	PCZ	PCW	PREVIOUS CLIP
27	PVX	PVY	PVZ		PREVIOUS VIEW

Figure A-3

Addressable Picture Processor Registers

exists after it has been multiplied by the Transformation Matrix, but prior to any clipping that may have taken place.

A.2.2.6 New Clip (NC), Register 22_8

This is a working register. The contents are only of interest at the completion of a 2DDRAW or 3DDRAW command execution, (except STATUS and CHARACTER) and then only if the data that accompanied the command resulted in data that would normally be passed to the Refresh Buffer and then the Picture Generator (i.e. a DOT or MOVETO that was within the clipping boundaries, or a DRAWTO that resulted in a line with some portion within the clipping boundaries). For all other cases the contents of this register are not defined. (Note that the status of the "hit bits" of the DRST are an indication of whether the data was within the clipping boundaries.) If the contents are valid, they represent the coordinate values of the data within the clipping boundaries. If clipping has occurred, they represent the results of the clipping computational process, either the original endpoint or the coordinates where the line intersected the clipping boundary.

A.2.2.7 New View (NV), Register 23_8

This is a working register. The contents are only of interest when the NC register has valid information stored in it. The contents represent the results of the viewport mapping process that performs the linear mapping and perspective division of the data in the NC, from the clipping boundaries to the viewport boundaries (specified by the viewport registers, 20 and 24).

Only three of the four elements contain valid information: NVX, NVY, NVZ. The fourth element is used strictly as a working register, and its contents are not defined.

It is the 12 least significant bits of NVX and NVY, and the 8 least significant bits of NVZ that are written in the Refresh Buffer and subsequently passed to the Picture Generator during the refreshing process.

A.2.2.8 Viewport Right, Top, Yon (VIEWR,VIEWT,VIEWY), Register 24_8

This register (in conjunction with register 20) is used to specify the boundaries to which data, that lies within the clipping boundaries, will be mapped by the viewport mapping process.

VIEWR is the right boundary
VIEWT is the top boundary
VIEWY is the yon boundary

The fourth component is not used and is undefined.

A.2.2.9 Base Register (BASE), Register 25_8

All 2DDRAW and 3DDRAW commands (except STATUS and CHARACTER) result in the Picture Processor performing computations on 4 data elements representing the drawing coordinates. The BASE register provides two functions. It supplies the fourth element, w, for 3DDRAW commands, and the third and fourth, z and w respectively, for 2DDRAW commands. The base register is also used as the absolute coordinates to which all relative details added to compute absolute coordinates when FSM2 specifies RELATIVE. The base register should always be LOADED with the necesssary values prior to executing 2DDRAW and 3DDRAW commands.

A.2.2.10 Previous Clip (PC), Register 26_8

This is a working register. Its contents are valid only at the completion of a 2DDRAW or 3DDRAW command whose FSM1 specified a DRAWTO, and whose execution resulted in a portion of the line being within the clipping boundaries, but the beginning of the line being outside the clipping boundaries (i.e. the most recent 2DDRAW or 3DDRAW whose FSM1 was MOVETO or DRAWTO was accompanied by data that was not within the clipping boundaries). For all other cases the contents of this register is not defined. (Note that the "hit bits" of the DRST are an indication of whether the above conditions are satisfied.) If the contents are valid, they represent a point computed by the clipping process that is interpreted as a MOVETO which specifies the beginning point of the portion of the line that lies within the clipping boundaries.

A.2.2.11 Previous View (PV), Register 27_8

This is a working register. The contents are only of interest when the PC register has valid information stored in it. The contents represent the results of the viewport mapping process that performs a linear mapping and perspective division of the PC from the clipping boundaries to the viewport boundaries (specified by the viewport registers, 20 and 24).

Only three of the four elements contain valid information: PVX,PVY,PVZ. The fourth element is used strictly as a working register, and its contents are not defined.

When the beginning point of a line has been clipped, it is the 12 least significant bits of PVX and PVY and the 8 least significant bits of PVZ that are written in the Refresh Buffer and subsequently passed to the Picture Generator during the refreshing process.

A.2.2.12 Matrix Stack

The Matrix Stack is a non-addressable (by LOAD or STORE commands) collection of registers that are used to temporarily store transformation matrices. It is a four level matrix stack and is accessed whenever a PUSH or POP command is executed.

A.2.3 Command Execution Details

This section details the flow of data within the Picture Processor internal register structure for each of the commands that can be specified by the RSR. In the case of 2DDRAW and 3DDRAW the operations that take place are treated step by step.

Notes on nomenclature:

1. IN represents the incoming data that accompanies the command.
2. A "→" refers to a four component (4 16-bit words) set of data.
3. Subscripts such as x, y, z and w refer to the individual 16-bit elements of the incoming data (IN) or internal registers.
4. The subscript i is used to specify an internal register address (i.e. REGi).
5. RB refers to the Refresh Buffer.
6. In the description of 2DDRAW and 3DDRAW, "new point" refers to the data accompanying the command, and "previous point" refers to the data that accompanied the most recent 2DDRAW or 3DDRAW. The term "line" refers to the vector that begins at the previous point and ends at the new point.

A.2.3.1 2DDRAW and 3DDRAW, FSM1 = DRAWTO

a. Input Data to Internal Registers.

1. If 2DDRAW and FSM2 = ABSOLUTE
   INx → PCx, BASEx
   INy → PCy, BASEy
   BASEz → PCz, BASEz
   BASEw → PCw, BASEw

2. If 2DDRAW and FSM2 = RELATIVE
   INx + BASEx → PCx, BASEx
   INy + BASEy → PCy, BASEy
   BASEz → PCz, BASEz
   BASEw → PCw, BASEw

3. If 3DDRAW and FSM2 = ABSOLUTE
   INx → PCx, BASEx
   INy → PCy, BASEy
   INz → PCz, BASEz
   BASEw → PCw, BASEw

4. If 3DDRAW and FSM2 = RELATIVE
   INx + BASEx → PCx, BASEx
   INy + BASEy → PCy, BASEy
   INz + BASEz → PCz, BASEz

BASEw ➡ PCw, BASEw

b. Transform Data.

$\overrightarrow{(PC)}$ x (TRANMAT) ➡ $\overrightarrow{SAVE}$, $\overrightarrow{NC}$

c. Clip

The clipping process determines whether the data lies within the clipping boundaries. In order for a point to be within these boundaries, it must satisfy the following requirements:

$$-SAVEw \leq SAVEx \leq SAVEw$$
$$-SAVEw \leq SAVEy \leq SAVEw$$
$$0 \leq SAVEz \leq SAVEz$$

1. If a portion of the line is inside the clipping boundaries:

   new point clipped ➡ $\overrightarrow{NC}$

2. If a portion of the line is inside the clipping boundaries, and the previous point is outside the clipping boundaries:

   previous point clipped ➡ $\overrightarrow{PC}$

d. Perspective/Viewport Transformation

1. If a portion of the line is inside the clipping boundaries, and the previous point is outside the clipping boundaries:

   PCx (perspective and viewport transformed) ➡ PVx, RBx
   PCy (perspective and viewport transformed) ➡ PVy, RBy
   PCz (perspective and viewport transformed) ➡ PVz, RBz

2. If a portion of the line is inside the clipping boundaries:

   NCx (perspective and viewport transformed) ➡ NVx, RBx
   NCy (perspective and viewport transformed) ➡ NVy, RBy
   NCz (perspective and viewport transformed) ➡ NVz, RBz

A.2.3.2 2DDRAW and 3DDRAW, FSM1 = MOVETO or DOT.

a. Input Data to Internal Registers.

1. If 2DDRAW and FSM2 = ABSOLUTE
   INx ➡ PCx, BASEx
   INy ➡ PCy, BASEy
   BASEz ➡ PCz, BASEz
   BASEw ➡ PCw, BASEw

2. If 2DDRAW and FSM2 = RELATIVE
   INx + BASEx → PCx, BASEx
   INy + BASEy → PCy, BASEy
   BASEz → PCz, BASEz
   BASEw → PCw, BASEw

3. If 3DDRAW and FSM2 = ABSOLUTE
   INx → PCx, BASEx
   INy → PCy, BASEy
   INz → PCz, BASEz
   BASEw → PCw, BASEw

4. If 3DDRAW and FSM2 = RELATIVE
   INx + BASEx → PCx, BASEx
   INy + BASEy → PCy, BASEy
   INz + BASEz → PCz, BASEz
   BASEw → PCw, BASEw

b. Transform Data.

$(\overrightarrow{\text{PC}}) \times (\text{TRANMAT}) \rightarrow \overrightarrow{\text{SAVE}}, \overrightarrow{\text{NC}}$

c. Perspective/Viewport Transformation.

If the transformed data (SAVE) is within the clipping boundaries:

$\overrightarrow{\text{NC}}$ (perspective/viewport transformed) → $\overrightarrow{\text{NV}}$, $\overrightarrow{\text{RB}}$

A.2.3.3 2DDRAW and 3DDRAW, FSM1 = STATUS or CHARACTER.

INx → PCx, RBx
INy → PCy, RBy
INz → PCz, RBz

A.2.3.4 PUSH

TRANMAT → Top of Matrix Stack
(TRANMAT is not destroyed).

A.2.3.5 POP

Top of Stack → TRANMAT.

A.2.3.6 MATCON Matrix Concatenation

$(\overrightarrow{\text{IN}}) \times (\overrightarrow{\text{TRANMATi}}) \rightarrow \overrightarrow{\text{TEMPMATi}}$ where i is the row specified by FSM2. If FSM2 = 3, then at the completion of the multiplication the TEMPMAT is normalized and the 16 most significant bits of each element placed in TRANMAT.

A.2.3.7 LOAD

$\overrightarrow{\text{IN}} \rightarrow \overrightarrow{\text{REGi}}$ where i is specified by the FSM fields of the RSR.

A.2.3.8 STORE

$\longrightarrow$

REGi→PDP-11 Memory via DMA, where i is specified by the FSM fields of the RSR.

A.2.4 Character Generator

The Character Generator portion of the Picture System supplies x and y displacement data directly to the Picture Generator. The characters to be displayed are specified by ASCII codes that are passed unmodified through the Picture Processor and stored in the Refresh Buffer.

The x and y data provided by the Character Generator is treated by the Picture Generator as relative vector drawing information. Therefore, the position at which a character string is to be displayed should be specified in the normal manner (i.e. 2DDRAW or 3DDRAW; MOVETO or DRAWTO) before the characters are output.

The Scale Register in the Character Generator is used to specify the size of the characters to be drawn. This register is loaded using 2DDRAW or 3DDRAW STATUS commands. This section provides information relating the size of the characters to the screen coordinate system. (Note: the screen coordinates range is -2048 to 2047 as described in Section 5.2.1.3).

The Character Generator contains character descriptions defined in screen coordinates. If the Scale x and Scale y portions of the Scale Register are both equal to 0 the smallest size is specified. This size character occupies an x space in the screen coordinate system that is 30 screen units wide. It occupies a y space that ranges from +30 to -12 screen units, depending on the character specified (lower case characters are the only ones that may occupy space in the negative direction; all upper case characters occupy the full +30 range). Figure A-4 shows the relative proportion of the upper and lower case smallest characters.

It is important to note that the character definition includes a MOVETO to the right boundary of the space it occupies in x to provide uniform spacing of characters.

The Scale Register contains a bit that provides for the rotation of the character counter-clockwise 90°. When this bit is set the range in scope units that the characters occupy (at the smallest size) is -30 to +12 in x and +30 in y. The final MOVETO in the character defintion goes to the top boundary in this case.

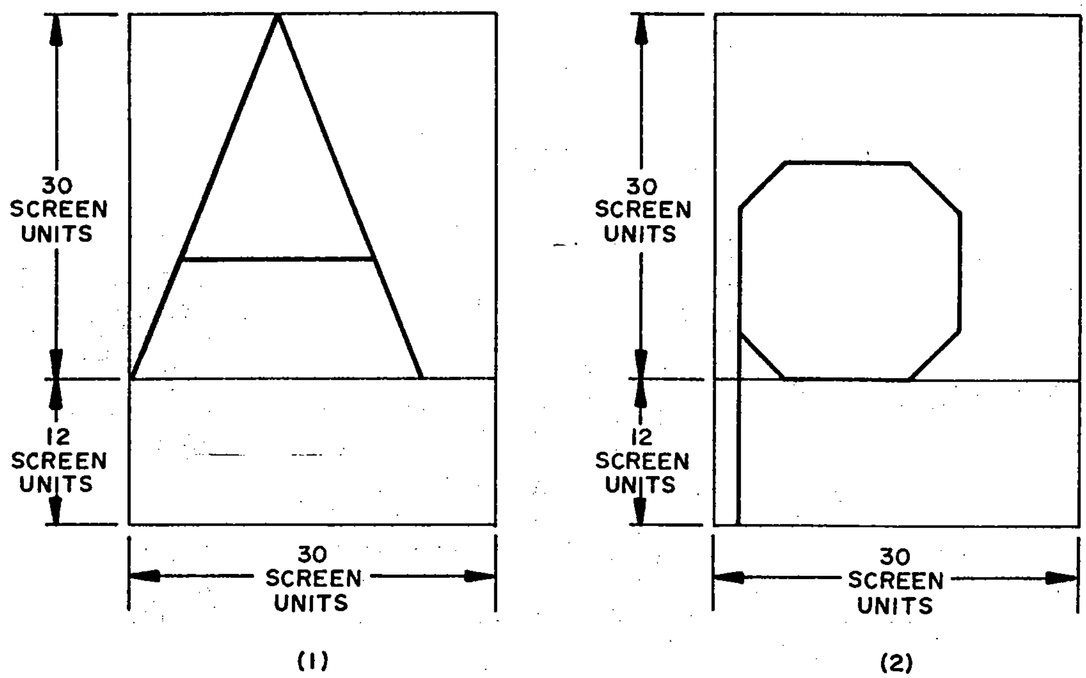


Figure A-4

Relative Character Sizes in Screen Units
showing: (1) the Range of Upper Case Characters
and (2) the Range of Lower Case Characters
for the Smallest Size Characters.

The Scale Register x and y scale fields provide for independent scaling of the height and width of the characters. Table 1 shows the range in screen units and inches for the various sizes:

	Range in Screen Units		Range in Inches	
Scale Value	X	Y	X	Y
0	30	+30,-12	+.07	+.07,-.03
1	60	+60,-24	+.14	+.14,-.06
2	90	+90,-36	+.21	+.21,-.09
3	120	+120,-48	+.28	+.28,-.12
4	150	+150,-60	+.35	+.35,-.15
5	180	+180,-72	+.42	+.42,-.18
6	210	+210,-84	+.49	+.49,-.21
7	240	+240,-96	+.56	+.56,-.24

A.3 PROGRAMMING THE PICTURE SYSTEM

Sections A.2.1 and A.2.2 described the PDP-11 and Picture Processor registers which are used in programming THE PICTURE SYSTEM. The purpose of this section is to describe how these registers may be used to produce a program which interfaces with the hardware at an assembly language level.

A.3.1 Program Description

To illustrate how to interface with THE PICTURE SYSTEM hardware, a simple program will be described which displays a cube and allows the cube to be translated in x,y and z according to console switch settings, while displaying the characters "CUBE" which blink continually. However, in addition to the details which this program will illustrate, the following points should be emphasized:

1. The RSR Coordinate count (CNT) and the DMA word count (DRWC) must have corresponding values to ensure that the operation specified continues to completion. Exactly what the relationship is depends upon the command (COM) specified. The following shows the CNT/DRWC relation for each of the commands: (It should be noted that the CNT field of the RSR will contain the two's complement of the number of executions to be performed and the DRWC will contain the two's complement of the number of PDP-11 words to be transferred.)

```
COM
000-2DDRAW    If FSM1 = MOVETO, DRAWTO or DOT:
                 CNT ←--(number of 16 bit word pairs)
                 DRWC← 2*CNT
              If FSM1 = STATUS OR CHARACTER
                 CNT ←--(number of 16 bit word triples)
                 DRWC← 3*CNT

001-3DDRAW       CNT ←--(number of 16 bit word triples)
                 DRWC← 3*CNT

010-PUSH         CNT ←--1
                 DRWC← 0

011-MATCON       CNT ←--(number of rows to concatenate)
                 DRWC← 4*CNT

100-POP          CNT ←--1
                 DRWC← 0

101-LOAD         CNT ←--(number of sequential registers
                         to load)
                 DRWC← 4*CNT
```

110-STORE   CNT ← (number of sequential registers to store)
             DRWC ← 4*CNT

111-NO OP   CNT ← 0
             DRWC ← 0

It should be noted that when the Enable RSR Update function is used, the DRWC must be adjusted to account for each of the commands which will be executed and the RSRs which are embedded within the data. The NO OP command should be used as the last RSR within the RSR/data list.

2. Data which has been processed by the Picture Processor may be read back into the memory of the PDP-11 by using the STORE command and addressing those registers where the data results are stored. These registers of interest are:

   a. SAVE Register: This will contain the transformed data coordinates before clipping was performed.
   b. New Clip (NC) Register: This will contain the transformed and clipped data coordinates if the most recent data resulted in an element which would normally be displayed.
   c. New View (NV) Register: This will contain the transformed, clipped, and viewport mapped data coordinates if the most recent data resulted in an element which would normally be displayed.
   d. Previous Clip (PC) Register: This will contain the transformed and clipped data coordinates of the computed beginning point of a line whose actual beginning point coordinates were clipped.
   e. Previous View (PV) Register: This will contain the transformed, clipped and viewport mapped data coordinates computed of the beginning point of a line whose actual beginning point coordinates were clipped.

   The "hit bits" may be interrogated as described in Section A.2.1.4 to determine when the NC, NV, PC and PV registers contain meaningful information.

3. Before a 2DDRAW or 3DDRAW command is performed (except STATUS and CHARACTER) the BASE Register (25) should be loaded with the constant z and w coordinates if 2DDRAW or the constant w coordinate if 3DDRAW. This is done because the BASE register supplies the z and w coordinates for 2DDRAW commands and the w coordinate for 3DDRAW commands. When the 2DDRAW command is used in conjunction with the STATUS or CHARACTER specification, three words will be accessed from the PDP-11, rather than

the two normally accessed for 2DDRAW commands.

4. It should be noted that the STATUS words are deposited directly into the Refresh Buffer, and once a STATUS is encountered by the Picture Generator, the STATUS specified will remain in effect (through subsequent frames when no STATUS is encountered) until an overriding STATUS is encountered. A STATUS command which does not have the TAKE bit (bit 11) set is considered to be a Refresh Buffer NO OP command, and hence does not affect the status of the Picture Generator or Character Generator.

The structure of this sample program is consistent with the general structure of a PICTURE SYSTEM program as described in Chapter 5 of THE PICTURE SYSTEM User's Manual and shown in Figure A-5. The following section contains the MACRO-11 assembly language listing of this program. A careful study of this program should clarify many of the topics covered within this appendix. The same program, but written in FORTRAN using the Graphics Software Package, is also shown.

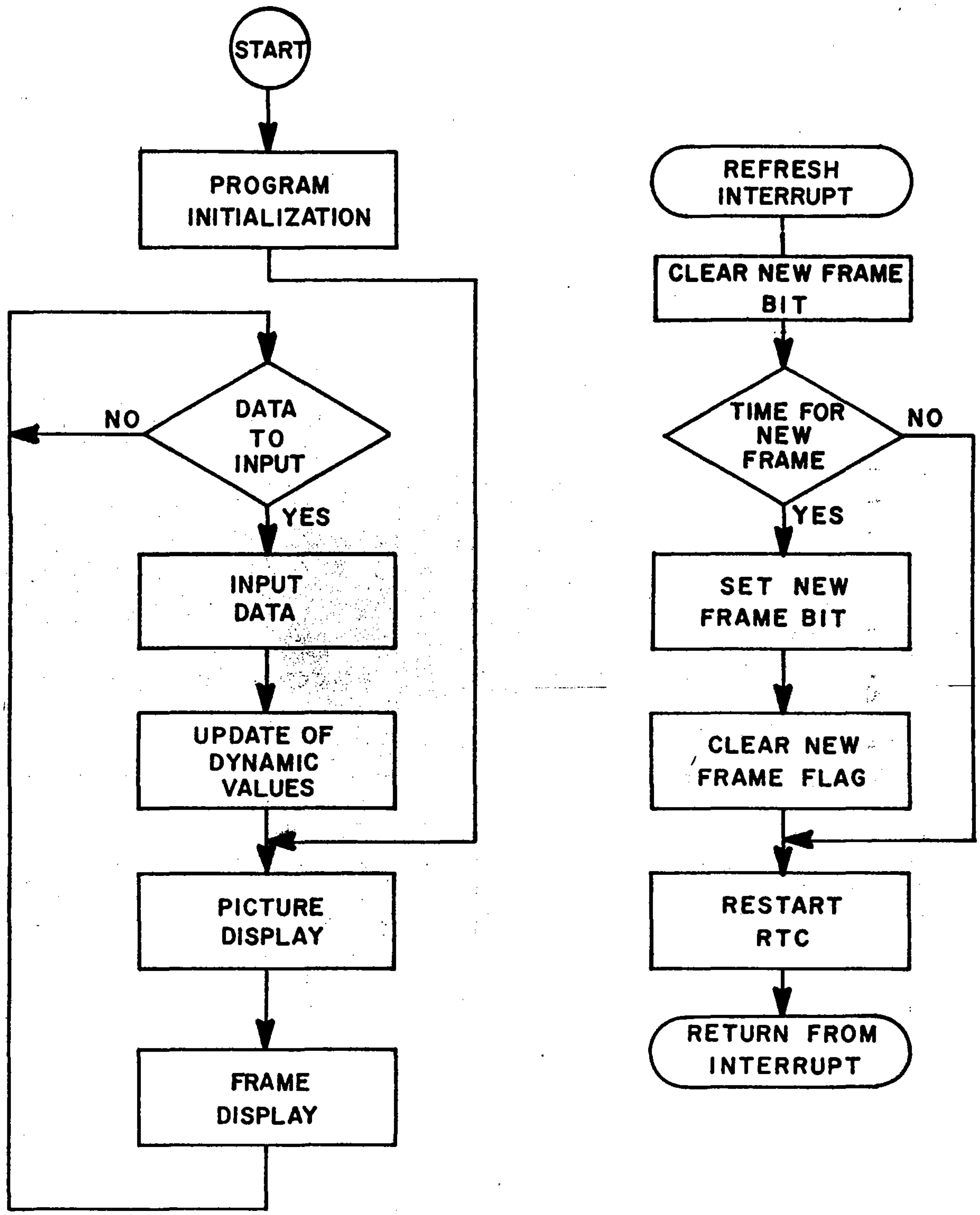


Figure A-5

Sample Program Structure

A.3.2 MACRO-11 Program Example

BLE OF CONTENTS

```
; EVANS & SUTHERLAND COMPUTER CORPORATION     ALL RIGHTS RESERVED ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; THIS SAMPLE PROGRAM DISPLAYS A CUBE ORTHOGRAPHICALLY WITHIN A
; VIEWPORT WHICH IS SPECIFIED AS THE ENTIRE SCREEN USING MACRO-11
; CODE WHICH INTERFACES DIRECTLY WITH THE PICTURE SYSTEM.  THE
; CUBE MAY BE TRANSLATED IN X, Y AND Z DIRECTIONS DEPENDENT UPON
; THE CONSOLE DATA SWITCH SETTINGS, ALL THE WHILE DISPLAYING THE
; CHARACTERS "CUBE" WHICH BLINK CONTINUALLY.
;
; CONSOLE SWITCH SETTINGS:
;
;       SWTICH 15:  PROGRAM RESTART
;       SWITCH 14:  TRANSLATE -X
;       SWITCH 13:  TRANSLATE -Y
;       SWITCH 12:  TRANSLATE -Z
;       SWITCH 11:  TRANSLATE +X
;       SWITCH 10:  TRANSLATE +Y
;       SWITCH  9:  TRANSLATE +Z
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
.TITLE  APPENDIX A:  SAMPLE PROGRAM
```

```
                                .SBTTL  DEFINITION OF REGISTERS AND COMMANDS

        000000                  R0      =%0
        000001                  R1      =%1
        000002                  R2      =%2
        000003                  R3      =%3
        000004                  R4      =%4
        000005                  R5      =%5
        000006                  SP      =%6
        000007                  PC      =%7

        000300                  RTCVEC  =000300         ;RTC INTERRUPT VECTOR
        000240                  RTCPS   =000240         ;RTC PROCESSOR STATUS (FOR INTERRUPT)

        167770                  RTC     =167770         ;RTC REGISTER
        167772                  SR      =167772         ;SR  REGISTER
        167774                  RSR     =167774         ;RSR REGISTER
        172410                  DRWC    =172410         ;DMA WORD COUNT REGISTER
        172412                  DRBA    =172412         ;DMA BASE ADDRESS REGISTER
        172414                  DRST    =172414         ;DMA STATUS REGISTER
        177570                  SWITCH  =177570         ;CONSOLE SWITCH REGISTER

        000000                  DRAW2D  =000000         ;2D DRAW BASIC COMMAND
        020000                  DRAW3D  =020000         ;3D DRAW BASIC COMMAND
        040377                  PUSH    =040377         ;PUSH COMMAND
        100377                  POP     =100377         ;POP COMMAND
        060374                  MATCON  =060374         ;MATCON (4 ROWS AT A TIME) COMMAND
        032377                  STATUS  =032377         ;STATUS (3DDRAW,STATUS,ABS.) COMMAND
        120374                  LOADTM  =120374         ;LOAD TRANSFORMATION MATRIX COMMAND
        130377                  LOADVL  =130377         ;LOAD VIEWPORT L, B, H COMMAND
        132377                  LOADVR  =132377         ;LOAD VIEWPORT R, T, Y COMMAND
        132777                  LOADBS  =132777         ;LOAD BASE REGISTER (2S) COMMAND
```

```
                                         .SBTTL  DEFINITION OF CONSTANTS AND DATA

 000000                                  BEGIN:

                                         ; IDENTITY MATRIX (DIAGONAL=40000 TO AVOID NORMALIZATION)

 000000  040000  000000  000000  IDENT:  .WORD   40000,0,0,0
 000006  000000
 000010  000000  040000  000000          .WORD   0,40000,0,0
 000016  000000
 000020  000000  000000  040000          .WORD   0,0,40000,0
 000026  000000
 000030  000000  000000  000000          .WORD   0,0,0,40000
 000036  040000

                                         ; TRANSLATION MATRIX

 000040  040000  000000  000000  TRNMAT: .WORD   40000,0,0,0     ;40000,00000,00000,00000
 000046  000000
 000050  000000  040000  000000          .WORD   0,40000,0,0     ;00000,40000,00000,00000
 000056  000000
 000060  000000  000000  040000          .WORD   0,0,40000,0     ;00000,00000,40000,00000
 000066  000000
 000070  000000                  TX:     .WORD   0               ;    TX,    TY,    TZ,40000
 000072  000000                  TY:     .WORD   0
 000074  000000  040000          TZ:     .WORD   0,40000

                                         ; INITIAL PICTURE GENERATOR STATUS

 000100  004017                  PGSTAT: .WORD   4017            ;SELECT ALL SCOPES, NO BLINK, NO DASH
 000102  004077                          .WORD   4077            ;SELECT HORIZONTAL, LARGEST CHARACTERS
 000104  000000                          .WORD   0               ;RESERVED FOR COLOR USE

                                         ; BLINK STATUS

 000106  004037  000000  000000  BLNKON: .WORD   4037,0,0        ;SELECT BLINK AND ALL SCOPES
 000114  004017  000000  000000  BLNKOF: .WORD   4017,0,0        ;SELECT ONLY ALL SCOPES

                                         ; VIEWPORT SPECIFICATION (FULL SCREEN, MAX DEPTH CUEING)

 000122  174001  174001  000377  VWLBH:  .WORD   -2047.,-2047.,255.,0
 000130  000000
 000132  003777  003777  000000  VWRTY:  .WORD   2047.,2047.,0,0
 000140  000000
```

```

                                      ; BASE REGISTER VALUES

   000142  000000  000000  000000  ZWBASE: .WORD   0,0,0,32767.       ;Z=0, W=32767. (X & Y DON'T COUNT)
   000150  077777

                                      ; VARIABLES AND DATA

   000152  154360  154360          SETPT:  .WORD   -10000.,-10000.
   000156     103     040     125  TEXT:   .ASCII  'C U B E '
   000161     040     102     040
   000164     105     040
   000166     000     000     000          .BYTE   0,0,0,0            ;FILL OUT THE COUNT TO SIX WITH 0'S
   000171     000
   000172  004000  004000  174000  CDAT1:  .WORD   2048.,2048.,-2048., -2048.,2048.,-2048.
   000200  174000  004000  174000
   000206  174000  174000  174000          .WORD   -2048.,-2048.,-2048., 2048.,-2048.,-2048.
   000214  004000  174000  174000
   000222  004000  004000  174000          .WORD   2048.,2048.,-2048., 2048.,2048.,2048.
   000230  004000  004000  004000
   000236  174000  004000  004000          .WORD   -2048.,2048.,2048., -2048.,-2048.,2048.
   000244  174000  174000  004000
   000252  004000  174000  004000          .WORD   2048.,-2048.,2048., 2048.,2048.,2048.
   000260  004000  004000  004000
   000266  174000  004000  174000  CDAT2:  .WORD   -2048.,2048.,-2048., -2048.,2048.,2048.
   000274  174000  004000  004000
   000302  174000  174000  174000          .WORD   -2048.,-2048.,-2048., -2048.,-2048.,2048.
   000310  174000  174000  004000
   000316  004000  174000  174000          .WORD   2048.,-2048.,-2048., 2048.,-2048.,2048.
   000324  004000  174000  004000
   000332  000000                  NUFRAM: .WORD   0
   000334  000310                  INCVAL: .WORD   200.
```

```
                                    .SBTTL  PROGRAM INITIALIZATION

000336  012706  000000'             START:  MOV     #BEGIN,SP       ;SET THE STACK POINTER FOR RESTART
                                    ;
000342  152767  000001  167771'             BISB    #1,RTC+1        ;MASTER CLEAR THE PICTURE SYSTEM
                                                                    ;THIS DOES THE FOLLOWING:
                                                                    ;  SR   <= 140001
                                                                    ;  DRWC <= 0
                                                                    ;  DRBA <= 0
                                                                    ;  DRST <= 200
                                                                    ;  RTC  <= 0
                                    ;
                                    ; LOAD THE TRANSFORMATION MATRIX WITH THE IDENTITY
                                    ;
000350  012700  120374                      MOV     #LOADTM,R0      ;SET THE LOAD COMMAND
000354  012701  177760                      MOV     #-16.,R1        ;16 WORDS TO BE LOADED
000360  012702  000030'                     MOV     #IDENT,R2       ;SET THE BASE ADDRESS
000364  004767  000632                      JSR     PC,DMAOUT       ;AND DMA IT OUT
                                    ;
                                    ; LOAD THE VIEWPORT AND BASE REGISTERS
                                    ;
000370  012700  130377                      MOV     #LOADVL,R0      ;SET THE LOAD COMMAND (REGISTER 20)
000374  012701  177774                      MOV     #-4.,R1         ;4 WORDS TO BE LOADED
000400  012702  000122'                     MOV     #VWLBH,R2       ;SET THE BASE ADDRESS
000404  004767  000612                      JSR     PC,DMAOUT       ;AND DMA IT OUT
                                    ;
000410  012700  132377                      MOV     #LOADVR,R0      ;SET THE LOAD COMMAND (REGISTER 24)
000414  012701  177774                      MOV     #-4.,R1         ;4 WORDS TO BE LOADED
000420  012702  000132'                     MOV     #VWRTY,R2       ;SET THE BASE ADDRESS
000424  004767  000572                      JSR     PC,DMAOUT       ;AND DMA IT OUT
                                    ;
000430  012700  132777                      MOV     #LOADBS,R0      ;SET THE LOAD COMMAND (REGISTER 25)
000434  012701  177774                      MOV     #-4.,R1         ;4 WORDS TO BE LOADED
000440  012702  000142'                     MOV     #ZWBASE,R2      ;SET THE BASE ADDRESS
000444  004767  000552                      JSR     PC,DMAOUT       ;AND DMA IT OUT
                                    ;
                                    ; NOTE:  REGISTER 25 COULD HAVE BEEN LOADED WHEN REGISTER 24 WAS
                                    ;        LOADED BY DECREMENTING THE RSR CNT BY 1 AND SETTING THE
                                    ;        DRWC TO TRANSFER 4 MORE WORDS
                                    ;
                                    ; SET THE INITIAL PICTURE GENERATOR STATUS
                                    ;
000450  012700  032377                      MOV     #STATUS,R0      ;SET THE STATUS COMMAND
000454  012701  177775                      MOV     #-3.,R1         ;3 WORDS TO BE OUTPUT
```

```
000460  012702  002100'          MOV     #PGSTAT,R2         ;SET THE BASE ADDRESS
000464  004767  000532           JSR     PC,DMAOUT          ;AND DMA IT OUT
                         ;
                         ; INITIALIZE VARIABLES
                         ;
000470  005067  177636           CLR     NUFRAY
000474  005067  177370           CLR     TX
000500  005067  177363           CLR     TY
000504  005067  177364           CLR     TZ
                         ;
                         ; TURN ON THE RTC TO REFRESH AT 40 FRAMES/SECOND
                         ;
000510  012767  000534' 000300'  MOV     #RTCINT,RTCVEC     ;SET THE ADDRESS OF THE INTERRUPT
000516  012767  000240  000302'  MOV     #RTCPS,RTCVEC+2    ;SET THE INTERRUPT PROCESSOR STATUS
000524  112767  000075  167770'  MOVB    #75,RTC            ;START THE RTC (REFRESH RATE = -3)
                         ;
                         ; AND OUTPUT AT LEAST THE FIRST FRAME
                         ;
000532  000512                   BR      DSPLAY             ;BRANCH AROUND THE DATA INPUT
```

```
                                        .SBTTL  RTC INTERRUPT SERVICE ROUTINE

000534  132767  000200  167773' RTCINT: BITB    #200,SR+1       ;CHECK FOR RM STOPPED
000542  001004                          BNE     RTC100          ;BRANCH IF RM STOPPED
000544  112767  000077  167770'         MOVB    #77,RTC         ;ELSE START THE RTC FOR 1 TICK
000552  000002                          RTI                     ;AND RETURN FROM INTERRUPT
                                ;
000554  132767  000200  172414' RTC100: BITB    #200,DRST       ;WAIT FOR DMA READY
000562  001774                          BEQ     RTC100          ;LOOP IF NOT READY
000564  132737  000001  167772' RTC150: BITB    #1,SR           ;WAIT FOR MAP DONE
000572  001774                          BEQ     RTC150          ;LOOP IF NOT DONE
                                ;
000574  142767  000007  167773'         BICB    #7,SR+1         ;CLEAR THE ND, NC AND DC BITS
                                ;
                                ;       TEST FOR A NEW FRAME INDICATION
                                ;
000602  005767  177524                  TST     NUFRAM          ;CHECK THE NEW FRAME FLAG
000606  001406                          BEQ     RTC200          ;BRANCH IF NO NEW FRAME (=0)
000610  152767  000004  167773'         BISB    #4,SR+1         ;SET THE NEW FRAME BIT
000616  005067  177510                  CLR     NUFRAM          ;CLEAR THE USER FLAG
                                ;
000622  142767  000200  167773' RTC200: BICB    #200,SR+1       ;RESTART THE REFRESH PROCESS
000630  112767  000075  167770'         MOVB    #75,RTC         ;RESTART THE RTC (40 FRAMES/SECOND)
000636  000002                          RTI                     ;AND RETURN FROM INTERRUPT
```

```
                                        .SBTTL  DATA TO INPUT?

000640  016700  177570'         PLOOP:  MOV     SWITCH,R0       ;GET SWITCH VALUE
000644  001775                          BEQ     PLOOP           ;AND LOOP IF NOTHING IS SET
000646  002633                          BLT     START           ;RESTART IF SW15 IS SET
                                ;
                                ; CHECK SWITCH 14
                                ;
000650  032700  040000                  BIT     #40000,R0       ;SW 14 SET?
000654  001403                          BEQ     SW13            ;BRANCH IF NOT
000656  166767  177452  177204          SUB     INCVAL,TX       ;TRANSLATE -X
                                ;
                                ; CHECK SWITCH 13
                                ;
000664  032700  020000          SW13:   BIT     #20000,R0       ;SW 13 SET?
000670  001403                          BEQ     SW12            ;BRANCH IF NOT
000672  166767  177436  177172          SUB     INCVAL,TY       ;TRANSLATE -Y
                                ;
                                ; CHECK SWITCH 12
                                ;
000700  032700  010000          SW12:   BIT     #10000,R0       ;SW 12 SET?
000704  001403                          BEQ     SW11            ;BRANCH IF NOT
000706  166767  177422  177160          SUB     INCVAL,TZ       ;TRANSLATE -Z
                                ;
                                ; CHECK SWITCH 11
                                ;
000714  032700  004000          SW11:   BIT     #4000,R0        ;SW 11 SET?
000720  001403                          BEQ     SW10            ;BRANCH IF NOT
000722  066767  177408  177140          ADD     INCVAL,TX       ;TRANSLATE +X
                                ;
                                ; CHECK SWITCH 10
                                ;
000730  032700  002000          SW10:   BIT     #2000,R0        ;SW 10 SET?
000734  001403                          BEQ     SW09            ;BRANCH IF NOT
000736  066767  177372  177126          ADD     INCVAL,TY       ;TRANSLATE +Y
                                ;
                                ; CHECK SWITCH 9
                                ;
000744  032700  001000          SW09:   BIT     #1000,R0        ;SW 9 SET?
000750  001403                          BEQ     DSPLAY          ;BRANCH IF NOT
000752  066767  177356  177114          ADD     INCVAL,TZ       ;TRANSLATE +Z
```

```
                                        .SBTTL  PICTURE DISPLAY

        000760                          DSPLAY:
                                        ;
                                        ; SET BLINK MODE ON
                                        ;
        000760  012700  032377                  MOV     #STATUS,R0      ;SET THE STATUS COMMAND
        000764  012701  177775                  MOV     #-3.,R1         ;3 WORDS TO BE OUTPUT
        000770  012702  000106'                 MOV     #BLNKON,R2      ;SET THE BLINK PASE ADDRESS
                                        ;
                                        ; WAIT FOR THE NEW FRAME FLAG TO BE CLEARED BEFORE OUTPUT
                                        ;
        000774  005767  177332          DSP010: TST     NUFRAM          ;IS IT =0 YET?
        001000  001375                          BNE     DSP010          ;BRANCH IF NOT
                                        ;
        001002  004767  000214                  JSR     PC,DMAOUT       ;NEW FRAME... DMA IT OUT
                                        ;
                                        ; RESET THE BASE REGISTER
                                        ;
        001006  012700  132777                  MOV     #LOADBS,R0      ;SET THE LOAD COMMAND (REGISTER 25)
        001012  012701  177774                  MOV     #-4.,R1         ;4 WORDS TO BE LOADED
        001016  012702  000142'                 MOV     #ZWBASE,R2      ;SET THE BASE ADDRESS
        001022  004767  000174                  JSR     PC,DMAOUT       ;AND DMA IT OUT
                                        ;
                                        ; DO THE SET-POINT FOR THE TEXT
                                        ;
        001026  012700  005377                  MOV     #DRAW2D+5377,R0 ;SET THE BASIC 2D DRAW COMMAND
                                                                        ;ALONG WITH MOVETO-DRAWTO-ETC.
        001032  012701  177776                  MOV     #-2.,R1         ;2 WORDS TO BE OUTPUT
        001036  012702  000152'                 MOV     #SETPT,R2       ;SET THE DATA ADDRESS
        001042  004767  000154                  JSR     PC,DMAOUT       ;AND DMA IT OUT
                                        ;
                                        ; NOW OUTPUT THE ASCII CHARACTERS
                                        ;
        001046  012700  037376                  MOV     #DRAW3D+17376,R0 ;SET THE BASIC 3D DRAW COMMAND
                                                                        ;ALONG WITH CHARACTER-ABSOLUTE
        001052  012701  177772                  MOV     #-6.,R1         ;6 WORDS OF CHARACTERS (1 PER WORD)
        001056  012702  000156'                 MOV     #TEXT,R2        ;SET THE TEXT ADDRESS
        001062  004767  000134                  JSR     PC,DMAOUT       ;AND DMA IT OUT
                                        ;
                                        ; SET BLINK MODE OFF
                                        ;
        001066  012700  032377                  MOV     #STATUS,R0      ;SET THE STATUS COMMAND
        001072  012701  177775                  MOV     #-3.,R1         ;3 WORDS TO BE OUTPUT
```

```
001076  012702  000114'          MOV     #BLNKOF,R2       ;SET THE BLINK OFF BASE ADDRESS
001102  004767  000114           JSR     PC,DMAOUT        ;AND DMA IT OUT
                          ;
                          ; PUSH THE CURRENT TRANSFORMATION MATRIX
                          ;
001106  012700  040377           MOV     #PUSH,R0         ;SET THE PUSH COMMAND
001112  005001                   CLR     R1               ;A DATALESS COMMAND
                                                          ;R2 CAN BE ANYTHING
001114  004767  000102           JSR     PC,DMAOUT        ;AND DMA IT OUT
                          ;
                          ; CONCATENATE THE TRANSLATION MATRIX
                          ;
001120  012700  060374           MOV     #MATCON,R0       ;SET THE MATCON COMMAND
001124  012701  177760           MOV     #-16.,R1         ;16 WORDS WILL BE OUTPUT
001130  012702  000040'          MOV     #TRNMAT,R2       ;SET THE MATRIX BASE ADDRESS
001134  004767  000062           JSR     PC,DMAOUT        ;AND DMA IT OUT
                          ;
                          ; NOW DRAW THE CUBE (IN 2 INSTALLMENTS)
                          ;
001140  012700  025366           MOV     #DRAW3D+5366,R0  ;SET THE BASIC 3D DRAW COMMAND
                                                          ;ALONG WITH MOVETO-DRAWTO-DRAWTO-ETC.
001144  012701  177742           MOV     #-30.,R1         ;30 WORDS TO BE OUTPUT
001150  012702  000172'          MOV     #CDAT1,R2        ;SET THE DATA BASE ADDRESS
001154  004767  000042           JSR     PC,DMAOUT        ;AND DMA IT OUT
                          ;
001160  012700  021372           MOV     #DRAW3D+1372,R0  ;SET THE BASIC 3D DRAW COMMAND
                                                          ;ALONG WITH MOVETO-DRAWTO-DRAWTO-ETC.
001164  012701  177756           MOV     #-18.,R1         ;18 WORDS TO BE OUTPUT
001170  012702  000236'          MOV     #CDAT2,R2        ;SET THE DATA BASE ADDRESS
001174  004767  000022           JSR     PC,DMAOUT        ;AND DMA IT OUT
                          ;
                          ; INITIATE THE DISPLAY OF THE DATA (SET THE NEW FRAME FLAG)
                          ;
001200  005267  177126           INC     NUFRAM
                          ;
                          ; POP THE ORIGINAL TRANSFORMATION MATRIX BACK
                          ;
001204  012700  100377           MOV     #POP,R0          ;SET THE POP COMMAND
001210  005001                   CLR     R1               ;A DATALESS COMMAND
001212  004767  000004           JSR     PC,DMAOUT        ;AND DMA IT OUT
                          ;
                          ; RESTART THE DISPLAY LOOP
                          ;
001216  000167  177416           JMP     PLOOP            ;AND CONTINUE
```

```
                                        .SBTTL  DMA OUTPUT ROUTINE

                                        ;
                                        ; THIS ROUTINE DOES ALL OF THE DMA OUTPUT FOR THIS PROGRAM.
                                        ;       EXPECTED PARAMETERS ARE:
                                        ;               R0=RSR COMMAND
                                        ;               R1=DMA WORD COUNT FOR DRWC REGISTER
                                        ;               R2=DMA BASE ADDRESS FOR DRBA REGISTER
                                        ;
 001222  032767  000200  172414' DMAOUT: BIT     #200,DRST       ;IS THE DMA "READY"?
 001230  001774                          BEQ     DMAOUT          ;LOOP IF NOT
                                        ;
 001232  010137  172410'                 MOV     R1,DRWC         ;SET THE DMA WORD COUNT
 001236  010237  172412'                 MOV     R2,DRBA         ;SET THE DMA BASE REGISTER.
                                        ;
 001242  132767  000001  167772' PPDONE: BITB    #1,SR           ;WAIT FOR THE PICTURE PROCESSOR
 001250  001774                          BEQ     PPDONE          ;TO FINISH
 001252  010067  167774'                 MOV     R0,RSR          ;AND THEN SET THE RSR
                                        ;
 001256  152767  000001  172414'         BISB    #1,DRST         ;START THE DMA TRANSFER
                                        ;
 001264  000207                          RTS     PC              ;AND RETURN
                                        ;
         000330'                         .END    START
```

```
APPENDIX A:  SAMPLE PROGRAM     MACRO V06-03 01-SEP-74 08:30 PAGE 8-1
SYMBOL TABLE

BEGIN   000000R        BLNKOF  000114R        BLNKON  000106R
CDAT1   000172R        CDAT2   000266R        DMAOUT  001222R
DRAWBD= 000000         DRAWSD= 000000         DREA  = 172412
DRST  = 172414         DRWC  = 172410         DSPLAY  000720R
DSY012  000774R        IDENT   000300R        INOVAL  001334R
LOADBS= 132777         LOADTM= 130374         LOADVL= 130377
LOADVR= 133377         MATCON= 000374         NUFRAM  000332R
PGSTRT  000100R        PLOOP   000340R        POP   = 130377
PRDONE  001134R        PUSH  = 140377         RSR   = 167774
RTC   = 167770         RTCINT  000334R        RTCPS = 000240
RTCVEC= 000302         RTC100  000554R        RTC150  000564R
RTC200  000522R        SETPT   000152R        SR    = 167772
START   000036R        STATUS= 133377         SWITCH= 177570
SW08    000744R        SW10    000730R        SW11    000714R
SW12    000700R        SW13    000364R        TEXT    000153R
TRNMAT  000040R        TX      000070R        TY      000072R
TZ      000074R        VXLBH   000122R        VXRTY   000132R
ZABASE  000142R
. ABS.  000000    000
        001332    001
ERRORS DETECTED:  0
FREE CORE  4435. WORDS
SAMPLE,SAMPLE/NL:TTM/CRF<SAMPLE
```

A.3.3 FORTRAN Program Example

```
      C EVANS & SUTHERLAND COMPUTER CORPORATION      ALL RIGHTS RESERVED C
      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      C
      C THIS SAMPLE FORTRAN PROGRAM DISPLAYS CUBE ORTHOGRAPHICALLY WITHIN
      C A VIEWPORT WHICH IS SPECIFIED AS THE ENTIRE SCREEN USING THE
      C PICTURE SYSTEM GRAPHICS SOFTWARE TO INTERFACE TO THE PICTURE SYSTEM.
      C THE CUBE MAY BE TRANSLATED IN X, Y AND Z DIRECTIONS DEPENDENT UPON
      C THE CONSOLE DATA SWITCH SETTINGS, ALL THE WHILE DISPLAYING THE
      C CHARACTERS "CUBE" WHICH BLINK CONTINUALLY.
      C
      C CONSOLE SWITCH SETTINGS
      C
      C        SWITCH 15:  PROGRAM RESTART
      C        SWITCH 14:  TRANSLATE -X
      C        SWITCH 13:  TRANSLATE -Y
      C        SWITCH 12:  TRANSLATE -Z
      C        SWITCH 11:  TRANSLATE +X
      C        SWITCH 10:  TRANSLATE +Y
      C        SWITCH  9:  TRANSLATE +Z
      C
      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      C
0001           INTEGER CDAT1(3,10),CDAT2(3,6),SETPT(2)
0002           INTEGER ITX,ITY,ITZ,INCVAL
      C
0003           DATA CDAT1/2048,2048,-2048,-2048,2048,-2048,
               1          -2048,-2048,-2048,2048,-2048,-2048,
               2           2048,2048,-2048,2048,2048,2048,
               3          -2048,2048,2048,-2048,-2048,2048,
               4           2048,-2048,2048,2048,2048,2048/
0004           DATA CDAT2/-2048,2048,-2048,-2048,2048,2048,
               1           -2048,-2048,-2048,-2048,-2048,2048,
               2            2048,-2048,-2048,2048,-2048,2048/
0005           DATA SETPT/-10000,-10000/
0006           DATA INCVAL/200/
      C
      C INITIALIZE VARIABLES AND THE PICTURE SYSTEM
      C
0007  100      ITX=0
0008           ITY=0
0009           ITZ=0
0010           CALL PSINIT(3,0,,,,)
      C
      C GO DISPLAY AT LEAST ONE FRAME
      C
0011           GO TO 400
```

```
        C
        C ANY CONSOLE SWITCHES SET?
        C
0012    200     CALL SWITCH(I)
0013            IF(I.EQ.0) GO TO 200
        C
        C THERE IS DATA TO INPUT...CHECK THE INDIVIDUAL SWITCHES
        C
        C RESTART PROGRAM?
        C
0014            CALL SSWTCH(15,I)
0015            IF(I.EQ.1) GO TO 100
        C
        C CHECK SWITCH 14
        C
0016            CALL SSWTCH(14,I)
0017            IF(I.EQ.1) ITX=ITX-INCVAL
0018            CALL SSWTCH(13,I)
0019            IF(I.EQ.1) ITY=ITY-INCVAL
0020            CALL SSWTCH(12,I)
0021            IF(I.EQ.1) ITZ=ITZ-INCVAL
0022            CALL SSWTCH(11,I)
0023            IF(I.EQ.1) ITX=ITX+INCVAL
0024            CALL SSWTCH(10,I)
0025            IF(I.EQ.1) ITY=ITY+INCVAL
0026            CALL SSWTCH(9,I)
0027            IF(I.EQ.1) ITZ=ITZ+INCVAL
        C
        C NOW DISPLAY THE DATA
        C
0028            CALL BLINK(1)          <= CALL TRAN (ITX, ITY, ITZ
0029            CALL DRAW2D(SETPT,1,2,2,0)
0030            CALL TEXT(4,'CUBE')
0031            CALL BLINK(0)
        C
        C AND THE CUBE
        C
0032            CALL DRAW3D(CDAT1,12,2,2)
0033            CALL DRAW3D(CDAT2,6,2,2)
        C
        C INITIATE THE DISPLAY OF THE NEW FRAME
        C
0034            CALL NUFRAM
0035            GO TO 200
0036            END
```

```
FORTRAN V06.13                    08:08:38    01-SEP-74   PAGE    3

      ROUTINES CALLED:
      PEINIT, SWITCH, SSWTCH, BLINK , DRAW2D, TEXT  , DRAW2D
      NUFRAM

      OPTIONS =/ON,/OP:1

      BLOCK        LENGTH
      MAIN    348    (001270)*

      **COMPILER ----- CORE**
         PHASE      USED  FREE
      DECLARATIVES 00122 08344
      EXECUTABLES  00643 07823
      ASSEMBLY     [illegible] 10751
```

CROSS REFERENCE TABLE   S-1

BEGIN	3- 3#	4- 3							
[illegible]LNKOF	3-30#	7-45							
[illegible]LNKON	3-23#	7- 9							
C[illegible]T1	3-48#	7-67							
C[illegible]T2	3-51#	7-73							
[illegible]	4-19	4-25	4-30	4-35	4-46	7-16	7-23	7-31	7-39
	7-48	7-55	7-60	7-62	7-74	7-84	8-10#	8-11	
[illegible]	3-20#	7-27							
[illegible]	3-24#	7-35	7-54	7-77					
[illegible]	3-19#	8-14@							
[illegible]	3-21	5- 2	8-12	8-20@					
[illegible]	7-12#	8-13@							
[illegible]LAY	4-12	6-48	7- 3#						
[illegible]	7-13#	7-14							
[illegible]	3- 7#	4-17							
[illegible]	3-55#	6-11	6-17	6-23	6-29	6-35	6-41		
[illegible]	3-32#	4-32	7-20						
[illegible]	3-20#	4-15							
[illegible]	3-33#	4-22							
[illegible]	3-31#	4-27							
[illegible]	3-27#	7-57							
[illegible]	3-54#	4-50@	5-17	5-20@	7-13	7-78@			
PC	3-12#	4-18@	4-25@	4-30@	4-35@	4-46@	7-16@	7-22@	7-31@
	7-39@	7-48@	7-53@	7-60@	7-68@	7-74@	7-84@	8-22@	
[illegible]	3-20#	4-45							
[illegible]	3- 3#	3- 4	7-59						
[illegible]	2-23#	7-22							
[illegible]	3-10#	8-17							
[illegible]	2-35#	7-50							
[illegible]	3-17#	8-18@							
[illegible]	3-15#	4- 5@	4-59@	5- 5@	5-23@				
[illegible]	4-57	5- 3#							
[illegible]	3-13#	4-52							
[illegible]	3-13#	4-57@	4-58@						
[illegible]	5- 4	5- 8#	5- 9						
[illegible]	5-10#	5-11							
[illegible]	[illegible]	5-22#							
R0	3- 3#	4-15@	4-22@	4-27@	4-32@	4-43@	6- 3@	6- 9	6-15
	3-21	8-27	6-33	6-23	7- 7@	7-20@	7-27@	7-35@	7-43@
	7-10@	7-57@	7-64@	7-72@	7-83@	8-18			
R1	3- 4#	4-18@	4-23@	4-23@	4-39@	4-44@	7- 8@	7-21@	7-29@
	7-37@	7-44@	7-51@	7-53@	7-66@	7-72@	7-83@	8-13	

CROSS REFERENCE TABLE   9-2

P2	2- 5#	4-17@	4-24[illegible]	4-29@	4-34@	4-45@	7- 9@	7-22@	7-30@
	7-3[illegible]	7-4[illegible]	7-5[illegible]	7-67@	7-72[illegible]	8-14			
P3	2- [illegible]								
R4	[illegible]								
RE	[illegible]								
SETPT	3-4[illegible]	7-30							
SP	2- [illegible]	4- 3@							
[illegible]	[illegible]	5- [illegible]	5-1[illegible]	5-13@	5-19[illegible]	5-22@	8-16		
[illegible]ART	[illegible]	8- 5	8-24						
STATUS	[illegible]	4-4[illegible]	7- [illegible]	7-43					
[illegible]	[illegible]	8- [illegible]							
[illegible]	[illegible]	[illegible]							
[illegible]	[illegible]	[illegible]							
[illegible]	[illegible]	[illegible]							
[illegible]	[illegible]	[illegible]							
[illegible]	[illegible]	[illegible]							
[illegible]	[illegible]	7-3[illegible]							
[illegible]	[illegible]	7-5[illegible]							
[illegible]	[illegible]	4-[illegible]	6-11@	6-29@					
[illegible]	[illegible]	4-[illegible]	6-17[illegible]	6-2[illegible]					
[illegible]	[illegible]	[illegible]	6-2[illegible]	[illegible]					
[illegible]	[illegible]	4-[illegible]							
[illegible]	[illegible]	4-[illegible]							
[illegible]	[illegible]	[illegible]	7-22						

APPENDIX B

SUMMARY OF THE GRAPHICS SUBROUTINES

This appendix contains a summary of the FORTRAN AND MACRO-11 assembly language calling sequence specifications for THE PICTURE SYSTEM Graphics Subroutines. Also included is Table B-1, which summarizes the Graphics Subroutines Error Codes. This code is used to indicate the subroutine which detected a user error should one occur.

TABLE B-1
SUBROUTINE ERROR CODE CORRESPONDENCE

SUBROUTINE NAME	ERROR CODE[1]
PSINIT	1
NUFRAM	2
VWPORT	3
WINDOW,MASTER	4
INST	5
PUSH	6
POP	7
TRAN	8
ROT	9
DRAW2D	10
DRAW3D	11
TEXT	12
TABLET	13
CURSOR	14
HITWIN	15
HITEST	16
SCALE	17
CHAR	18
DASH	19
BLINK	20
SCOPE	21
SETBUF	22
BLDCON	0

----

[1] If an error occurs that is detected by one of these subroutines, then the error code will indicate which subroutine the error was detected in.

B.1 FORTRAN CALLING SEQUENCES:

```
[EXTERNAL ERRSUB]
CALL PSINIT(IFTIME,INRFSH,[ICLOCK],[ERRSUB],[ISTKCT],
     [ISTKAD][,IFMCNT])
CALL VWPORT(IVL,IVR,IVB,IVT,IHI,IYI)
CALL WINDOW(IWL,IWR,IWB,IWT[,IW])
CALL WINDOW(IWL,IWB,IWT,IWH,IWY[,IE[,IW]])
CALL ROT(IANGLE,IAXIS)
CALL TRAN(ITX,ITY,ITZ[,IW])
CALL SCALE(ISX,ISY,ISZ[,IW])
CALL PUSH
CALL POP
CALL DRAW2D(IDATA,INUM,IF1,IF2,IZ[,IW])
CALL DRAW3D(IDATA,INUM,IF1,IF2[,IW])
CALL CHAR(IXSIZE,IYSIZE,ITILT)
CALL TEXT(NCHARS,ITEXT)
CALL INST(INL,INR,INB,INT[,IW])
CALL INST(INL,INR,INB,INT,INH,INY[,IW])
CALL MASTER(IML,IMR,IMB,IMT[,IW])
CALL MASTER(IML,IMR,IMB,IMT,IMH,IMY[,IW])
CALL DASH(ISTAT)
CALL BLINK(ISTAT)
CALL SCOPE(INUM)
CALL TABLET(ISTAT[,IX,IY,IPEN])
CALL CURSOR(IX,IY,ISTAT[,IPEN])
CALL HITWIN(IX,IY,ISIZE[,IW])
CALL HITEST(IHIT,ISTAT)
CALL NUFRAM
CALL SETBUF(ISTAT)
CALL PSWAIT
CALL BLDCON(ITYPE,IARRAY)
```

B.2 ASSEMBLY LANGUAGE CALLING SEQUENCES

All subroutines should be declared global (.GLOBL). All arguments are addresses of parameters.

PSINIT

```
        MOV     #ADR,R5
        JSR     PC,PSINIT
ADR:    BR      .+14.
        .WORD IFTIME,INRFSH,ICLOCK,ERRSUB,ISTKCT,ISTKAD
 or
        MOV     #ADR,R5
        JSR     PC,PSINIT
ADR:    BR      .+16.
        .WORD IFTIME,INRFSH,ICLOCK,ERRSUB,ISTKCT,ISTKAD,IFMCNT
```

VWPORT

```
        MOV     #ADR,R5
        JSR     PC,VWPORT
ADR:    BR      .+14.
        .WORD IVL,IVR,IVB,IVT,IHI,IYI
```

WINDOW

```
        MOV     #ADR,R5
        JSR     PC,WINDOW
ADR:    BR      .+10.
        .WORD IWL,IWR,IWB,IWT
 or
        MOV     #ADR,R5
        JSR     PC,WINDOW
ADR:    BR      .+12.
        .WORD IWL,IWR,IWB,IWT,IW
 or
        MOV     #ADR,R5
        JSR     PC,WINDOW
ADR:    BR      .+14.
        .WORD IWL,IWR,IWB,IWT,IWH,IWY
 or
        MOV     #ADR,R5
        JSR     PC,WINDOW
ADR:    BR      .+16.
        .WORD IWL,IWR,IWB,IWT,IWH,IWY,IE
 or
        MOV     #ADR,R5
        JSR     PC,WINDOW
ADR:    BR      .+18.
        .WORD IWL,IWR,IWB,IWT,IWH,IWY,IE,IW
```

ROT

```
        MOV    #ADR,R5
        JSR    PC,ROT
ADR:    BR     .+6.
        .WORD IANGLE,IAXIS
```

TRAN

```
        MOV    #ADR,R5
        JSR    PC,TRAN
ADR:    BR     .+8.
        .WORD ITX,ITY,ITZ
  or
        MOV    #ADR,R5
        JSR    PC,TRAN
ADR:    BR     .+10.
        .WORD ITX,ITY,ITZ,IW
```

SCALE

```
        MOV    #ADR,R5
        JSR    PC,SCALE
ADR:    BR     .+8.
        .WORD ISX,ISY,ISZ
  or
        MOV    #ADR,R5
        JSR    PC,SCALE
ADR:    BR     .+10.
        .WORD ISX,ISY,ISZ,IW
```

PUSH

```
        JSR    PC,PUSH
```

POP

```
        JSR    PC,POP
```

DRAW2D

```
        MOV    #ADR,R5
        JSR    PC,DRAW2D
ADR:    BR     .+12.
        .WORD IDATA,INUM,IF1,IF2,IZ
  or
        MOV    #ADR,R5
        JSR    PC,DRAW2D
ADR:    BR     .+14.
        .WORD IDATA,INUM,IF1,IF2,IZ,IW
```

DRAW3D

```
        MOV     #ADR,R5
        JSR     PC,DRAW3D
ADR:    BR      .+10.
        .WORD   IDATA,INUM,IF1,IF2
 or
        MOV     #ADR,R5
        JSR     PC,DRAW3D
ADR:    BR      .+12.
        .WORD   IDATA,INUM,IF1,IF2,IW
```

CHAR

```
        MOV     #ADR,R5
        JSR     PC,CHAR
ADR:    BR      .+8.
        .WORD   IXSIZE,IYSIZE,ITILT
```

TEXT

```
        MOV     #ADR,R5
        JSR     PC,TEXT
ADR:    BR      .+6.
        .WORD   NCHARS,ITEXT
```

INST

```
        MOV     #ADR,R5
        JSR     PC,INST
ADR:    BR      .+10.
        .WORD   INL,INR,INB,INT
 or
        MOV     #ADR,R5
        JSR     PC,INST
ADR:    BR      .+12.
        .WORD   INL,INR,INB,INT,IW
 or
        MOV     #ADR,R5
        JSR     PC,INST
ADR:    BR      .+14.
        .WORD   INL,INR,INB,INT,INH,INY
 or
        MOV     #ADR,R5
        JSR     PC,INST
ADR:    BR      .+16.
        .WORD   INL,INR,INB,INT,INH,INY,IW
```

MASTER

```
        MOV    #ADR,R5
        JSR    PC,MASTER
ADR:    BR     .+10.
        .WORD  IML,IMR,IMB,IMT
  or
        MOV    #ADR,R5
        JSR    PC,MASTER
ADR:    BR     .+12.
        .WORD  IML,IMR,IMB,IMT,IW
  or
        MOV    #ADR,R5
        JSR    PC,MASTER
ADR:    BR     .+14.
        .WORD  IML,IMR,IMB,IMT,IMH,IMY
  or
        MOV    #ADR,R5
        JSR    PC,MASTER
ADR:    BR     .+16.
        .WORD  IML,IMR,IMB,IMT,IMH,IMY,IW
```

DASH

```
        MOV    #ADR,R5
        JSR    PC,DASH
ADR:    BR     .+4.
        .WORD  ISTAT
```

BLINK

```
        MOV    #ADR,R5
        JSR    PC,BLINK
ADR:    BR     .+4.
        .WORD  ISTAT
```

SCOPE

```
        MOV    #ADR,R5
        JSR    PC,SCOPE
ADR:    BR     .+4.
        .WORD  INUM
```

TABLET

```
        MOV    #ADR,R5
        JSR    PC,TABLET
ADR:    BR     .+4.
        .WORD  ISTAT
```

or

```
        MOV     #ADR,R5
        JSR     PC,TABLET
ADR:    BR      .+10.
        .WORD   ISTAT,IX,IY,IPEN
```

<u>CURSOR</u>

```
        MOV     #ADR,R5
        JSR     PC,CURSOR
ADR:    BR      .+8.
        .WORD   IX,IY,ISTAT
```

or

```
        MOV     #ADR,R5
        JSR     PC,CURSOR
ADR:    BR      .+10.
        .WORD   IX,IY,ISTAT,IPEN
```

<u>HITWIN</u>

```
        MOV     #ADR,R5
        JSR     PC,HITWIN
ADR:    BR      .+8.
        .WORD   IX,IY,ISIZE
```

or

```
        MOV     #ADR,R5
        JSR     PC,HITWIN
ADR:    BR      .+10.
        .WORD   IX,IY,ISIZE,IW
```

<u>HITEST</u>

```
        MOV     #ADR,R5
        JSR     PC,HITEST
ADR:    BR      .+6.
        .WORD   IHIT,ISTAT
```

<u>NUFRAM</u>

```
        JSR     PC,NUFRAM
```

<u>SETBUF</u>

```
        MOV     #ADR,R5
        JSR     PC,SETBUF
ADR:    BR      .+4.
        .WORD   ISTAT
```

<u>PSWAIT</u>

```
        JSR     PC,PSWAIT
```

BLDCON

```
        MOV    #ADR,R5
        JSR    PC,BLDCON
ADR:    BR     .+6.
        .WORD ITYPE
        .WORD IARRAY
```

P$DMA

```
        R0 = Repeat Status Register (RSR) Value
        R1 = DMA Word Count
        R2 = DMA Base Address
        JSR  PC,P$DMA
```

I$MATX

```
        JSR    PC,I$MATX
```

ERROR

```
        JSR PC,ERROR
        .BYTE ICODE,IERR
```

P$DIV

```
        R0,R1 = Dividend
        R2    = Divisor
        JSR  PC,P$DIV
```

P$MUL

```
        R0 = Multiplicand
        R2 = Multiplier
        JSR  PC,P$MUL
```

APPENDIX C

PDP-11 FORTRAN CALLING SEQUENCE CONVENTION

C.1 INTRODUCTION

This calling sequence convention is compatible with all PDP-11 processor options, (including use of distinct Instruction and Data Space capabilities of the KT-11D Memory Management Option), provides both reentrant and non-reentrant forms, and is as fast and short as possible, consistent with these requirements.

This description is oriented toward the programmer who wishes to write assembly language routines which can be called by or which call FORTRAN-compiled routines. This calling convention is completely compatible with the Threaded Polish code of the FORTRAN Compiler V06, though the assembly language programmer need not be concerned with or use the Polish technique or service routines.

C.2 THE CALL SITE

The basic form of the non-reentrant out of line call is:

```
        ;  INSTRUCTION SPACE
        MOV       #LIST,R5         ;ADDRESS OF ARGUMENT LIST
                                   ;TO REGISTER 5
         .
         .
         .
        ;  IN DATA SPACE
LIST:   .BYTE     N,0              ;NUMBER OF ARGUMENTS
        .WORD     ADR1             ;FIRST ARGUMENT ADDRESS
         .
         .
         .
        .WORD     ADRN             ;N'TH ARGUMENT ADDRESS
         .
         .
         .
ADR1:   .WORD      1               ;FIRST ARGUMENT
         .         .                   .
         .         .                   .
         .         .                   .
ADRN:   .WORD     N                ;N'TH AEGUMENT
```

C.3 RETURN

Control is returned to the calling program by restoring (if necessary) the stack pointer to its value on entry and executing:

```
RTS    PC
```

C.4 RETURN VALUE TRANSMISSION

FORTRAN FUNCTION subprograms will return the function value in general register R0 through R3 as appropriate to the type as follows:

BYTE (LOGICAL*1), LOGICAL, INTEGER	R0
REAL	R0, R1
DOUBLE PRECISION REAL COMPLEX	R0, R1, R2, R3

The only difference between a SUBROUTINE subprogram and a FUNCTION subprogram is that a FUNCTION returns a value in the general registers.

C.5 CONTEXT SAVE AND RESTORE CONVENTION

A calling program must save any values in general purpose registers R0 through R4, which it requires after a return from a subprogram. The argument list pointer value in register R5 may _not_ be assumed to be valid after return.

C.6 NON-REENTRANT EXAMPLE

In non-reentrant forms, the argument list may either be placed in line with the call or be placed out of line in an impure data section. (The latter is recommended and illustrated here.) Figure C-1 illustrates the assembly language code to implement a small FORTRAN FUNCTION subprogram using the non-reentrant form of call. Note that the non-reentrant form, is shorter and generally faster than the reentrant form since addresses of simple variables can be assembled into the argument list.

```
        INTEGER FUNCTION FNC(I,J)
        INTEGER FNC1

        FNC=FNC(I+J,5)+I

        RETURN
        END

        .CSECT
        .GLOBL      FNC,FNC1
FNC:    MOV         R5,-(SP)          ;SAVE ARG LIST POINTER
        MOV         @2(R5),-(SP)      ;FORM I+J ON STACK
        ADD         @4(R5),@SP
        MOV         SP,LIST+2         ;ADDRESS OF I+J TO
                                      ;ARG LIST
        MOV         #LIST,R5
        JSR         PC,FNC1
        ADD         #2,SP             ;DELETE TEMPORARY I+J
        MOV         (SP)+,R5          ;RESTORE R5
        ADD         @2(R5),R0         ;ADD I TO FNC1 RESULT
        RTS         PC                ;RETURN VALUE IN R0
                                      ;DATA AREA
LIST:   .BYTE       2,0               ;TWO ARGUMENTS
        .WORD       0                 ;DYNAMICALLY FILLED IN
        .WORD       LIT5              ;ADDRESS OF CONSTANT 5
LIT5:   .WORD       5                 ;CONSTANT 5
        .END
```

Figure C-1

Example Call Sequence Convention Usage: Non-Reentrant

C.7 REENTRANT EXAMPLE

The PDP-11 FORTRAN calling convention also has a reentrant form in which the argument list is constructed at run-time on the execution stack. Note that the argument addresses must be pushed on the stack backwards in order to be correctly arranged in memory for the subroutine that references the list. Basically it consists of:

```
        MOV          #ADRn,-(SP)      ;ADDRESS OF NTH ARGUMENT
         .
         .
         .
        MOV          #ADR2-(SP)
        MOV          #ADR1-(SP)       ;ADDRESS OF 1ST ARGUMENT
        MOV          #N,-(SP)         ;NUMBER OF ARGUMENTS
        MOV          SP,R5
        JSR          PC,SUB           ;CALL SUBROUTINE
        ADD          #2*N+2,SP        ;DELETE ARGUMENT LIST
```

Figure C-2 illustrates assembly language code using reentrant call forms for the same example shown in Figure C-1.

```
        INTEGER FUNCTION FNC (I,J)
        INTEGER FNC1

        FNC=FNC1 (I+J,5)+I

        RETURN
        END

        .SCECT
        .GLOBL       FNC,FNC1
FNC:    MOV          R5,-(SP)
        MOV          @2(R5),-(SP)     ;SAVE ARG LIST POINTER
        ADD          @4(R5),-@SP      ;FORM I+J
        MOV          SP,R4            ;REMEMBER WHERE
        MOV          #CON5,-(SP)      ;BUILD ARG LIST ON STACK
        MOVV         R4,-(SP)         ;ADDRESS OF TEMPORARY
        MOV          #2,-(SP)         ;ARGUMENT COUNT
        MOV          SP,R5            ;ADDRESS OF LIST TO R5
        JSR          PC,FNC1          ;CALL FNC1
        ADD          #10,SP           ;DELETE ARG LIST AND TEMP I+J
        MOVV         (SP)+,R5         ;RESTORE ARG LIST POINTER
        ADD          @2(R5),R0        ;ADD I TO RESULT OF FNC1
        RTS          PC               ;RETURN RESULT IN R0
                                      ;DATA AREA
CON5:   .WORD        5
        .END
```

Figure C-2
Example Call Sequence Convention Usage: Reentrant Form

Note that the list must reside in Data-space and that except for label type arguments, all addresses in the list must also refer to Data-space.

Also note that the byte at address LIST+1 should be considered undefined and not referenced. (Use of this byte is reserved for use as defined by DEC.)

The basic form of the non-reentrant in line call is:[1]

```
             ;  IN INSTRUCTION/DATA SPACE
             MOV       #LIST,R5        ;ADDRESS OF ARGUMENT LIST
                                       ;TO REGISTER 5
             JSR       PC,SUB          ;CALL SUBROUTINE
LIST:        BR        .+2*N+2         ;BRANCH AROUND
                                       ;PARAMETER LIST
             .WORD     ADR1            ;FIRST ARGUMENT ADDRESS
               .
               .
               .
             .WORD     ADRN            ;N'TH ARGUMENT ADDRESS
               .
               .
               .
             ;  IN DATA SPACE
ADR1:        .WORD      1              ;FIRST ARGUMENT
 .             .
 .             .
 .             .
ADRN:        .WORD      N              ;N'TH ARGUMENT
```

Note that the byte at address LIST will contain the value N and that the byte at address LIST+1 will contain the Value 1.

[1]This form of call is not compatible with distinct use of Instruction and Data Space Capabilities.

C.8 NULL ARGUMENTS

Null arguments are represented in an argument list by using an address of -1 (177777 octal). This address is chosen because it is easy to test for and also to assure that the use of null arguments, in subroutines that are not prepared to handle them, will result in an error when the routine is called at execution time. The errors most likely to occur are illegal memory reference and/or word reference to odd byte address.

Note that null arguments are included in the argument count as shown in Figure C-3.

FORTRAN Statement	Resulting Argument List
CALL SUB	.BYTE 0,0
CALL SUB ( )	.BYTE 1,0 .WORD -1
CALL SUB (A,)	.BYTE 2,0 .WORD A .WORD -1
CALL SUB (,B)	.BYTE 2,0 .WORD -1 .WORD B

Figure C-3

Example Argument Lists with Null Arguments

APPENDIX D

USE OF THE GRAPHICS SOFTWARE WITH THE PAPER TAPE SOFTWARE SYSTEM

D.1 DESIGN AND USE OF THE PAPER TAPE GRAPHICS SOFTWARE PACKAGE

The Paper Tape Graphics Software Package was designed to execute in a minimal memory configuration and yet provide user flexibility in using only those subroutines necessary for a particular application program, allowing a maximum memory availability for the application program and data base. This was done in the following manner:

> THE PICTURE SYSTEM initialization subroutine (PSINIT) is written as an absolute program to be loaded at a fixed location in memory. This subroutine contains <u>all</u> the system level software required to interface to THE PICTURE SYSTEM, as well as all global constants and variables that are used for intercommunication between subroutines. Since PSINIT is written as an absolute routine, all references to these global constants and variables may be made to an absolute location.
>
> All other subroutines are in position independent code[1] (i.e., may be loaded and executed anywhere in memory). This allows a user to load only those subroutines necessary for a particular application by utilizing a feature of the PDP-11 Absolute Loader[2]. This feature is the ability to load a routine from the last location loaded previously by the loader. Using this technique, the user may load those routines necessary in any order, ensuring that the minimum core required will be taken.

All PICTURE SYSTEM Subroutines must have a Transfer Vector linkage of the form shown in Figure D-1.

---

[1]Reference PDP-11 Paper Tape Software Programming Handbook DEC-11-XPTSA-A-D,Chapter 6.

[2]Ibid, Chapter 9.1.

1st word:	167754_8 (subroutine identifier) (="PI+"CS+"YS)
2nd word:	Relative location of last word of routine from 1st word
3rd word:	Subroutine identifier (=relative location to PSINIT in Transfer Vector)
4th word:	Relative location of subroutine entry point from 1st word
nth word:	-1 (if last entry point, otherwise same as words 3 and 4 above for subroutines with multiple entry points[1].)

Figure D-1

PICTURE SYSTEM Graphics Software Package
Subroutine Heading Format

---

[1]Reference listing of subroutine CHAR for example.

D.2 ERRORS USING THE BASIC TAPE SOFTWARE PACKAGE

Errors that occur during use of the Paper Tape PICTURE SYSTEM Graphics Software Package may be of two types:

a. User Software Errors
b. Equipment Failure (Hardware Errors)

The conditions that may cause these errors are as follows:

User Software Errors

A user may make five programming errors that will be detected by the Graphics Software Package. These are:

1. The call of a routine which has not been loaded (or loaded properly) into memory.
2. The call of a routine with an invalid number of parameters specified.
3. The call of a routine with an invalid parameter value.
4. The attempt by a user to PUSH a transformation to a depth greater than that specified by the user.
5. The attempt by a user to POP a matrix that had not been previously PUSHed.

Error detection results in the following for errors 1-5 above:

Error 1:
Upon the call of a routine which has not been loaded into memory, a halt will occur at location 276 (showing 300 in the console data lights). The user then may determine the origin of the "invalid" call by examining the last element on the stack as pointed to by Register 6 (SP).

Error 2:
Upon the call of a routine with an invalid number of parameters, the user's error routine (as specified in call to PSINIT) will be called using the standard FORTRAN calling sequence and a parameter indicating the origin of the error detected (see Figures D-2 and D-3) will be passed. If the user's error routine has not yet been established, then a halt will occur at location 276[1] (showing 300 in the console data lights). The error code may then be determined by examining the location pointed to by the second from last element on the stack (SP-4) as pointed to by Register 6 (SP). Return from the user's error routine will result in a halt occuring at location 276.

---

[1]It should be noted that if the user's error routine has not yet been established by PSINIT, then the programmer will be unable to discern the difference between Error 1 and Errors 2,3,4,5 without a detailed knowledge of the position of the routines in memory.

FIGURE D-2

SUBROUTINE INFORMATION

	Subroutine Name	Vector Offset$_{10}$	Length[1] Bytes$_{10}$	Length[1] Bytes$_{8}$	Stack[2] Space Required	Registers Destroyed	Error Codes & Meaning
1	PSINIT	0	1846	3466	30	None	1,0-Invalid No. of Parameters 1,1-Invalid Parameter 1,2-Direct Memory Access Error
2.	NUFRAM	4	(1)	(1)	2	None	None
3.	VWPORT	8	(1)	(1)	16	None	3,0-Invalid No. of Parameters
4.	WINDOW	12	498	762	38	None	4,0-Invalid No. of Parameters
5.	INST	16	(4)	(4)	36	None	5,0-Invalid No. of Parameters
6.	PUSH	20	(1)	(1)	20	None	6,0-PUSH Error
7.	POP	24	(1)	(1)	20	None	7,0-POP Error
8.	TRAN	28	150	226	28	None	8,0-Invalid No. of Parameters
9.	ROT	32	386	602	30	None	9,0-Invalid No. of Parameters 9,1-Invalid Parameter
10.	SCALE	64	138	212	28	None	17,0-Invalid No. of Parameters
11.	DRAW2D	36	260	404	20	None	10,0-Invalid No. of Parameters 10,1-Invalid Parameter
12.	DRAW3D	40	(11)	(11)	20	None	11,0-Invalid No. of Parameters 11,1-Invalid Parameter
13.	TEXT	44	188	274	18	None	12,0-Invalid No. of Parameters
14.	TABLET	48	188	274	16	None	13,0-Invalid No. of Parameters
15.	CURSOR	52	440	662	76	None	14,0-Invalid No. of Parameters
16.	HITWIN	56	276	422	32	None	15,0-Invalid No. of Parameters
17.	HITEST	60	(16)	(16)	20	None	16,0-Invalid No. of Parameters
18.	PSWAIT	72	(1)	(1)	2	None	None
19.	CHAR	76	258	402	16	None	18,0-Invalid No. of Parameters
20.	DASH	80	(19)	(19)	16	None	19,0-Invalid No. of Parameters
21.	BLINK	84	(19)	(19)	16	None	20,0-Invalid No. of Parameters
22.	SCOPE	88	(19)	(19)	16	None	21,0-Invalid No. of Parameters
23.	SETBUF	68	80	98	2	None	22,0-Invalid No. of Parameters 22-1-Invalid Parameter

----

[1]The numbers in these columns within parenthesis (i.e.,(1)) indicate that the subroutine is included as part of the subroutine whose number is in parenthesis.

[2]This column indicates the number of bytes of stack space that must be available when this subroutine in called (includes the call).

FIGURE D-3

SYSTEM LEVEL SUBROUTINE INFORMATION

Subroutine Name	Vector Offset$_{10}$	Length[1] Bytes$_{10}$	Length[1] Bytes$_{8}$	Stack[2] Space Required	Registers Destroyed	Error Codes & Meaning
BLDCON (1)	-4	(1)	(1)	18	None	0,0-Invalid No. of Parameters 0,1-Invalid Parameter
R$TORE	-8	(1)	(1)	2	R0-R5	None
P$AVE	-12	(1)	(1)	14	None	None
I$MATX	-16	(1)	(1)	2	R0,R1,R2	None
P$DMA	-20	(1)	(1)	2	None	Halt at Location 272 Direct Memory Access Error
ERROR	-24	(1)	(1)	6	None	Branch to User Error Routine or Halt at Location 276
P$DIV	-28	(1)	(1)	12	R0,R1	Overflow set on Error
P$MUL	-32	(1)	(1)	8	R0,R1	None

----

[1]The numbers in these columns within parenthesis (i.e.,(1)) indicate that the subroutine is included as part of the subroutine whose number is in parenthesis.

[2]This column indicates the number of bytes of stack space that must be available when this subroutine is called (includes the call).

Error 3:
Same as Error 3 (Parameter error)

Error 4:
Same as Error 2 (Push error)

Error 5:
Same as Error 2 (Pop error)

Equipment Failure (Hardware Errors)
detection of hardware errors is to a minimal level, within the Graphics Software Package. The only error that may be detected is a DMA error which will result in a halt occuring at location 272 (showing 274 in the console data lights) If this occurs, it indicates a failure in the Digital Equipment Corporation DR-11B DMA unit. However, other errors may occur as a result of a hardware malfunction or a general programming problem. These errors will result in a halt occuring at a location in memory. These halt locations are summarized in Figure D-4.

HALT LOCATION$_8$	ERROR TYPE
000006 (000010)[1]	Time Out
000012 (000014)[1]	Illegal & Reserved Instructions
000016 (000020)[1]	BPT
000022 (000024)[1]	IOT
000026 (000030)[1]	Power Fail/Auto Restart
000032 (000034)[1]	EMT
000036 (000040)[1]	TRAP
000272 (000274)[1]	DMAError
000276 (000300)[1]	Non-Existant Program Error

FIGURE D-4

Paper Tape PICTURE SYSTEM Graphics Software Package
Halt Locations

----
[1]Location (xxxxxx) is the location shown in the console data lights when the halt occurs.

D.3 PROGRAMMING THE PICTURE SYSTEM USING THE PAPER TAPE SOFTWARE PACKAGE

Programs written for use with the Paper Tape Software Package use the same general program structure and techniques as described in Chapter 5. The user, however, has the additional responsibility of:

1. Defining the linkage to the graphics subroutines.
2. Initializing the program stack pointer (register 6) to the reserved stack area.
3. Ensuring that the program does occupy the same area of memory as the graphics software.

The linkage to the graphics subroutines is provided by equating the entry in the transfer vector to the subroutine name as defined in Figure D-2. This method allows the subroutines to be referenced symbolically and also make the program upward compatible with all DEC operating systems by simply replacing the equate with a global symbol definition (.GLOBL). Figure D-5 illustrates the manner in which the transfer vector entries are equated with the graphics subroutines.

An area of memory is reserved for the program stack area beginning at 620_8 and extending through 400 in memory as shown in Figure D-6. The stack pointer to this area must, however, be initialized by the user's program before any subroutines are called or any interrupts occur. Figure D-5 shows a typical manner in which this may be done.

The user must ensure that his program's starting address does not overlap an area of memory where a graphics subroutine resides. To do this the user must total the lengths of all of the graphics subroutines used (Figure D-2) and position his program above that area of memory by using the ".= start address" notation of the DEC assemblers. Figure D-5 illustrates this.

Except for these three additional responsibilities, the user is free to utilize all of the capabilities of the graphics subroutines without constraint in the Paper Tape environment.

---

[1]See eference 3, Part 3, Chapter 2, Monitor Keyboard Commands, for specific details.

```
        T$VECT      =1000.
        PSINIT      =T$VECT+0.
        NUFRAM      =T$VECT+4.
        VWPORT      =T$VECT+8.
        WINDOW      =T$VECT+12.
        INST        =T$VECT+16.
        PUSH        =T$VECT+20.
        POP         =T$VECT+24.
        ROT         =T$VECT+32.
        TRAN        =T$VECT+28.
        SCALE       =T$VECT+64.
        DRAW2D      =T$VECT+36.
        DRAW3D      =T$VECT+40.
        TEXT        =T$VECT+44.
        TABLET      =T$VECT+48.
        CURSOR      =T$VECT+52.
        HITWIN      =T$VECT+56.
        HITEST      =T$VECT+60.
        PSWAIT      =T$VECT+72.
        CHAR        =T$VECT+76.
        DASH        =T$VECT+80.
        BLINK       =T$VECT+84.
        SCOPE       =T$VECT+88.
        SETBUF      =T$VECT+68.
;
        BLDCON      =T$VECT-4.
;
        STACKP      =620
;
        .=7770                          ;   SET THE PROGRAM START ADDRESS
;
STOP:   TST         (R5)+               ;   A USER ERROR SUBROUTINE
        MOV         @(R5),R0            ;   MOVE THE ERROR CODE TO R0
        HALT                            ;   AND HALT
;

START:  MOV         #STACKP,SP
;
;   INITIALIZE THE PICTURE SYSTEM
;
        MOV         #.+8.,R5
        JSR         PC,PSINIT
        BR          .+14.
        .WORD       THREE,ZERO,-1,STOP,-1,-1
          .            .
          .            .
          .            .
```

Figure D-5
User Responsibilities in the Paper Tape Software System

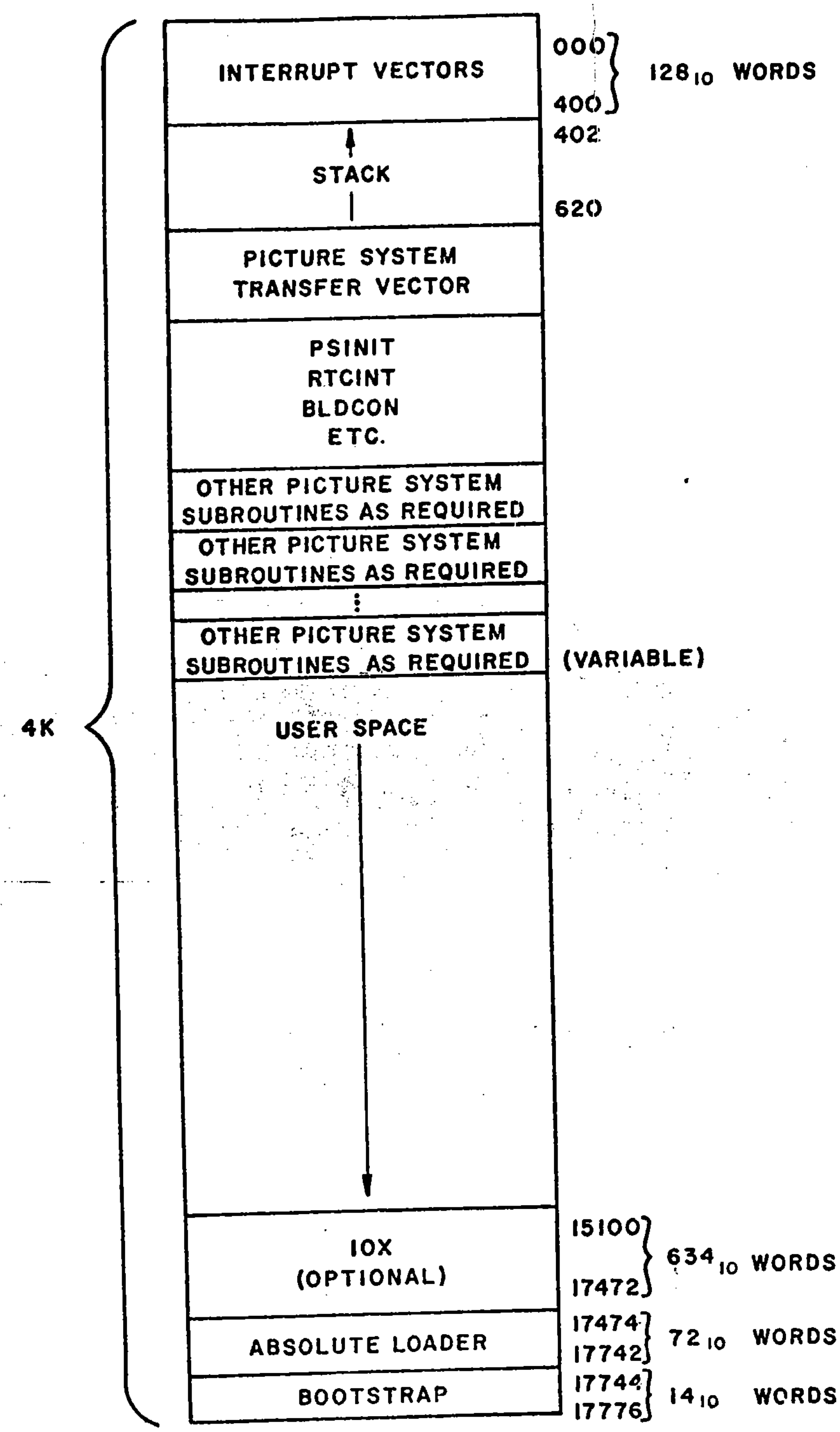


Figure D-6

Typical PICTURE SYSTEM Paper Tape Memory Configuration (4K)

APPENDIX E

USE OF THE GRAPHICS SOFTWARE WITH THE DOS/BATCH DISK OPERATING SYSTEM

E.1 USE OF THE GRAPHICS SOFTWARE PACKAGE

The Graphics Software Package is available to the DOS/BATCH user as a library of catalogued object Modules which may be linked with the user's FORTRAN[1] or MACRO-11[2] Program to form graphics application programs. The PICTURE SYSTEM Graphics Library (PICLIB), which contains all of the subroutines described in Chapter 4, is searched by the linker (LINK)[3] to load those subroutines called by the user program. The resulting program forms a load module (LDA format) which may be executed upon user demand.

---

[1] DOS/BATCH FORTRAN Compiler and Object Time System, Reference 3, Part 7.
[2] DOS/BATCH Assembler (MACRO), Reference 3, Part 6.
[3] DOS/BATCH Linker (LINK), Reference 3, Part 9.

E.2 USE OF PDP-11 FORTRAN IV WITH THE PICTURE SYSTEM

DOS/BATCH FORTRAN conforms to the specifications for American National Standard FORTRAN and is also highly compatable with IBM 1130 FORTRAN. DOS/BATCH FORTRAN programs can be compiled and run on any PICTURE SYSTEM configuration that support the DOS/BATCH Operating System, and which has a minimum of 16K of memory. DOS/BATCH FORTRAN supports all standard hardware options supported by the operating system.

Graphics applications programs written using FORTRAN interface to THE PICTURE SYSTEM by means of the subroutines contained in the Graphics Library (PICLIB). All FORTRAN statements and functions are available to the user of THE PICTURE SYSTEM; however, the following should be stressed to the PICTURE SYSTEM FORTRAN user:

1. All parameters passed to the subroutines of the Graphics Library are integers. Should a REAL parameter be passed as a parameter to a graphics subroutine, the sign, binary excess 128 exponent and high-order mantissa will be treated as an integer.

2. The "one word integers" switch (/ON) should be specified to the FORTRAN compiler to ensure that the elements of integer arrays are contiguous in memory as required by the graphics software.

Figure E-1 outlines the steps required to prepare a FORTRAN source program for execution under the DOS/BATCH monitor: (1) Compilation, (2) Linking and (3) Execution.

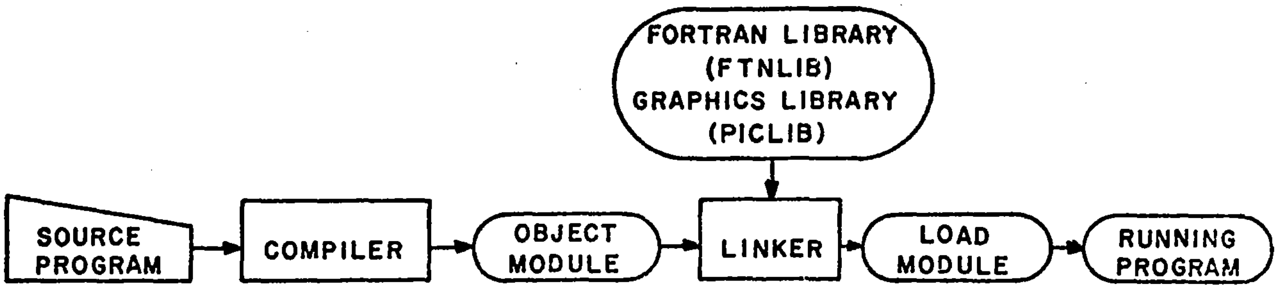


Figure E-1

Steps in Compiling and Executing
a FORTRAN Graphics Program

Step 1 in Figure E-1 is initiated by a call to the FORTRAN Compiler, accompanied by a command string that describes input and output files, and switch options to be used by the Compiler. Step 2 is initiated by a call to the Linker, accompanied by a similar command string. Step 3 is initiated upon user keyboard request or a user programmed request.

<u>Step 1</u>[1]: The DOS/BATCH FORTRAN compiler accepts a standard DOS command string of the form:

```
#object module, listing < source/options
```

A typical FORTRAN command string is of the form:

```
#SY:PROG1.OBJ,SY:PROG1.LST < SY:PROG1.FTN/ON
              or
#PROG1,PROG1 < PROG1/ON
```

(device SY: assumed the default device, just as the filename extentions .OBJ, .LST and .FTN are the default filename extentions when not specified.)

In the above example, the user should note the use of the "one word integers" switch (/ON) in the source file specification: < PROG1/ON.

---

[1]See Reference 3, Part 7, Chapter 9, Operating Procedures, for specific details.

Step 2[1]: The DOS/BATCH Linker accepts a standard DOS command string of the form:

```
#load module,load map,symbol table < object modules/E
```

A typical LINK command string is of the form:

```
#SY:PROG1.LDA,SY:PROG1.MAP,SY:PROG1.STB < SY:PROG1.OBJ
#SY:PICLIB.OBJ,SY:FTNLIB.OBJ/E
                 or
#PROG1,PROG1,PROG1<PROG1,PICLIB,FTNLIB/E
```

(device SY:is assumed to default device, just as the filename extentions .LDA, .MAP, .STB and .OBJ are the default filename extensions when not specified.)

In the above example, the user should note the specification of THE PICTURE SYSTEM Graphics Library (PICLIB) and the FORTRAN OTS Library (FTNLIB). These libraries are seached to resolve all global references for the load module. These libraries (PICLIB) and (FTNLIB) reside in the systems area [1,1] and are therefore available to all users. Note: The Linker searches the user's [UIC] area for all object files specified. If an object file is not found, the system area [1,1] is searched regardless of the user UIC.

Step 3[2]: To run a load module which has been created by the Linker, a user need only request the monitor to run the program. This is accomplished by the monitor command:

```
$RUN SY:PROG1.LDA
      or
$RUN PROG1
```

(device SY: is assumed the default device, just as the filename extention .LDA is the default filename extention when not specified.)

The following is a typical listing which illustrates the process described by Figure E-1 and steps 1, 2 and 3 above.

---

[1]See Reference 3, Part 9, Chapter 3, Operating Procedures, for specific details.
[2]See Reference 3, Part 3, Chapter 2, Monitor Keyboard Commands.

```
$LOG 102,113
DATE:-31-MAY-74
TIME:-15:35:05
$RUN FORTRN
FORTRAN  V06.13
#PROG1,KB:<PROG1/ON

FORTRAN V06.13                        15:35:44     31-MAY-74    PAGE     1

        C FORTRAN DEMONSTRATION PROGRAM
        C
0001            DIMENSION IHOUSE(14)
        C
0002            DATA IHOUSE/-10000,10000,-10000,-10000,10000,-10000,10000
                1 ,10000,-10000,10000,0,20000,10000,10000/
        C
        C INITIALIZE THE PICTURE SYSTEM
        C
0003            CALL PSINIT(3,0,,,,)
        C
        C DRAW THE DATA
        C
0004            CALL DRAW2D(IHOUSE,7,2,2,0)
        C
        C AND DISPLAY THE "NEW FRAME"
        C
0005            CALL NUFRAM
        C
0006            PAUSE
        C
0007            STOP
0008            END

        ROUTINES CALLED:
        PSINIT, DRAW2D, NUFRAM

        OPTIONS =/ON,/OP:2

        BLOCK       LENGTH
        MAIN.    67     (000206)*

        **COMPILER ----- CORE**
           PHASE       USED  FREE
        DECLARATIVES 00622 10228
        EXECUTABLES  00702 10148
        ASSEMBLY     00889 14601

#^C
.KILL
```

```
$RUN LINK

LINK V01-03
#PROG1,PROG1<PROG1,PICLIB,FTNLIB/E



SPACE USED 005530, SPACE FREE 063304
#^C
.KILL

$RUN PROG1
```

APPENDIX F

USE OF THE GRAPHICS SOFTWARE WITH THE RT-11 OPERATING SYSTEM

F.1 USE OF THE GRAPHICS SOFTWARE PACKAGE

The Graphics Software Package is available to the RT-11 user as a library of catalogued object Modules which may be linked with the user's FORTRAN[1] or MACRO-11[2] program to form graphics application programs. THE PICTURE SYSTEM Graphics Library (PICLIB), which contains all of the subroutines described in Chapter 4, is searched by the Linker (LINK)[3] to load those subroutines called by the user program. The resulting program forms a load module (SAV format) which may be executed upon user demand.

---

[1] Reference 4.
[2] Reference 5, Chapter 5.
[3] Reference 5, Chapter 6.

F.2 USE OF PDP-11 FORTRAN IV WITH THE PICTURE SYSTEM

RT-11 FORTRAN conforms to the specifications for American National Standard FORTRAN and is also highly compatible with IBM 1130 FORTRAN. RT-11 FORTRAN programs can be compiled and run of any PICTURE SYSTEM configuration that supports the RT-11 Operating System, and which has a minimum of 8K of memory. RT-11 FORTRAN supports all standard hardware options supported by the operating system.

Graphics applications programs written using FORTRAN interface to THE PICTURE SYSTEM by means of the subroutines contained in the Graphics Library (PICLIB). All FORTRAN statements and functions are available to the user of THE PICTURE SYSTEM: however, the following should be stressed to THE PICTURE SYSTEM FORTRAN user:

> All parameters passed to the subroutines of the Graphics Library are integers. Should a REAL parameter be passed as a parameter to a graphics subroutine, the sign, binary excess 128 exponent and high-order mantissa will be treated as an integer.

Figure F-1 outlines the steps required to prepare a FORTRAN source program for execution under the RT-11 Monitor: (1) Compilation, (2) Linking, and (3) Execution.

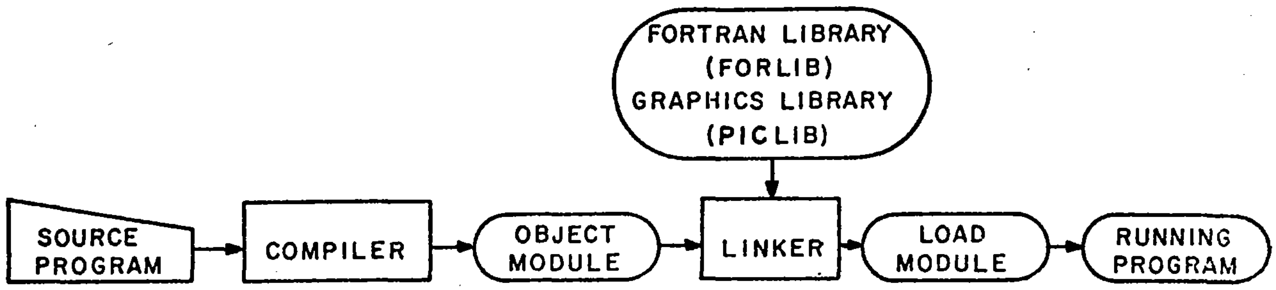


Figure F-1

Steps in Compiling and Executing
a FORTRAN Graphics Program

Step 1 in Figure F-1 is initiated by a call to the FORTRAN Compiler, accompanied by a command string that describes input and output files, and switch options to be used by the Compiler. Step 2 is initiated by a call to the Linker, accompanied by a similar command string. tep 3 is initiated upon user keyboard request or a user programmed request.

Step 1[1]: The RT-11 FORTRAN compiler accepts a command string of the form:

```
*object module, listing = source/options
```

A typical FORTRAN command string is of the form:

```
*SY:PROG1.OBJ,SY:PROG1.LST=SY:PROG1.FOR
                   or
*PROG1,PROG1=PROG1
```

(device SY: is assumed the default device, just as the filename extensions .OBJ, .LST and .FOR are the default filename extensions when not specified.)

[1]See Reference 4, Chapter 1 for specific details.

Step 2[1]: The RT-11 Linker accepts a command string of the form:

*load module,load map=object modules/switches

A typical LINK command string is of the form:

*SY:PROG1.SAV,SY:PROG1.MAP=SY:PROG1.OBJ,SY;PICLIB.OBJ/F
or
*PROG1,PROG1,PROG1=PROG1,PICLIB/F

(device SY: is assumed to be the default device, just as the filename extensions .SAV, .MAP and .OBJ are the default filename extensions when not specified.)

In the above example, the user should note the explicit specification of THE PICTURE SYSTEM Graphics Library (PICLIB) and the FORTRAN OTS Library (FORLIB) by the /F switch. These libraries are searched to resolve all global references for the load module.

Step 3[2]: To run a load module which has been created by the Linker, a user need only request the monitor to run the program. This is accomplished by the monitor command:

.RUN SY:PROG1.SAV
or
.RUN PROG1

(device SY: is assumed the default device, just as the filename extension .SAV is the default filename extension when not specified).

The following is a typical listing which illustrates the process described by Figure F-1 and steps 1, 2 and 3 above.

---

[1]See Reference 5, Chapter 6 for specific details.
[2]See Reference 5, Chapter 2 for specific details.

```
RT-11 V01-15I

.DATE 25-NOV-74

.RUN FORTRAN
*PROG1,TT:=PROG1

RT-11 FORTRAN IV          V01-11  SOURCE LISTING          PAGE 001

      C FORTRAN DEMONSTRATION PROGRAM
      C
0001        DIMENSION IHOUSE(14)
      C
0002        DATA IHOUSE/-10000,10000,-10000,-10000,10000,-10000,10000
           1 ,10000,-10000,10000,0,20000,10000,10000/
      C
      C INITIALIZE THE PICTURE SYSTEM
      C
0003        CALL PSINIT(3,0,,,,)
      C
      C DRAW THE DATA
      C
0004        CALL DRAW2D(IHOUSE,7,2,2,0)
      C
      C AND DISPLAY THE "NEW FRAME"
      C
0005        CALL NUFRAM
      C
0006        PAUSE
      C
0007        STOP
0008        END
RT-11 FORTRAN IV        STORAGE MAP

NAME    OFFSET  ATTRIBUTES

IHOUSE  000006  INTEGER*2 ARRAY (14)
PSINIT  000000  REAL*4    PROCEDURE
DRAW2D  000000  REAL*4    PROCEDURE
NUFRAM  000000  INTEGER*2 PROCEDURE
*
```

```
^C

.RUN LINK
*PROG1,PROG1=PROG1,PICLIB/F

*^C

.RUN PROG1
```