PICTURE SYSTEM 2

USER'S MANUAL

Evans & Sutherland Computer Corporation
580 Arapeen Drive
Salt Lake City, Utah 84108

First Edition May 13, 1977

All reference to this document should be made to:
Document: E&S #901129-001 NC

PREFACE

The PICTURE SYSTEM 2 USER'S MANUAL provides the information necessary for the user of PICTURE SYSTEM 2 to gain an understanding of the general principles of interactive computer graphics and to apply these principles using PICTURE SYSTEM 2. The chapters of this manual are arranged so that each chapter builds upon the information presented in the preceding chapters. In this manner, the reader with little or no prior knowledge of computer graphics is only presented information with adequate introduction, explanation or reference.

Chapter 2 contains an overview and discussion of the general terms and principles of interactive computer graphics. This overview is intended as an introduction to computer graphics, but may be augmented by referring to the material listed as references for that chapter.

Chapter 3, OVERVIEW OF PICTURE SYSTEM 2, describes the components and operations which comprise PICTURE SYSTEM 2 and how these are used to provide an interactive graphics system.

Chapter 4 describes the Graphics Software Package developed for PICTURE SYSTEM 2. This software package consists of a set of FORTRAN callable subroutines which allow comprehensive use of the system by the FORTRAN programmer. This chapter details the subroutine calling sequences and contains specific examples of how they may be used. This chapter is provided to guide the user in programming PICTURE SYSTEM 2 and contains numerous FORTRAN examples illustrating the way the graphics subroutines are most often called.

Chapter 5 contains several working samples of PICTURE SYSTEM 2 FORTRAN programs, illustrating the use of most of the subroutines and techniques discussed in Chapter 4.

Chapter 6 details each of the graphic subroutines referenced in Chapter 4, with the individual calling sequence specifications and errors that may be detected when using these subroutines. This chapter is intended as a reference chapter, with the subroutines listed in alphabetical order for easy user access. Each of the errors that may occur are summarized in Section 6.2.

This manual has been written assuming the reader has a background of programming at the FORTRAN level. For a further detailed discussion of subjects within the material presented, the reader should refer to the references listed.

The examples, tables, figures and specifications contained in this Manual relating to the refreshing of the Picture Display are based upon the assumption that PICTURE SYSTEM 2 is deriving its A.C. power from a 60 Hertz power source. All numbers used in this Manual are base 10 numbers, unless denoted otherwise.

The terms PICTURE SYSTEM 2 and PICTURE SYSTEM are used interchangeably throughout this manual.

TABLE OF CONTENTS

CHAPTER 4 PROGRAMMING PICTURE SYSTEM 2

CHAPTER ONE

1. INTRODUCTION

PICTURE SYSTEM 2 is a microprogrammed, 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. PICTURE SYSTEM 2 has the same digital hardware capabilities as the previous systems, but in addition, has increased throughput capabilities and picture buffering for refreshing the display. The internal Picture 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 available in PICTURE SYSTEM 2. The basic hardware processing components of the system are: a Picture Processor for performing such functions as rotation, zooming and perspective; a Picture Memory; a Picture Generator; a Character Generator; a 21" Picture Display; devices to facilitate picture interaction; and software to support the system.

CHAPTER TWO

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 allowing 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 is not considered interactive. In some systems, however, the time lag is a small fraction of a second, and the user feels he is actually manipulating the picture.

The purpose of this chapter is to present, in general terms, the concepts necessary for understanding and using PICTURE SYSTEM 2. Consequently, it devotes little discussion to some aspects of computer graphics which may be of interest and importance to some readers, but which are not prerequisites to understanding the remainder of this manual*.

The study of interactive graphics can be broken down into four major topic areas:

1. Presenting a prepared picture.
2. Representing structures to be depicted.
3. Preparing a picture of such structures.
4. Interacting with the picture.

Each of these areas will be explored in the following sections.

---

*"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 artificial way of presenting pictorial data. Computer graphics offers an output medium on which data can be presented visually.

2.1.1 Graphical Output Media

There are many types of devices available for graphical output. These devices may be grouped, by performance and interaction ability, to form a spectrum. At one end of this spectrum lies pen and ink plotters. For these devices, a computer-driven pen creates a stroke-by-stroke picture. Plotters are unmatched for resolution (a measure of the density of individually distinguishable output values), but are extremely slow compared to more sophisticated graphic devices. Figure 2.1-1 shows a plot produced by a PICTURE SYSTEM/drum plotter installation.

Next in the spectrum lies the electrostatic dot matrix printer/plotter. This plotter operates by "writing" elements of a dot matrix with ink. The pattern of filled and empty elements is assembled by the eye to form a picture when viewed from a reasonable distance. A plotter of this type is known as a "raster" device (i.e., a picture produced as a succession of horizontal lines). Plotters of this type are typically of a coarse resolution, but are capable of producing several hundred raster lines per minute. If, however, a picture consists of a series of line segments, these line segments must be converted to raster information--a sorting process which may require up to several minutes to complete. Figure 2.1-2 shows a plot produced by a PICTURE SYSTEM/electrostatic plotter installation.

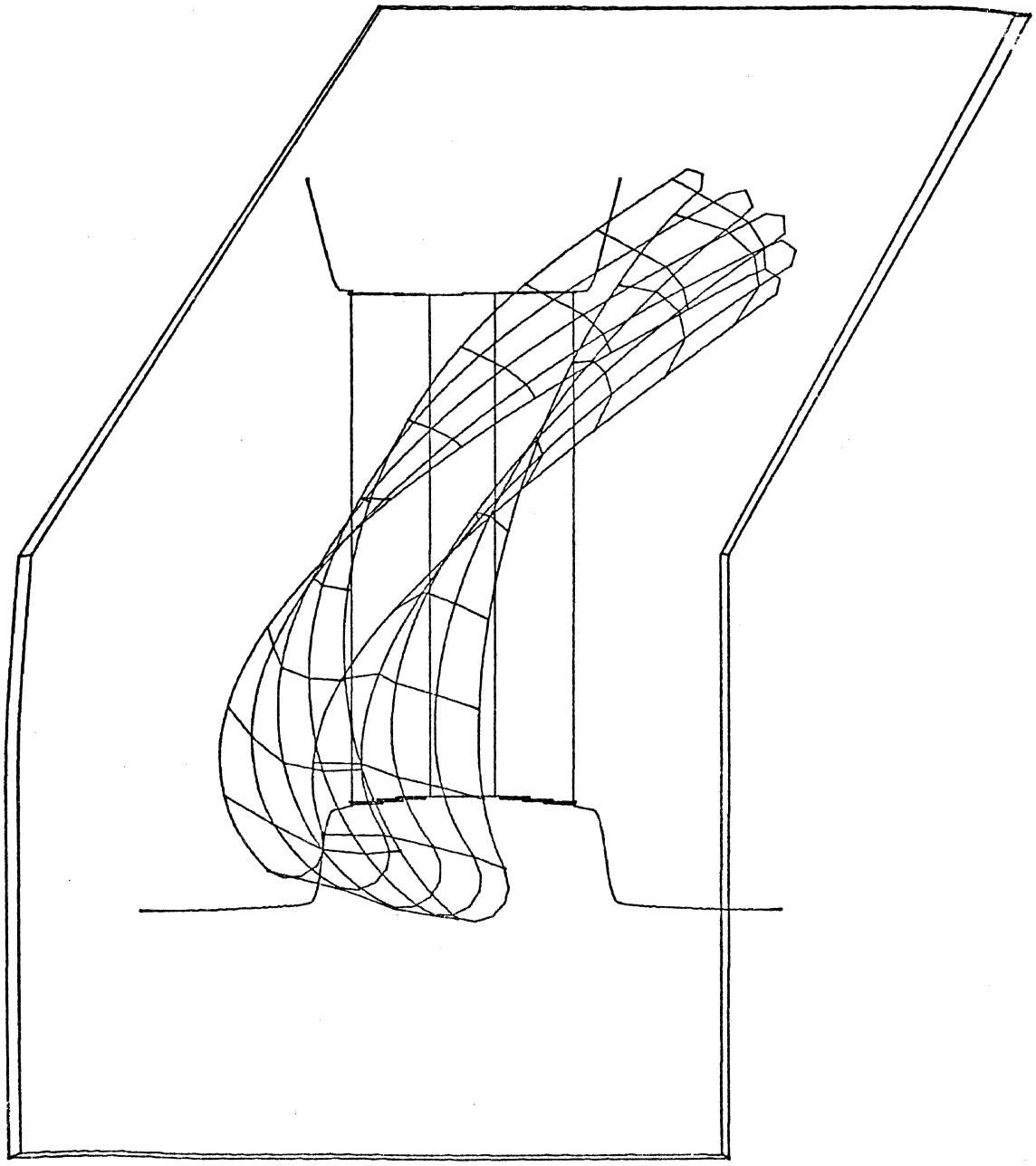


Figure 2.1-1

Plot Produced by PICTURE SYSTEM/Drum Plotter Installation

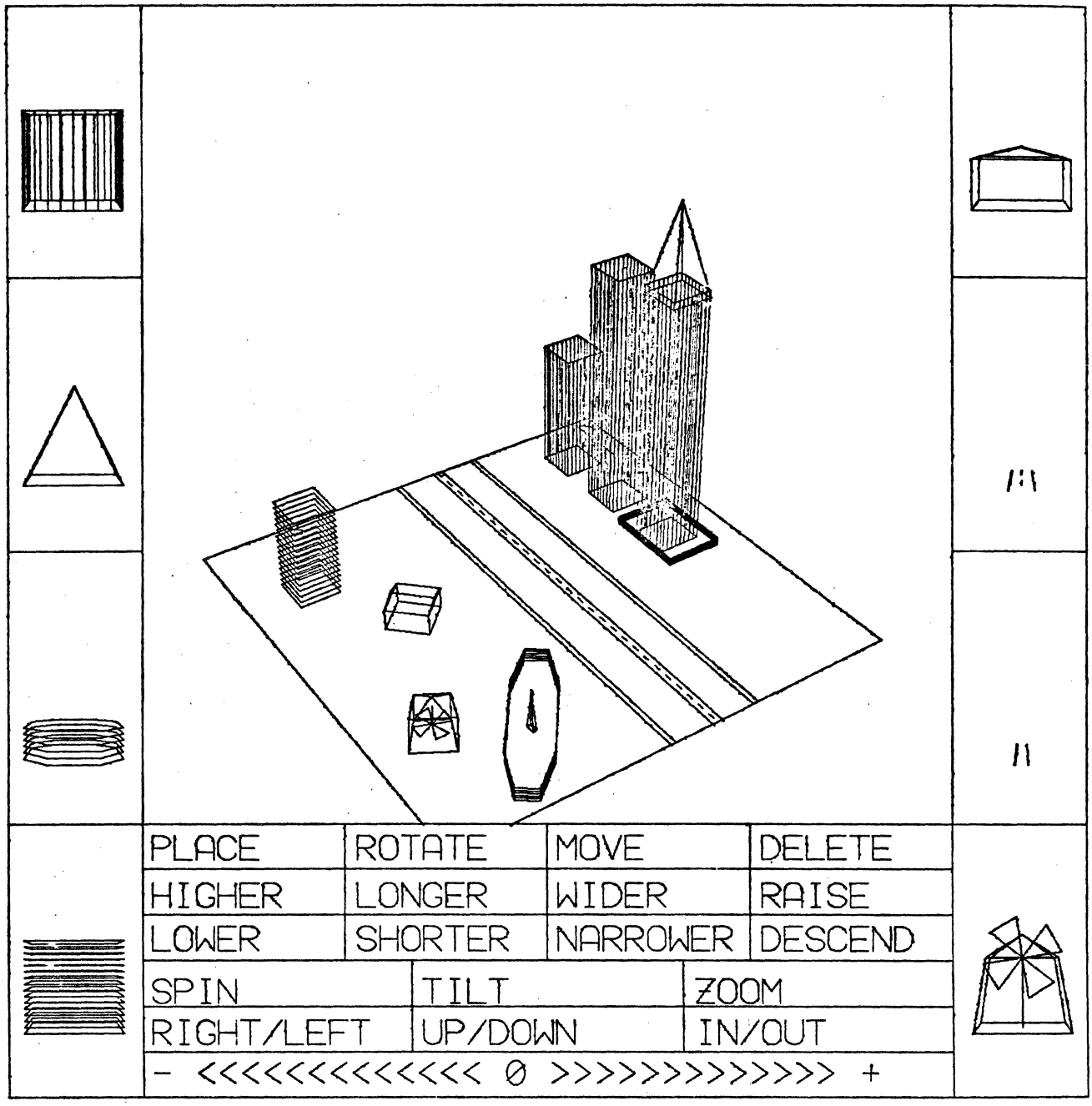


Figure 2.1-2

Plot Produced by PICTURE SYSTEM
Electrostatic Dot Matrix Plotter Installation

The previous two forms of output media are termed "hard copy", since the final image is produced on paper. Such output is permanent which can be a disadvantage if any interaction with the picture produced is desired. The cathode ray tube (CRT) has been used since the early development of computers to display non-permanent images. There are three basic types of CRTs:

1. Direct-View storage tube
2. Raster scan display
3. Calligraphic refresh display

These three types of displays fall in the performance-interaction spectrum from mid-range to high, respectively. Each of these types of CRTs function basically in the same manner; i.e., an electron beam is deflected by an electrostatic or electromagnetic field to the phosphor-coated face of the screen, causing the phosphor to glow. The length of time the phosphor glows after it is excited by the electron beam determines its persistence. Normally, the glow produced by the electron beam fades away shortly after the beam moves from the spot.

The direct-view storage tube behaves in a similar manner to a CRT with an extremely long persistence phosphor, since a line written onto the screen will remain visible for up to an hour before it fades away. The electron beam does not, however, write directly onto the phosphor, but rather charges a fine wire grid mounted behind the phosphor-coated face of the screen. The charge left on the grid is continuously transferred onto the screen by an unfocused field of electrons (distinct from the writing electron beam) which minimally affects the charge on the wire storage grid. The storage tube has the disadvantages of fairly coarse resolution and erasing the picture causes a momentary flash covering the entire screen. Also, selective portions of the screen cannot be erased, which limits the direct-view storage tube to graphical applications that are more passive than active in nature. A recent development for storage tubes is the ability to display data in "write through" or "refresh" mode. Although this allows the storage tube to be used in applications where refresh displays were formerly required, there are disadvantages associated with this device. The storage tube has no facility for intensity variation, a feature necessary whenever the illusion of depth is to be imparted to a three-dimensional object, and also necessary in many two-dimensional applications. Also, the amount of data which can be displayed in "write through" mode is rather limited. Typically only approximately 800 inches of flicker free vectors can be displayed each frame. Data displayed in "write through" mode also tend to be of a fainter intensity than the rest of the data on the display.

Also, the basic problems associated with storage tubes are still present, even with the enhanced features of this device.

The raster scan display is, in many instances, the same CRT used in the commercial television sets. This device functions in a similar manner to the electrostatic dot matrix printer/plotter previously described; dots (or pixels) in the form of a matrix are individually intensified as a succession of raster, or horizontal, lines which are assembled by the eye to form an image. A standard television image consists of 480 raster lines. Each line is divided into some number of dots, typically 512. Unlike the electrostatic printer plotter, the raster display must reproduce the entire picture each time the image on the phosphor fades--typically, 30 times a second. This requires that the raster information be stored in digital form. This quantity of data, however, rapidly becomes rather large with 245,760 bits of information required for a standard television image of only 1 intensity level. Almost 2 million bits of information are required to produce a gray scale image with 256 intensity levels or a color image containing 256 colors. Recently, many more raster display systems are beginning to be used. However, these systems are usually used in applications that are not suited to traditional CRT graphics systems. These include the portrayal of shaded or half-tone images or handling or enhancing "real" (scanned in) pictures. Raster systems are, in general, inappropriate for use in applications where an image must be changed rapidly. Figure 2.1-3 shows a picture of a raster graphics display generated using the Evans & Sutherland Video Frame Buffer.

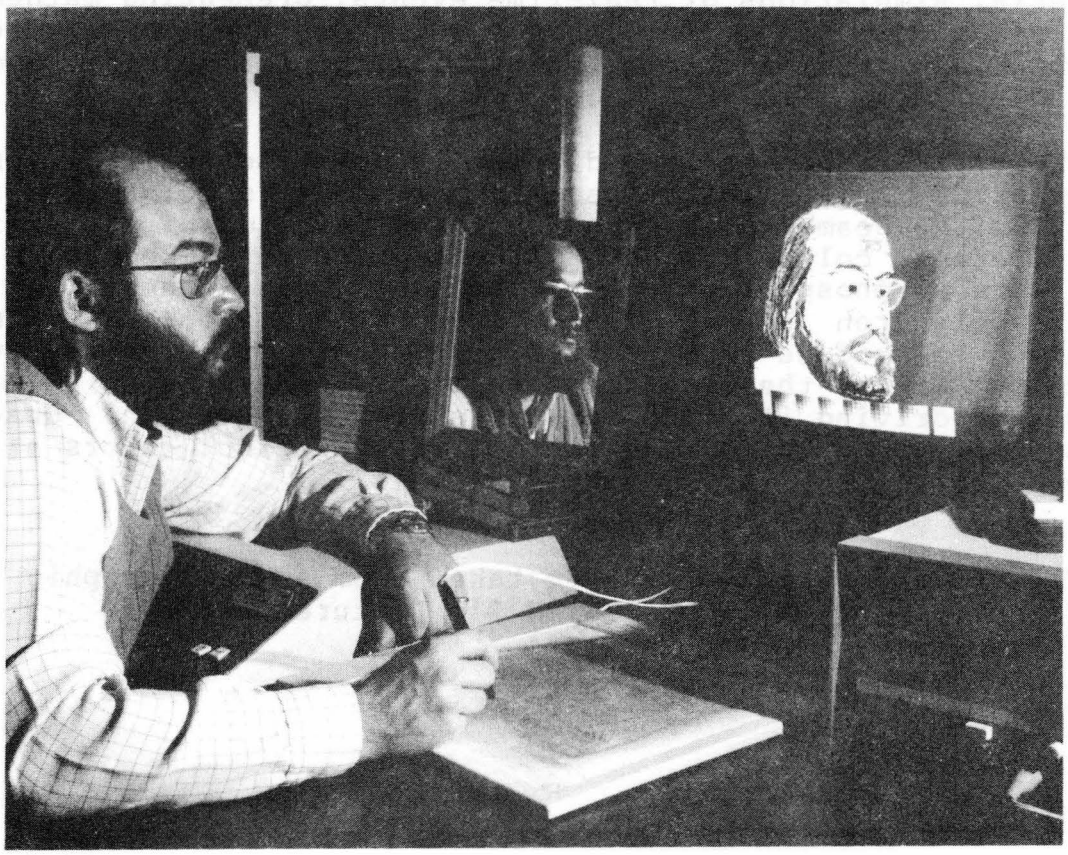


Figure 2.1-3

Picture Produced by Evans & Sutherland
Video Frame Buffer Raster Display System

The calligraphic refresh display is the most versatile of the CRTs. The term calligraphic means "fine writing" and is descriptive of the higher resolution available to display dots, characters and lines. Although dependent upon the capabilities of the graphics system with which it is used, the refresh display in general provides the ability to rapidly display information as a series of line segments. Thus, it is used in those applications where interaction between the person using the display device and the display is required. Calligraphic refresh displays are uniquely suited for applications such as Computer-Aided Design (CAD) where dynamic motion is desirable or required to fulfill the design criteria, simulations of real-time events, presenting mathematical models depicting things which could not otherwise be observed or interacted with (e.g., molecular models), etc.

The calligraphic refresh display is now also available as a color CRT. The monitor used to provide this color capability is the Beam-Penetration Monitor. This display provides different colors by coating the face of the screen with two layers of phosphor which emit red and green when excited by the electron beam. These phosphor emissions can then be mixed to produce four or five distinct colors. The device which drives the electron beam must, however, also be capable of driving it at different energy levels and drawing speeds to provide the four or five distinct colors at approximately the same intensity levels.

Figure 2.1-4 shows a picture taken from a calligraphic refresh display--PICTURE SYSTEM 2's Picture Display.

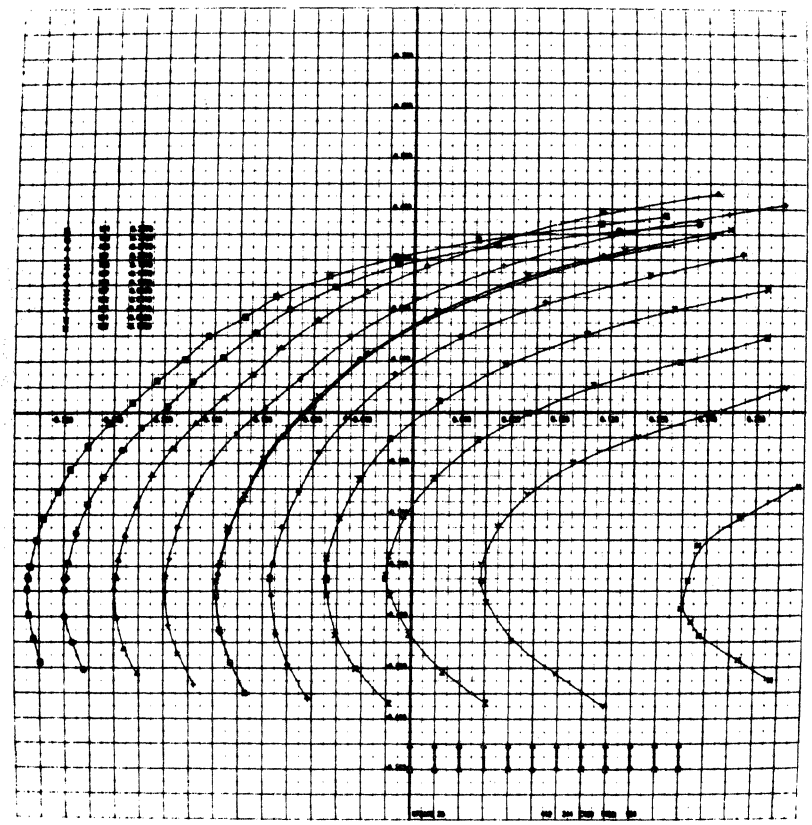


Figure 2.1-4

Picture Produced on a PICTURE SYSTEM Picture Display

2.1.2 Line Generation

A line is specified by two endpoints (x,y) and (x',y'), expressed in the coordinate system of the CRT, known as 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 sophisticated line generator is also capable of drawing lines with a program-specified intensity, or varying the intensity of a line from one end to the other. 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 be made to draw lines in a choice of modes and textures such as blinking, solid, long dashed, short dashed, etc. Line generators which can service more than one CRT are equipped with a facility for CRT selection. A display program may select one or more CRTs and to allow any subsequently drawn lines to appear on all selected CRTs.

2.1.3 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. The rate at which it is redrawn, the refresh rate, is usually measured in frames per second. If the picture is not redrawn frequently, the eye will notice fading between refreshes, producing an 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 frames-per-second. For example, pictures refreshed more than 30 frames-per-second appear flicker-free; pictures refreshed less than 30 frames-per-second flicker; and pictures refreshed exactly 30 frames-per-second are marginal.

2.1.4 Update Rate

The advantage of the calligraphic refresh display is that it can display smoothly-changing pictures. Lines drawn on a CRT do not 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 to-

gether into a smoothly-moving picture in much the same way it does with a motion picture. The rate at which these different frames can be displayed is known as 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 which are different. The update rate of a high-performance graphics system is directly dependent upon the time required to process the individual elements which describe the objects to be displayed.

2.1.5 Picture Buffering

A refresh buffer provides storage which allows the refresh and update rates to differ. Although a refresh of more than 30 frames per second is required to avoid flicker, an update rate 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 elements resident in a refresh buffer are referred to as elements of 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 each second and the picture is refreshed 40 times each 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. 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 which can be processed.

A potential problem exists when a picture is being refreshed from a memory which is concurrently being used in creating a new frame. The picture being displayed may consist of some lines from one frame and additional lines from another, producing a number of undesirable effects. To avoid this problem, the refresh buffer can be divided into two separate areas. The update and refresh can then be switched between the two areas in a way such that refresh and update are always performed in separate buffers. This method is called double-buffering, and its only disadvantage is the amount of refresh data which may be buffered is cut in half. In some cases, this can place an unnecessarily low ceiling on the line capacity. As pictures become less dynamic, single-buffering is usually sufficient, allowing the user to take full advantage of his memory space.

In many applications, only part of a picture changes dynamically, and therefore, that portion of the picture should be able to be updated independently. To enable this, the display file must be created as a series of segments. Thus, when only a part of a picture is to be changed, a segment is "opened" in a free area of the refresh buffer and the new data output. When the creation of the "new" segment is completed, the segment is "closed" and added to the data which is being refreshed as the "old" segment is deleted. This allows use of the refresh buffer in a double-buffered mode without requiring the refresh buffer to be divided in half. Only the size of the largest segment which is to be updated need be reserved for double-buffering. The creation and deletion of segments, however, could rapidly cause the refresh buffer to become fragmented if there were no way to compact the segments together. Therefore, a memory compacting method must be employed to reorganize the memory.

2.2 PICTURE DEFINITION

Data ultimately deposited in a refresh buffer must originate in the memory of the computer controlling the system. Such computer-resident data are called a data base. This data base may vastly differ in form from the display file which is generated.

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. Together with coordinate information, there may also be pointers, substructure names, and other non-graphic information.

Points are the basic geometric entities in the data base. The most common way to specify the position of a point is simply to state its absolute coordinates. An alternative, called relative coordinates, entails stating the displacement required to get to a point from the previous point. Another alternate way to specify the position of a point is in relation to a previously set origin. This mode of operation, origin offset, is an alternate form of relative coordinates. In a table of points, different codes are used to distinguish between absolute, relative and origin offset. 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., move to (30000,30000), draw relative to (20000,20000), leaving the beam positioned at (50000,50000)). Such is not the case 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). Graphics systems are often designed to understand codes for several common sequences of basic instructions (i.e., "move,draw,move,draw,..."), so that large tables of points can be processed based upon a single pre-specified code. Such sequences can be very useful in eliminating overhead otherwise required for each new type command code.

If a structure to be displayed lies in a plane, the most efficient way to define this is to use two-dimensional data. In this case, it is typical to supply the cartesian "x" and "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, the largest expressable positive number is 32767. This is sufficient for many applications, but the need to express larger numbers sometimes arises. By employing an alternate means of expressing data, called homogenous coordinates, this need can be met at the expense of some loss of resolution in data definition. 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.

For a 16-bit computer, 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, such as x, is 50000, the value of homogenous coordinates becomes apparent because "h" can be made 1/2 to make x expressable; the point is then defined as the set of coordinates (1/2*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 that resolution is lost when "h" is 1/2, making it 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. Further, resolution is lost in all three coordinates if only one of them is out of bounds. Smaller values of "h" impose a correspondingly greater loss of resolution.

It is possible to conserve data base memory by supplying only the first two coordinates (hx,hy) for two-dimensional points (with a common value for hz) or three coordinates (hx,hy,hz) for three-dimensional points, and to pre-specify a fourth coordinate (referred to as "w") which applies to several such points. There are cases, however, where it is desirable to specify a fourth coordinate for every point. For those instances, the form (x,y,z,w) is used to specify every point.

2.3 PICTURE PREPARATION

The data base is seldom identical to the display file which is used to refresh the display. The data base represents a 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 usually required. In order to prepare a structure for display, it may need to be changed in size, position, or orientation; it may need to be put in perspective as seen from a given viewpoint; parts of it may need to be removed to keep the scene 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 digital 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 homogenous coordinates or, as special cases, two- or three-dimensional coordinates.

This matrix expression of transformations is used due to its simplicity, allowing system design to 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 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 transformation to be applied. Enormous savings can be introduced, however, by taking advantage of the fact that matrix multiplication is associative; it is equivalent to first forming a compound matrix by multiplying together matrices representing all the simple transformations in the sequence, in the same order in which the data would have encountered the original transformations, and then applying this compound matrix to all points to be transformed. This process is known as matrix concatenation.

2.3.3 Perspective

It is relatively easy 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 since 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 projection.

2.3.4 Windowing

In some graphics applications, the data base is to be displayed in its entirety on the screen. Often, however, only a portion of the data base is to be viewed. The specification of what portion of the data base is to be viewed is called a window and is usually defined in terms of the coordinates of the data base. Determining what parts of the data base are within the window and what parts are not is a difficult problem. In fact, this determination is so time-consuming in software that it jeopardizes the dynamic movement of the picture.

Sophisticated graphics systems address this 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 window superimposed on the data base. Lines or portions of lines outside the window are eliminated and only visible lines are passed to the screen for display.

In two dimensions, the window is 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 may be specified by supplying values for its left, right, bottom and top boundaries. Two-dimensional clipping is diagrammed in Figure 2.3-2.

In three dimensions, the window 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 will 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.

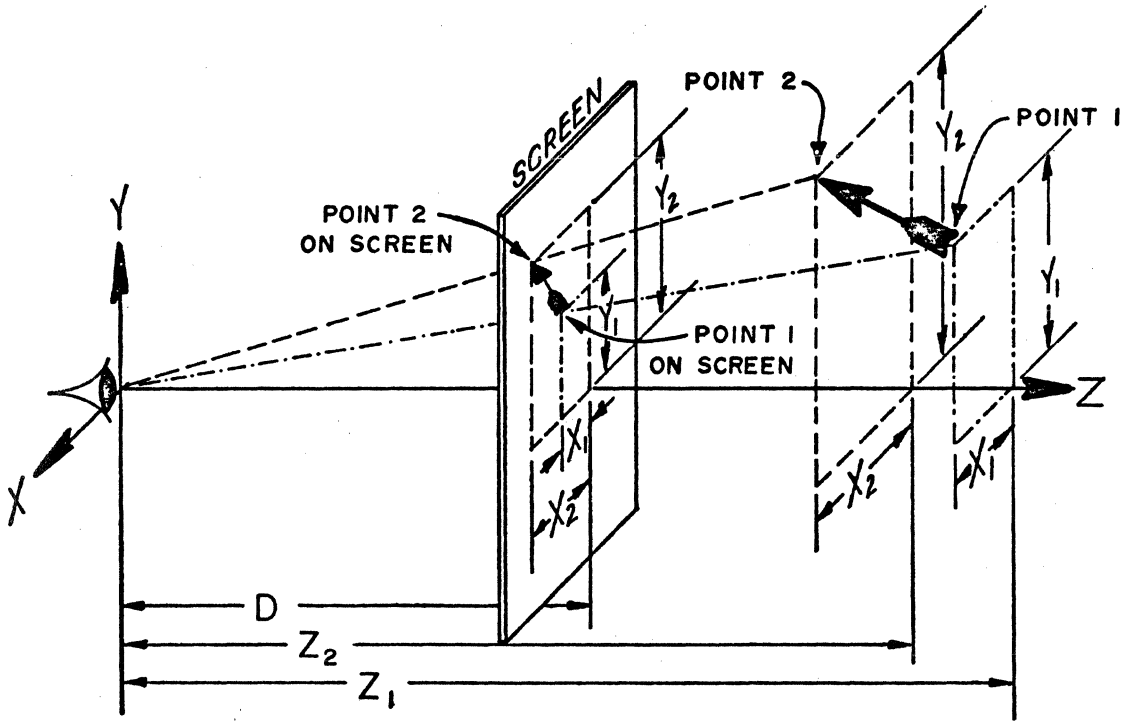


Figure 2.3-1

Three-Dimensional Perspective Projection
onto a Two-Dimensional Plane

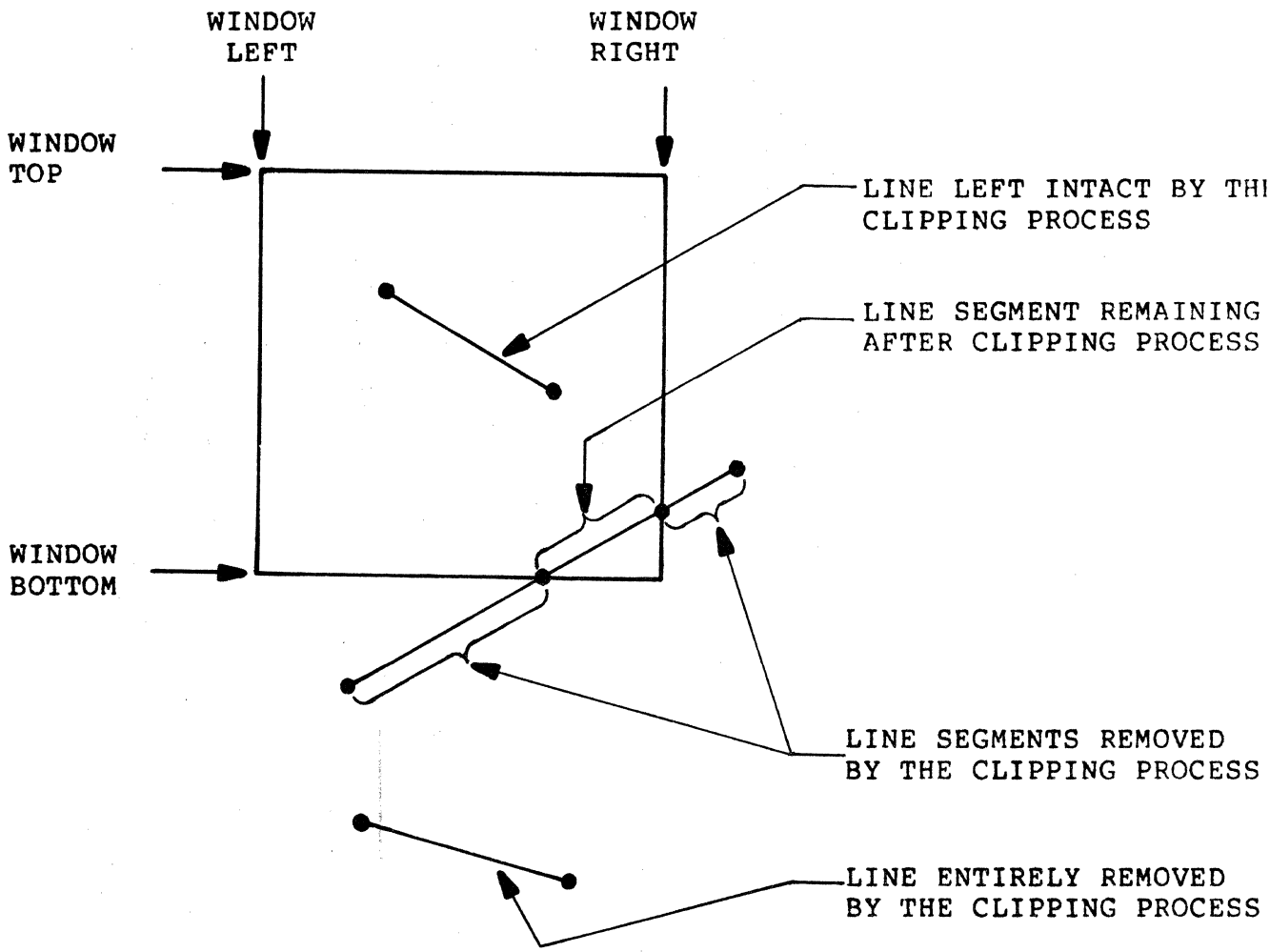


Figure 2.3-2

Two-Dimensional Clipping

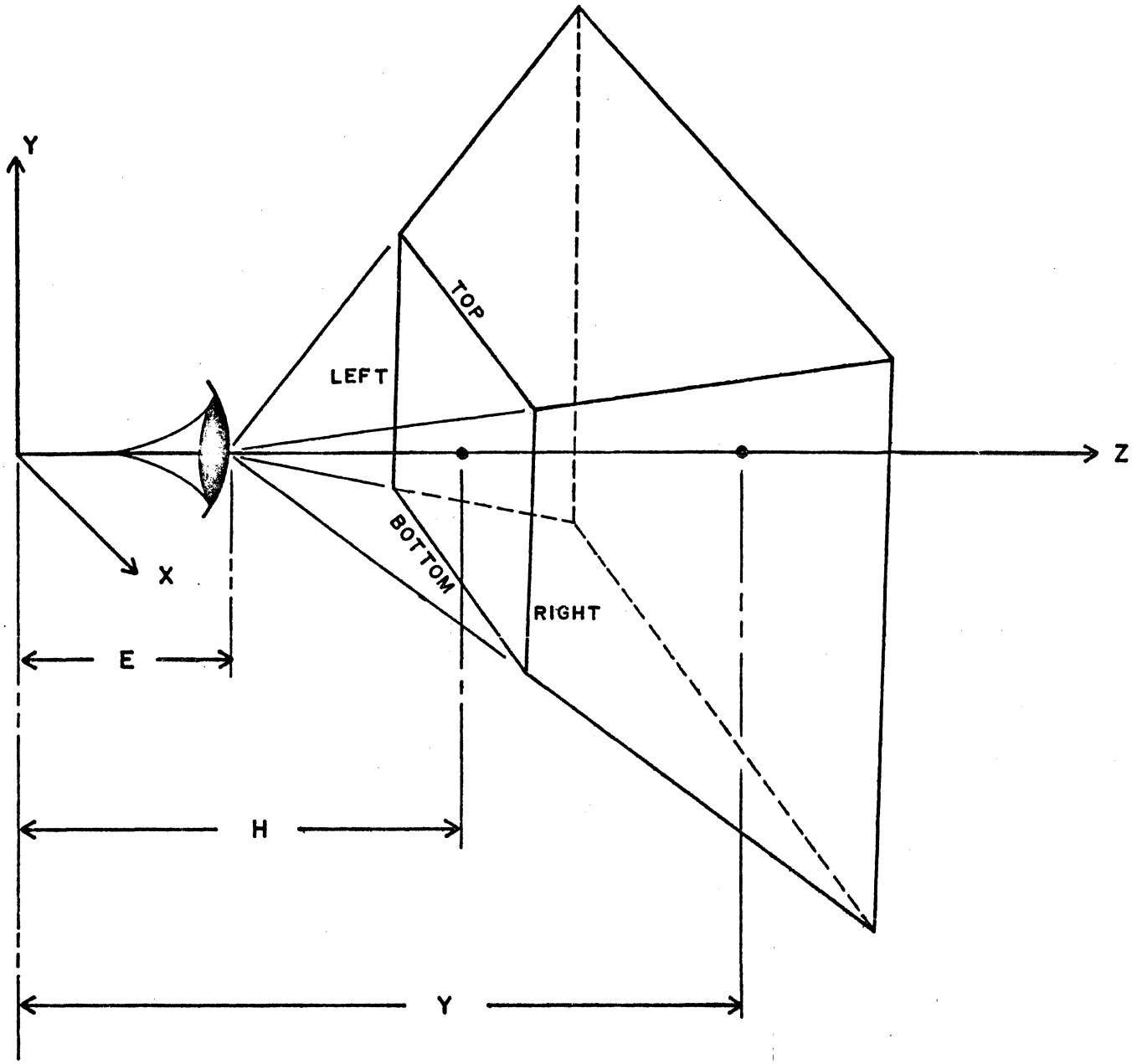


Figure 2.3-3

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

As in the two-dimensional case, lines are retained, completely eliminated, or partially eliminated depending upon 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 those lines and line segments actually on the screen. Scissoring is easier and less time-consuming to implement than clipping 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, causing flicker to occur more readily.

2.3.5 Conversion to Screen Coordinates

Coordinate data remaining after the clipping process, which may be of any size and at any position in the data base definition space, should be properly scaled (or mapped) so that it fills some pre-specified area on the screen. This area, known as a viewport, can be any portion of the viewing region on the display.

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. The data can be displayed in a viewport, the whole screen, or part of it, as shown in Figures 2.3-4a and b. 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 simultaneously showing different views of an object or views in different directions from the same point on the same output device.

It should be noted that if the windowing process were performed by scissoring rather than clipping, the use of viewports would be impossible since scissoring is done using the screen coordinate drawing region of the CRT.

2.3.6 Text Display

Almost all graphics applications call for the presentation of alphanumerics on the screen at various times. It is, of course, possible to define character shapes in the data base like other picture elements. In fact, this is necessary if characters are to be treated like other objects, i.e., rotated, clipped, etc. When they do not require such sophisticated treatment, it is possible to derive efficiencies from the foreknowledge of character properties 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. To use such a device to draw a string of characters, a display program must first stipulate character size, shape and orientation values. The program then positions the beam where the string is to begin and inserts 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.

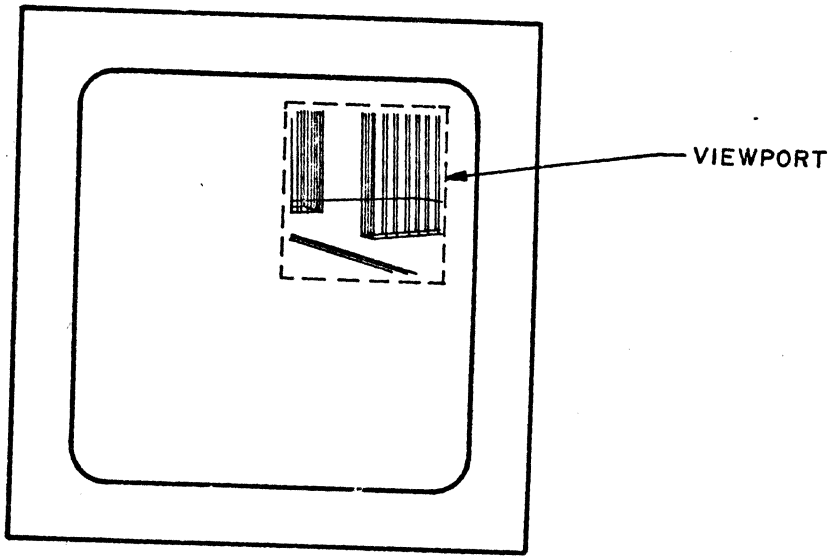


Figure 2.3-4a

Partial Screen Viewport

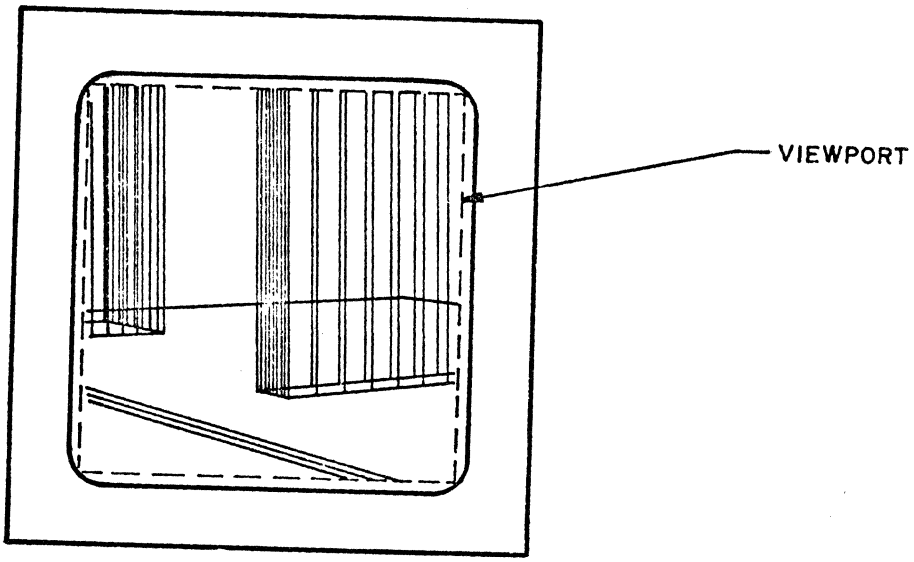


Figure 2.3-4b

Full Screen Viewport

2.4 PICTURE INTERACTION

Sophisticated graphics applications often require that the form or content of the picture be changable by the user. Several input devices for picture interaction are generally available for interactive graphics systems. These devices include light pens, functions switches, control dials, joy sticks, trackballs, and data tablets. Each has its own advantages and disadvantages.

The light pen, a light sensitive stylus connected to the computer, is a common input device. When the tip of the stylus is held against the screen and over a line segment, an interrupt is generated when the line is drawn. The computer can then determine what line in the display file was being pointed to. If the display file is a transformed display file where clipping may have been performed, the display file data may have little or no correlation to the original data base. Therefore, light pen systems must be very carefully designed to provide information which facilitates the correlation of the display file with the original data base.

Function switches, whose polarity can be read, are frequently attached to the computer in a graphics system. Each switch can be assigned a meaning unique to the program. Often there is a light associated with each switch which may be programmably turned off and on to provide operator feedback as to the polarity of the switch.

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

A versatile interactive input device is the data tablet. It consists of a flat rectangular plate which can be positioned on a table in front of, or near, the display screen, and a pen which may be moved about over the plate. The pen position on the plate may be read with fine resolution by the computer controlling the system. The computer can also detect if the pen is actually touching the plate and may also indicate if the pen is near the plate. A cursor is generally drawn on the screen to relate the pen motion to the image being displayed. This cursor is a small symbol which continually moves about in unison 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 operation of function switches or the analog input 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, especially if clipping has been performed.

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 at any distance desired from the graphic display. This reduces user fatigue and improves operating conditions.

CHAPTER THREE

3. OVERVIEW OF PICTURE SYSTEM 2

This chapter provides an overview of the hardware components which comprise PICTURE SYSTEM 2. The user of PICTURE SYSTEM 2 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.

The basic hardware components comprising PICTURE SYSTEM 2 consist of the Picture Controller Interface, Picture Data Bus, Picture Processor, Picture Memory, Picture Generator, Picture Display and Interactive Devices. These components are used in conjunction with the Picture Controller to comprise a stand-alone computer graphics system. All operations performed by the PICTURE SYSTEM and its components are overlapped to allow for simultaneous throughput in each of the processing units. A functional diagram of PICTURE SYSTEM 2 is shown in Figure 3-1.

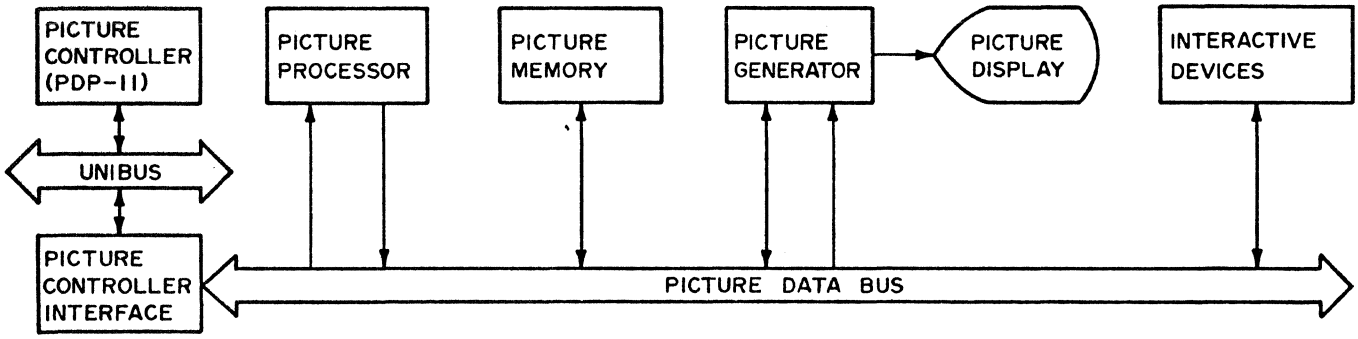


Figure 3-1

PICTURE SYSTEM 2 Components

The Picture Controller is a general-purpose computer, typically a minicomputer such as Digital Equipment Corporation's PDP-11, which directs the display of pictures on PICTURE SYSTEM 2. The Picture Controller communicates with the PICTURE SYSTEM through a Direct I/O and Direct Memory Access component called the Picture Controller Interface.

The Picture Controller Interface allows the PICTURE SYSTEM work station to be positioned up to 500 feet from the Picture Controller.

The Picture Controller is used to:

* Contain the data base describing the object(s) to be viewed.
* Control 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 control program executes.

All PICTURE SYSTEM 2 components and interactive devices are connected to and interact with the synchronous Picture Data Bus. These devices and components, as listed below, are described in detail in the following sections.

* The Picture Controller Interface
* The Picture Data Bus
* The Picture Processor
* The Picture Memory
* The Picture Generator
* The Interactive Devices

3.1 PICTURE CONTROLLER INTERFACE

The Picture Controller Interface has a double cable which runs from the Picture Controller to the PICTURE SYSTEM work station. This cable allows the PICTURE SYSTEM hardware to be located up to 500 feet from the Picture Controller.

Communication between the Picture Controller and the PICTURE SYSTEM is achieved through the use of two I/O paths:

1. Direct I/O (DIO) path
2. Direct Memory Access (DMA) path

The Direct I/O path is typically used to pass single-word commands to or from the PICTURE SYSTEM. This provides convenient access to the PICTURE SYSTEM registers by the computer.

Using the DMA path, blocks of coordinate data may be transferred to or from the PICTURE SYSTEM without direct supervision by the Picture Controller.

An example of the use of the Direct I/O path would be the transfer of coordinate data to the PICTURE SYSTEM for processing while the DMA path is concurrently transferring the processed data back to the Picture Controller. Another example would be the use of the DMA to transfer data to the Picture Processor for processing at the same time the Direct I/O path is setting up for a current frame to be refreshed. In both cases, these data paths may be used concurrently for information flow.

The Picture Controller Interface also provides interrupt capabilities for management of PICTURE SYSTEM 2 components and interactive devices. The Picture Controller Interface distributes data to and retrieves data from the various components and devices of the PICTURE SYSTEM via the Picture Data Bus.

3.2 PICTURE DATA BUS

All PICTURE SYSTEM 2 components and devices connect to and interact with each other on a single, high-speed synchronous data bus. Memory, device registers, and command, control and status registers all exist and are addressable memory locations on the Picture Data Bus.

Use of the Data Bus allows coordinate data to be transferred from the Picture Controller to the Picture Processor while data are concurrently being transferred from the Picture Processor to the Picture Memory. During this process, data are also being transferred from the PICTURE SYSTEM Memory to the Picture Generator for display. In addition, data may be entered from the Data Tablet or other interactive devices and read by the control program in the Picture Controller.

Data flow is supervised by a bus arbitration system which is integral to the Picture Data Bus.

3.3 PICTURE PROCESSOR

The Picture Processor is a high-speed, microprogrammed, digital arithmetic processor capable of accepting two-, three-, or four-dimensional (homogeneous) data; transforming the data; clipping the transformed data at the boundaries of a six-sided window; performing a perspective division and viewport mapping on the clipped data and outputting the transformed, clipped and viewport mapped data for subsequent display. There are three basic units in the Picture Processor:

* Input Controller
* Matrix Arithmetic Processor (MAP)
* Output Formatter

The Input Controller receives data, typically from the Picture Controller, and channels the data to the Matrix Arithmetic Processor. The MAP Output Formatter receives the processed data from the Matrix Arithmetic Processor and outputs the data formatted for subsequent display. The Output Formatter may alternately output the data in full 16-bit precision for use by the Picture Controller.

The Matrix Arithmetic Processor (MAP) is the major unit of the Picture Processor and consists of a Transformation Matrix, Transformation Matrix Stack, Parameter Register File and Arithmetic Unit.

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 fourteen 4x4 matrices may be "stacked" or saved for future recall.

The Parameter Register File is an array of 16-bit registers into which parameters specifying viewport boundaries, scale factors, etc. are stored and may be retrieved.

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

The MAP utilizes its units to perform digital operations on the data received from the Picture Controller. These operations are:

* To transform two-, three- and four-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 or Picture Memory.
* To store the contents of the Transformation Matrix into the Picture Controller or Picture Memory.
* To concatenate the contents of the Transformation Matrix with a 4x4 matrix in the Picture Controller or Picture Memory to obtain a compound transformation.
* To load and store the registers of the Picture Processor.
* To check transformed coordinate data for visibility by comparing 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 line segments are processed further.
* To convert three-dimensional data to two dimensions by computing perspective or orthographic projections.
* To perform a linear mapping of points from the object's coordinate system into that of the Picture Display or any other coordinate system specified.

3.4 PICTURE MEMORY

The Picture Memory is a dual-port MOS memory (distinct from the Picture Controller's) organized as addressable 16-bit words. This memory is available in increments of 16K words, expandable to 64K words of memory, dependent upon user requirements.

Picture Memory may be used in a variety of ways to satisfy the user's application. Typically, a portion of the Picture Memory serves as a refresh buffer into which data, still in digital form, is deposited. This data represents information to be shown on the Picture Display. For each frame displayed, the Refresh Controller reads the data from the Picture Memory and channels this data to the Line Generator where the data are then converted to analog signals to drive the Picture Display.

The ways in which the Picture Memory may be used as a refresh buffer include the single-buffer mode, double-buffer mode and segmented-buffer mode. (See Refresh Controller, Section 3.5.1, for further information.)

Pages 3-9 and 3-10 missing from original

45-degrees counter-clockwise, etc.) while maintaining the correct response to the character positioning commands.

The Character Generator has a memory, half of which is pre-programmed to interpret the 128 ASCII characters as described above, and half of which is available for the user to program various character fonts and special symbols.

3.5.3 Line Generator

The Line Generator receives data detailing coordinate point positions, status information which describes the modes of operation, and character codes which are passed to the Character Generator for interpretation. This information is used to produce the final image seen by the viewer.

Coordinate point positions are specified by Move or Draw commands in the Picture Display coordinate system. Each coordinate point, defined as X, Y and intensity (Z), causes the Line Generator to position the electron beam of the Picture Display to the location specified. If the command is a Move, the beam will then be positioned without intensification. If the Line Generator is currently in Dot mode, the beam will be positioned and then intensified. If the command is a Draw, a line will be drawn from the current beam location, positioned from a previous Move or Draw command, to the coordinate specified. The Line Generator has the capability of varying the intensity of a line between its endpoints. The line drawn will be in the line texture (dashed, etc.) currently selected for the Line Generator.

A status command to the Line Generator specifies the modes in which lines and characters are to be displayed. These modes are:

* Blink
* Texture
* Color
* Display Select

Setting the Blink mode indicates that all subsequent lines and characters output to the display are to blink until this mode is reset.

The Texture mode is used to indicate the manner in which all subsequent lines and characters are to be drawn by the Line Generator. Line textures available are: solid line mode; long dashed line mode; long-short dashed line mode; long-short-short dashed line mode and short dashed line mode.

For all of the above dashed line textures, the Line Generator ensures that dashes begin and end at the endpoints of the line.

Conversely, the Line Generator may be set to a mode whereby textured line segments may be displayed without concern for individual line endpoints. This mode effectively allows the appearance of a smoother curve by de-emphasizing the individual line segment endpoints.

Color selection is used in conjunction with a color monitor to initiate changes in color for all subsequent lines and characters which are output to the display after this mode is initiated.

In connection with the color mode selection, an intensity selection is also used to control the drawing speed of the Line Generator. This change in drawing speed is utilized to present an appearance of equal intensity for the various colors available (i.e., red, red-orange, orange, yellow and green).

The Display Select mode is used in multiple Picture Display configurations to choose which Picture Display(s) all subsequent lines and characters are to be output. The Display Select capabilities allow one or more images to be viewed on any combination of the six possible Picture Displays.

Character codes are not interpreted by the Line Generator, but are passed to the Character Generator. The Character Generator interprets the character codes, generating individual strokes which are then channeled back to the Line Generator for display.

The Line Generator is capable of displaying lines and characters at one distinct intensity level. The depth-cue feature increases the number of intensity levels which may be displayed from 1 to 64. This feature also allows the intensity of a single line to vary, from bright to dim according

to its Z value, imparting the illusion of depth to three-dimensional images.

3.5.4 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, characters and dots on the Picture Display.

3.6 PICTURE SYSTEM INTERACTIVE DEVICES

All PICTURE SYSTEM 2 interactive devices are interfaced directly to the Picture Data Bus. Data may be input from the interactive devices by the control program using the Direct I/O path of the Picture Controller Interface. These interactive devices may be used under interrupt control or may be polled directly by the user program in the Picture Controller.

The following interactive devices are supported by PICTURE SYSTEM 2:

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

The use of these graphical input devices provide all the capabilities typically required for graphical interaction with PICTURE SYSTEM 2. Appropriate use of these interactive devices, along with the dynamic qualities of PICTURE SYSTEM 2, provide the user with the tools required for a three-dimensional, interactive graphics system.

3.6.1 Tablet

The Tablet serves as the standard, general-purpose graphic input device in PICTURE SYSTEM 2. The Tablet can be used for positioning or pointing to the picture elements by use of a stylus, or 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. In conjunction with the Picture Processor, the tablet and pen can perform the interactive functions usually reserved for such graphic input devices as light pens, joysticks and function switches.

3.6.2 Control Dials

Control Dials permit the user to control size, position and orientation of objects or other functions required by the application.

3.6.3 Function Switches & Lights

Function Switches & Lights provide the user the capability to utilize switches for functions assigned under program control. The lights may be used to indicate function switch setting or to display programmed information to the user.

3.6.4 Alphanumeric Keyboard

The Alphanumeric Keyboard is a standard 61-key, 128-character keyboard used for text or data input to the Picture Controller for graphical interaction or other functions required by the application.

CHAPTER FOUR

4. PROGRAMMING PICTURE SYSTEM 2

This chapter describes the Graphics Software Package developed for the PICTURE SYSTEM. The software package consists of a set of FORTRAN callable subroutines which allow the FORTRAN programmer to perform general purpose graphics functions. This chapter explains basic program structures, the various coordinate systems of PICTURE SYSTEM 2, simple and compound transformations, and the manner in which each of the subroutines of the Graphics Software Package are used. The information is organized to lead the programmer through the task of designing a program structure, defining the data to be viewed, specifying the scene definition, applying transformations to the data and interacting with the picture being displayed. The intent of this chapter is not to provide instruction in programming technique, but rather to illustrate the use of the PICTURE SYSTEM 2 Graphics Software Package.

4.1 GENERAL PROGRAM STRUCTURE

Programs written for PICTURE SYSTEM 2 generally contain the following segments:

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

The Data Definition section 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 section 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 application programs is structured as shown in Figure 4.1-1. This program structure lends itself to the interactive environment of PICTURE SYSTEM 2 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 4.1-1 illustrates, the display loop consists of:

1. Data Input (e.g., update of tablet values, 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 segmented or 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.

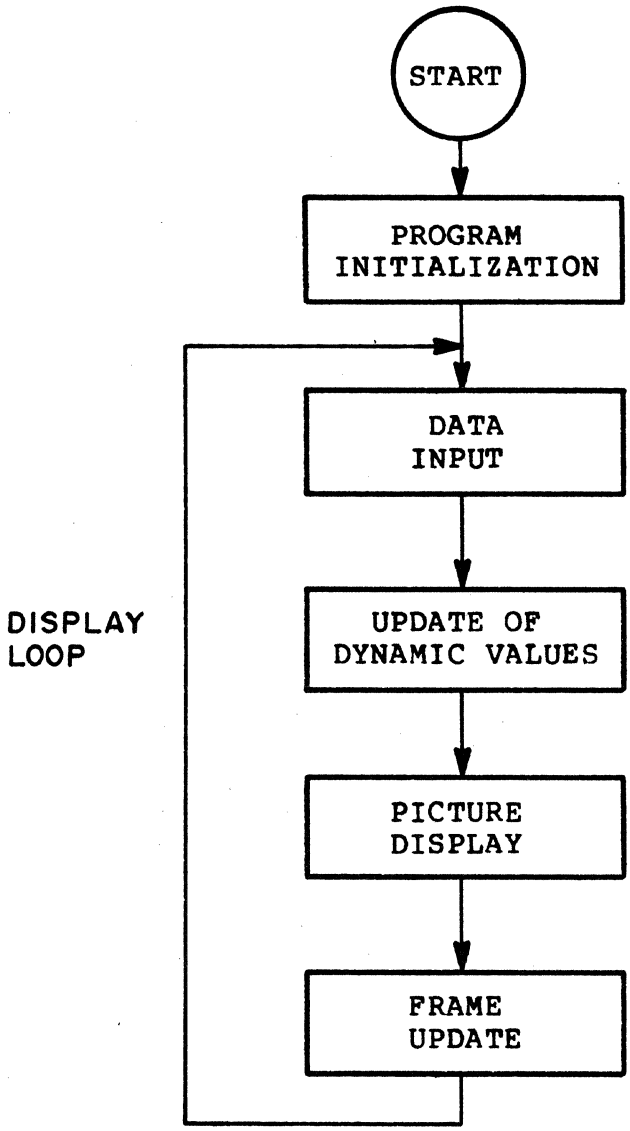


Figure 4.1-1

General Interactive Program Structure

Item c, above is important in the design of the system as it allows maximum utilization of the host Picture Controller and increases the frame update rate. The program structure of Figure 4.1-1 may be modified as shown in Figure 4.1-2 to increase the frame update rate. A comparison of Figures 4.1-1 and 4.1-2 shows that the difference in program structure is 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 conjuction with the tablet. In this case, menu selection testing or hit testing need not be done unless the pen is "down", i.e., touching the surface of the tablet. The user need only test the IPEN parameter to determine whether data input is to be done from the tablet (see Section 4.13).

The program structure of Figure 4.1-1 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 4.1-3, is a modification of Figure 4.1-2 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 only if a portion of the picture has 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. It should be noted that the program structures of Figures 4.1-1 and 4.1-2 create a new frame with each execution of the display loop, whether a new frame creation is necessary or not.

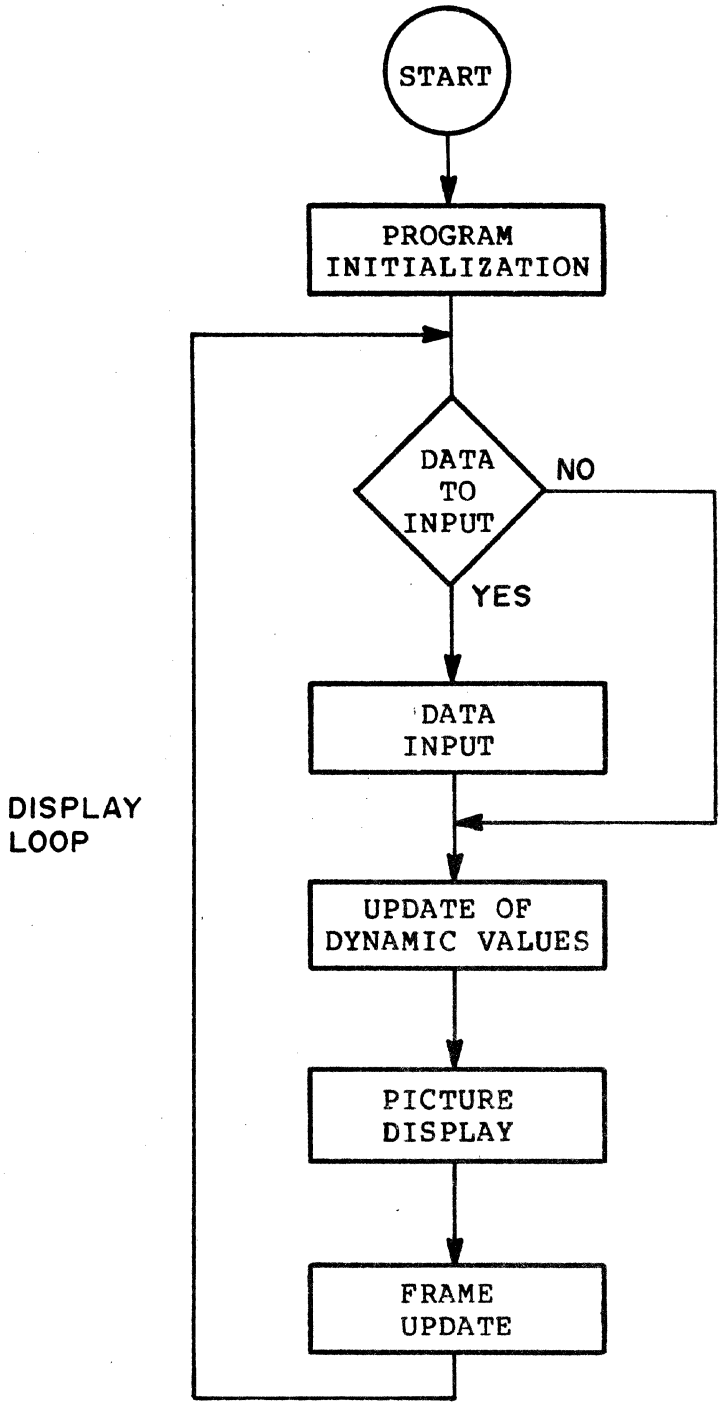


Figure 4.1-2

Program Structure to Increase Frame Update Rate

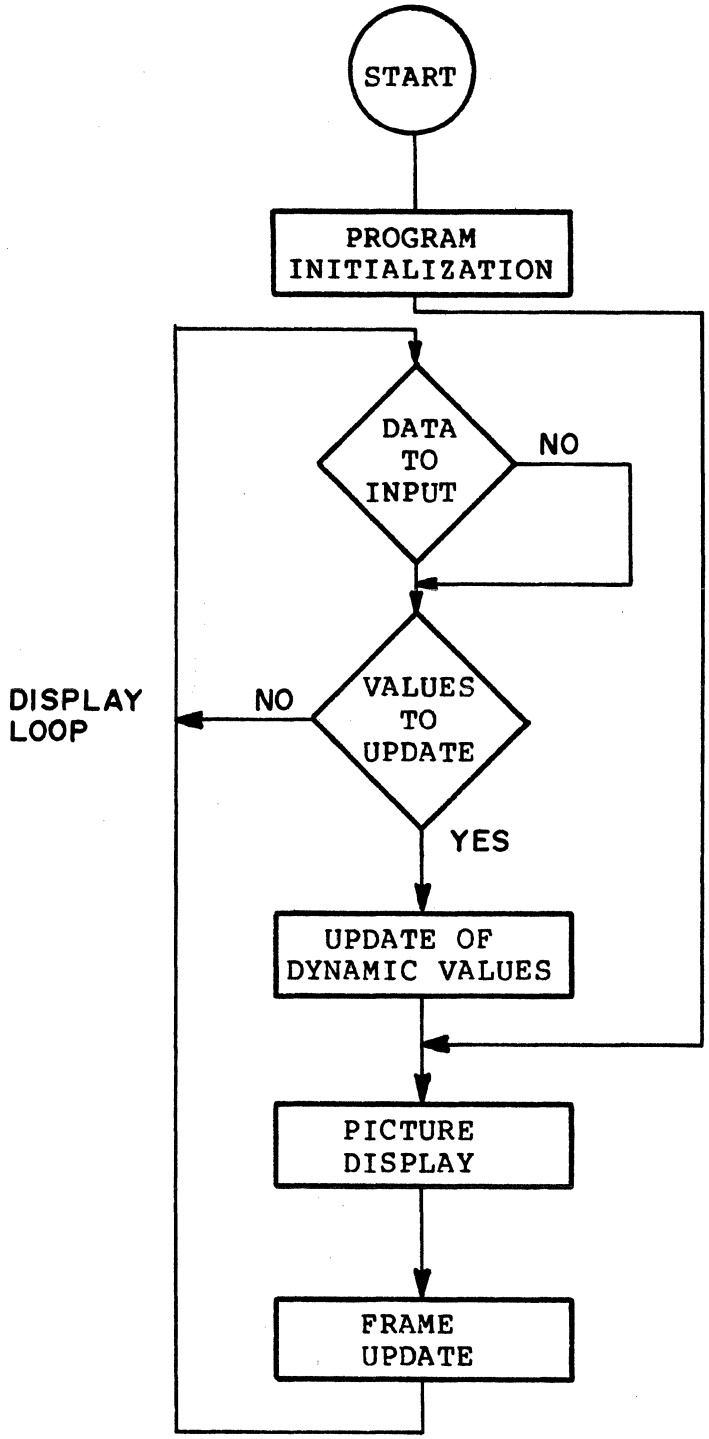


Figure 4.1-3

Program Structure to Increase System Response to User Action

If the PICTURE SYSTEM is operating in a stand-alone environment in which graphics display and interaction are the only functions of the Picture Controller, frame update rate is the only time constraint, and the user program may remain in the display loop in Figures 4.1-1, 4.1-2 or 4.1-3 without concern for processing time. However, if the graphics system shares a Picture Controller in a foreground/background or multi-programming mode 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 wich to utilize the program structure shown in Figure 4.1-4. This structure would allow a graphics program to execute in foreground mode, with all the priorities and priviliges afforded a foreground program, and yet allow background programs to execute whenever possible.

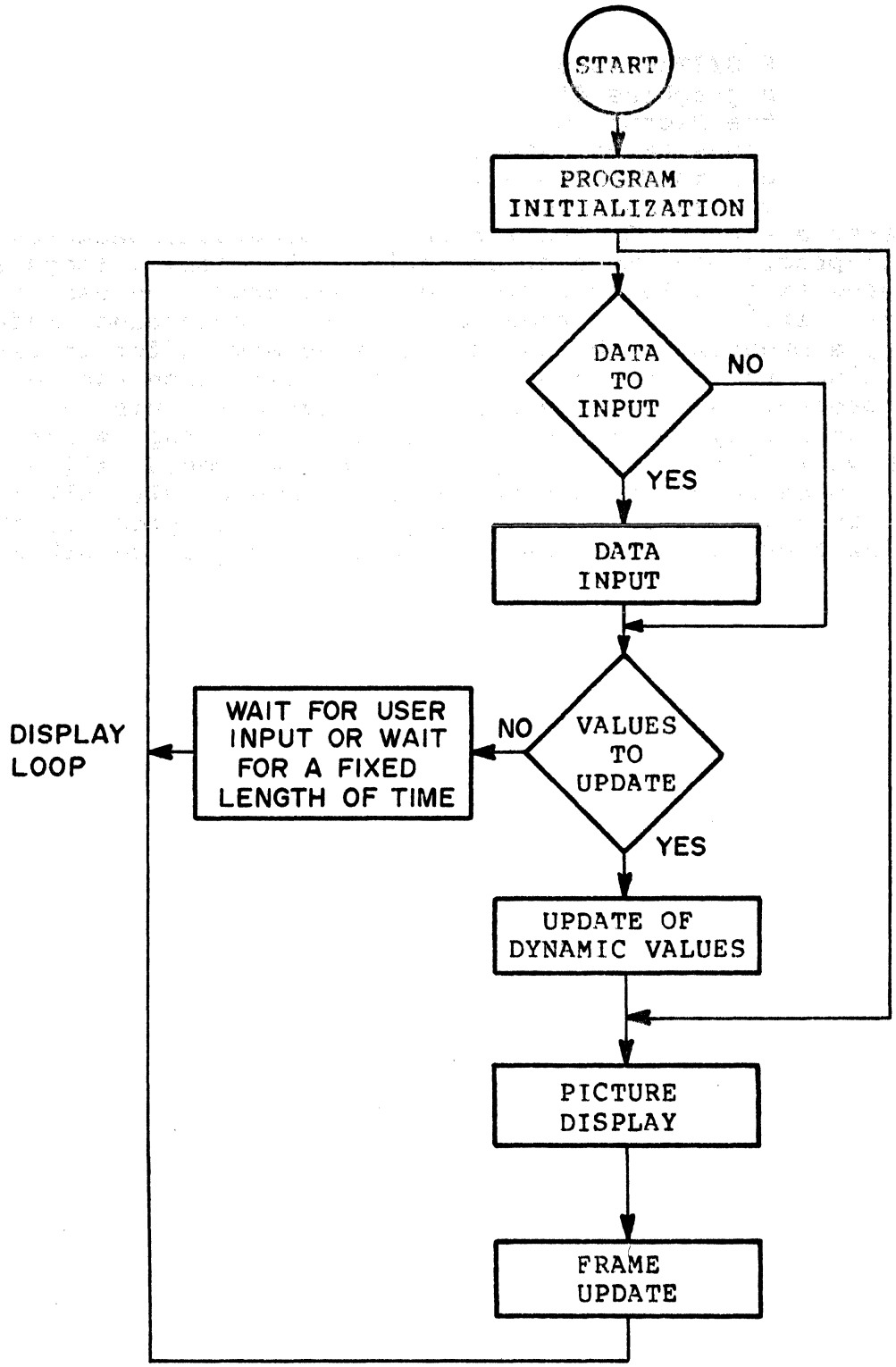


Figure 4.1-4

Program Structure for Foreground Execution

4.2 SCENE DEFINITION

All data displayed on PICTURE SYSTEM 2 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 points are defined within these coordinate systems and the transformations and order in which they should be applied to the data.

4.2.1 Coordinate Systems

The user of PICTURE SYSTEM 2 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.

4.2.1.1 Data Space Coordinates

The data space coordinate system is the region of definition space in which all data to be viewed are 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 4.2-1. 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-dimensions. Within this data space reside 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.

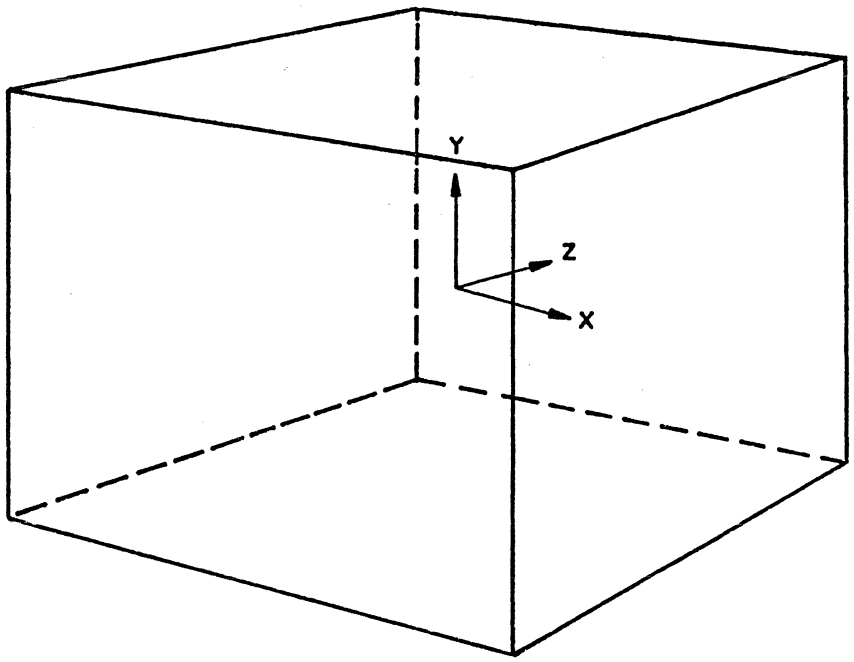


Figure 4.2-1

The Data Space Coordinate System

4.2.1.2 Homogeneous Coordinates

All data defined for use on PICTURE SYSTEM 2 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. Such an adjustment will have no effect on the graphic interpretation of the point. 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*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:

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

In the above example, this would result in:

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

or the homogeneous coordinates:

(32767,19660,-5243,10737)

Usually, 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 the resolution if 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 converted to the scale of the other by merely supplying the appropriate homogeneous coordinate (IW) when drawing the data using the DRAW2D, DRAW3D or DRAW4D subroutine. To determine the appropriate homogeneous coordinate for this example, the following equation would be used:

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

or

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

or

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

or

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

Therefore, the homogeneous coordinate, IW, would be:

```
IW = .78740*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 4.2-1.

```
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 4.2-1

There are ten subroutines in the PICTURE SYSTEM 2 Graphics Software Package in which the user may utilize the homogeneous coordinate system. They are:

```
WINDOW
TRAN
SCALE
MASTER
INST
DRAW2D
DRAW3D
HITWIN
GETTRN
GETSCL
```

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

It should be noted that, while homogeneous coordinates supply means of expressing very large values, fractional values (e.g., .5, .75, etc.) must be multiplied by an appropriate scale factor to form integer values for transformation by the PICTURE SYSTEM.

4.2.1.3 Screen Coordinates

All data within the data space (homogeneous or not) defined for display are ultimately mapped into the screen coordinate system by the Picture Processor for display by the Picture Generator. This mapping from the data 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 is concerned with the screen coordinate system only when specifying viewport boundaries.

The screen coordinate system for the Picture Display of the PICTURE SYSTEM 2 is shown in Figure 4.2-2. 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 in the x and y axes. 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 4.2-3. It is within this coordinate system that all viewports are specified. Since viewports may encompass a portion of the screen and pictorial data elements are 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.

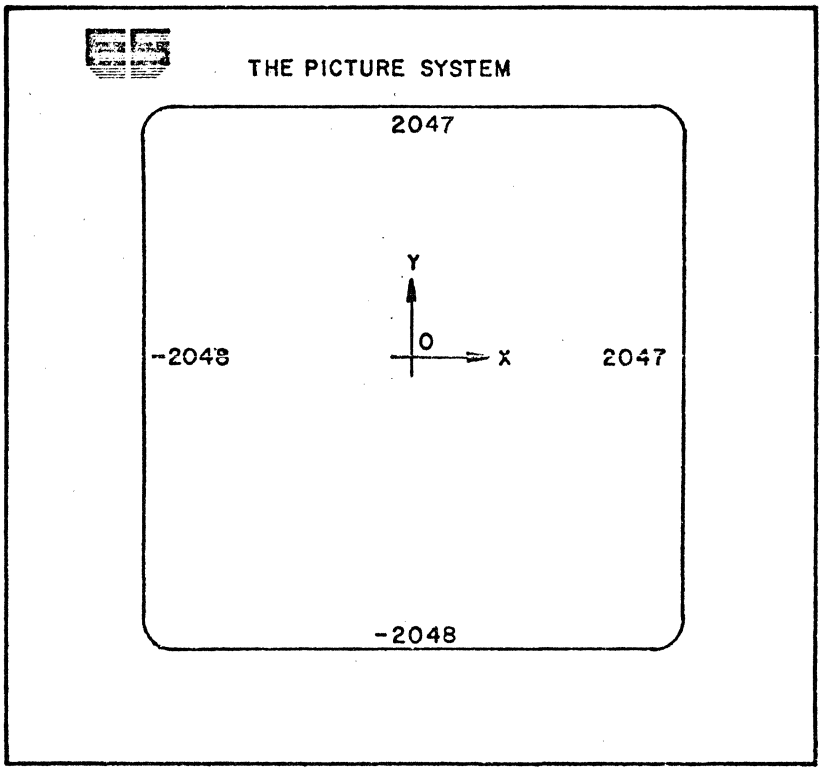


Figure 4.2-2

Screen Coordinate System of the
Picture Display

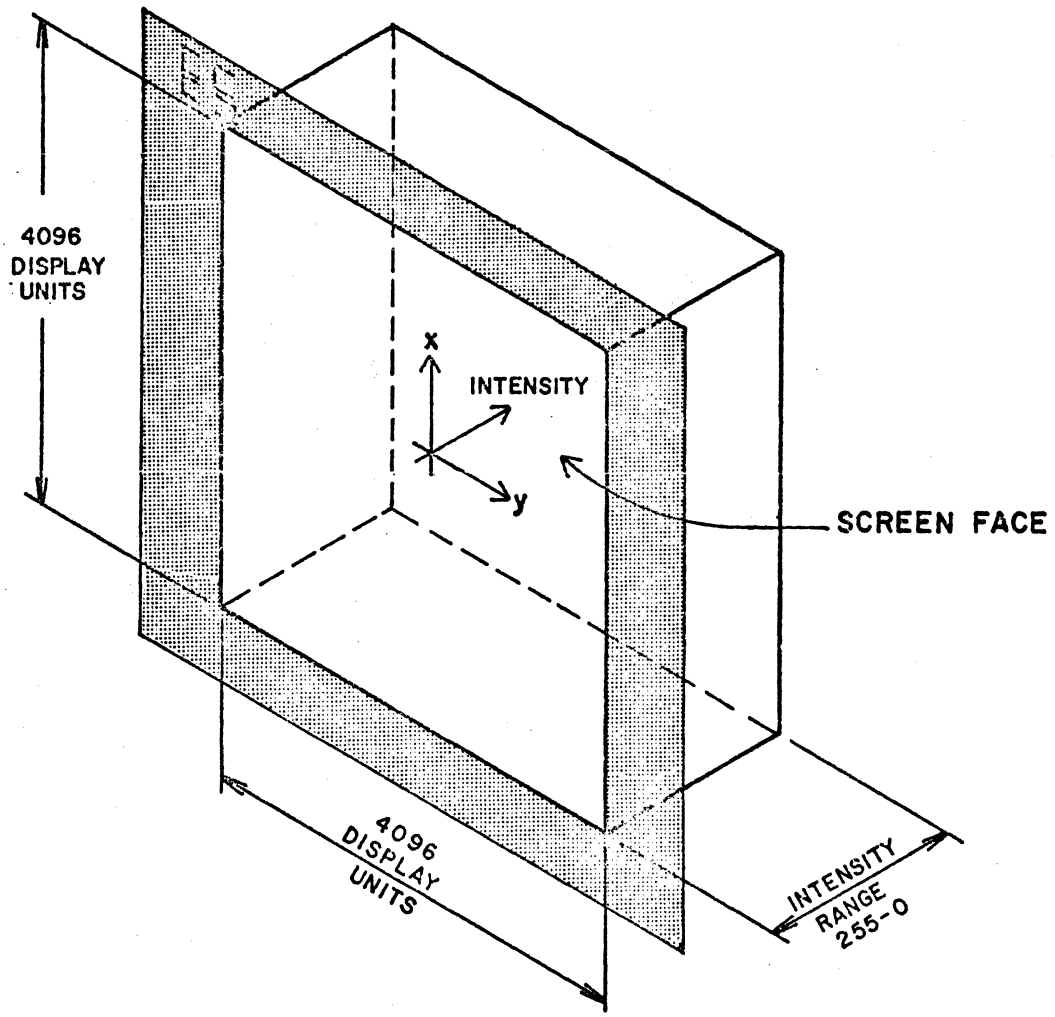


Figure 4.2-3

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

4.2.2 Data Definition

Graphic data to be displayed on PICTURE SYSTEM 2 are defined in the Picture Controller as a data set--a data set being an array of two-, three- or four-dimensional coordinate points to be drawn in a particular drawing mode.

All data displayed on the PICTURE SYSTEM 2 are treated by the hardware as homogeneous coordinate data; that is, each data point 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 are actually three-dimensional data that reside in a constant z plane), and three-dimensional data consists of x,y,z triples with a constant w coordinate. The notation used to represent a data set is illustrated in Figures 4.2-4a and b. All data of a particular data set to be displayed should be stored in the memory of the Picture Controller in a contiguous integer array to facilitate the accessing of data by the Direct Memory Access (DMA) interface of the Picture Processor. To ensure that data are stored as contiguous data elements in memory, the user should understand the array storage convention of ANSI 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:

```
N(1,1), N(2,1), . . . , N(J,1)
N(1,2), N(2,2), . . . , N(J,2)
   .
   .
   .
N(1,K), . . .         , N(J,K)
```

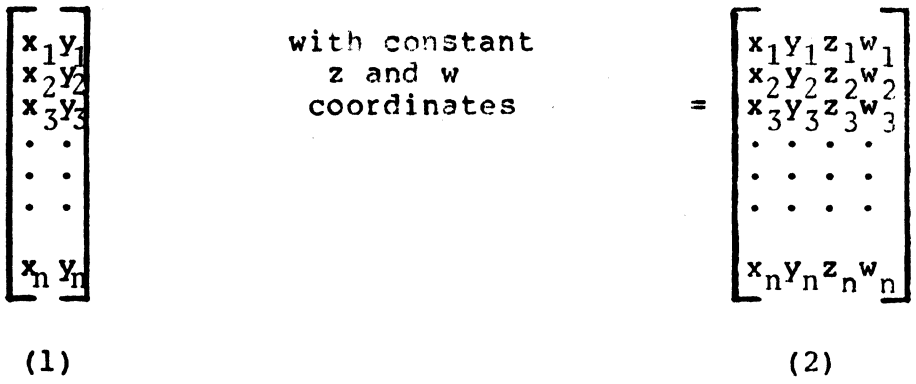


Figure 4.2-4a

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.

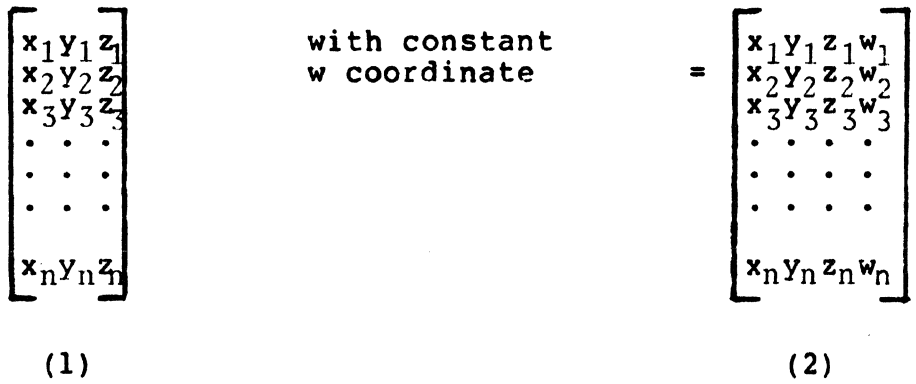


Figure 4.2-4b

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-, three- and four-dimensional data are accessed properly:

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

```
DIMENSION IDATA(2,n)*
```

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

```
DIMENSION IDATA(3,n)*
```

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

```
DIMENSION IDATA(4,n)*
```

In this manner, the data will appear as:

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

and for three-dimensional data:

```
IDATA(3,i) = zi
```

and for four-dimensional data:

```
IDATA(4,i) = wi
```

A specified data set as described above may then be displayed by calling the appropriate display subroutine (DRAW2D, DRAW3D or DRAW4D) and providing the drawing specifications. Figures 4.2-5a, b and c show the calling sequence used to display two-, three- and four-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 displayed picture 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 increase the distance of the data set from the hither

---

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

clipping plane (normally the hither clipping plane = 0 for two-dimensional displays).

Data displayed on the Picture Display are transformed, clipped and mapped to a portion of the display screen (viewport mapped) by the Picture Processor and stored in the refresh buffer. As a result, data elements within the refresh buffer are referred to as a transformed display file and may bear little resemblance to the original data.

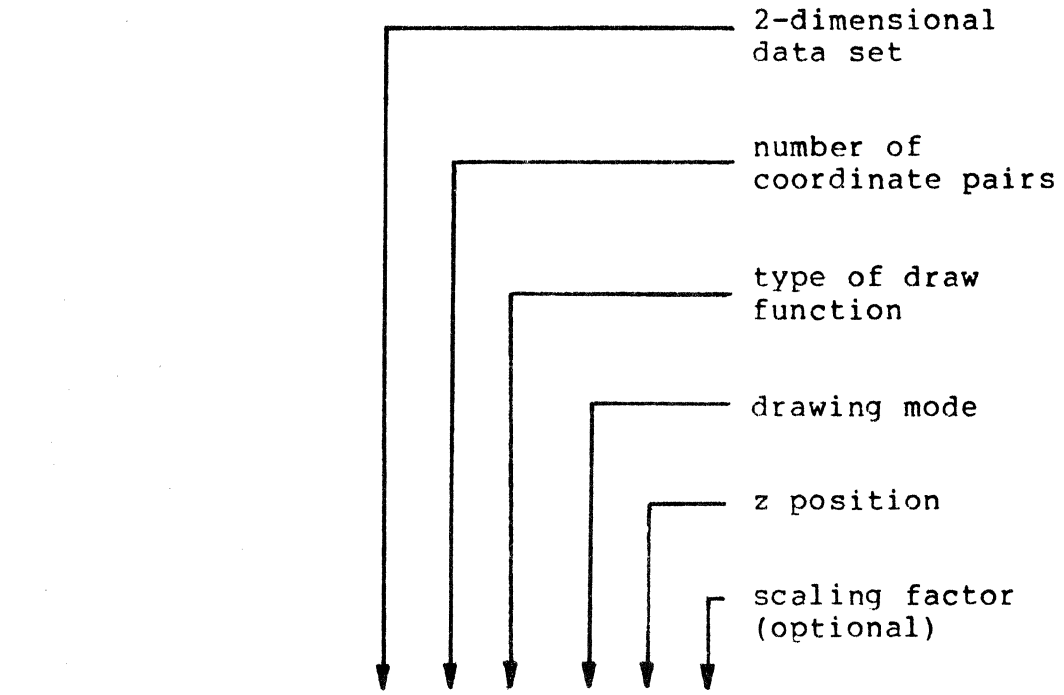


Figure 4.2-5a

Calling Sequence for Two-dimensional Display of Data

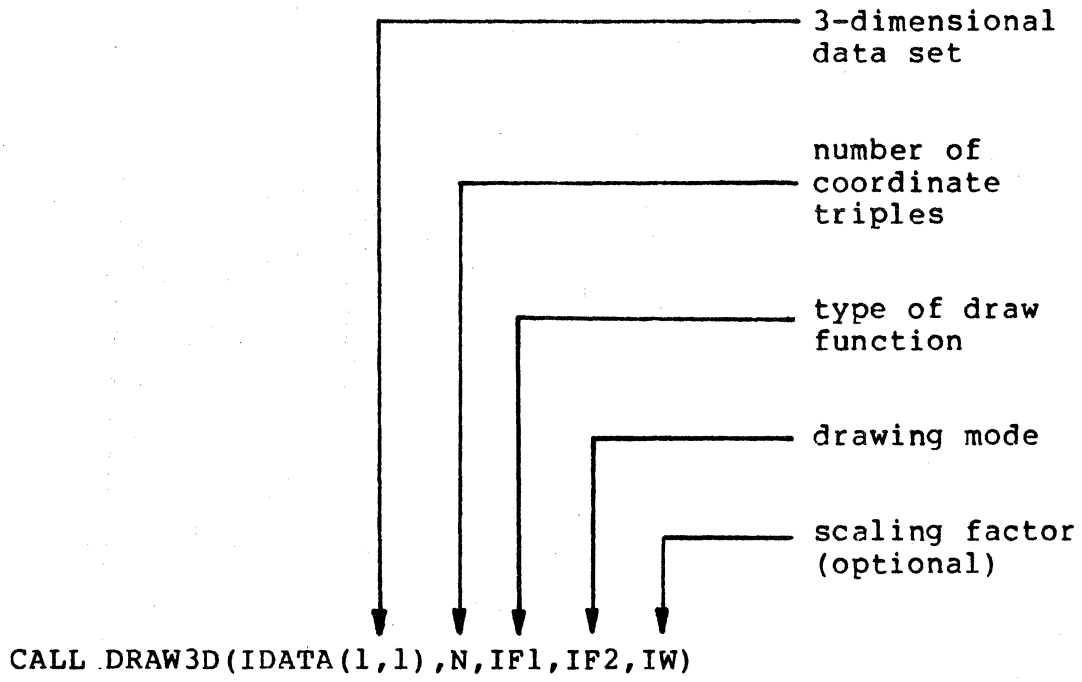


Figure 4.2-5b

Calling Sequence for Three-dimensional Display of Data

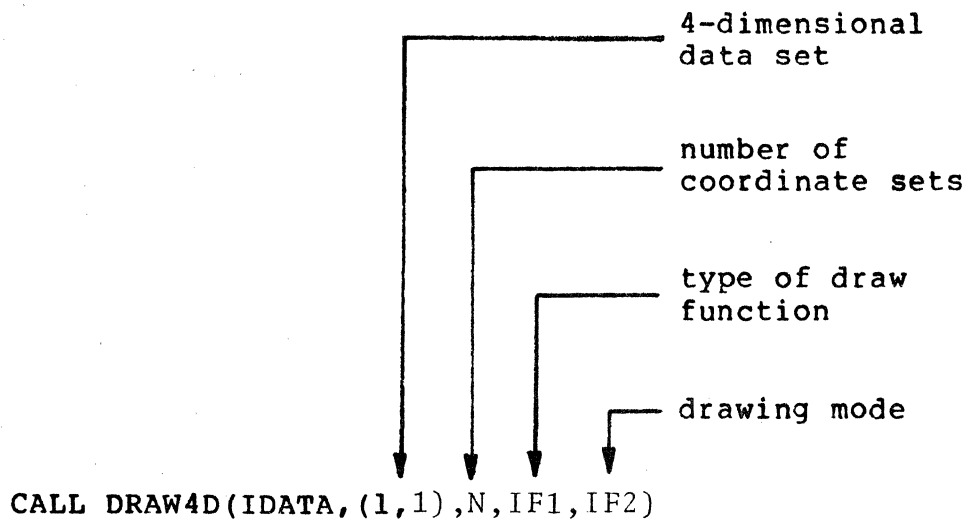


Figure 4.2-5c

Calling Sequence for Four-dimensional Display of Data

4.2.3 Transformations

All data shown on the Picture Display are linearly transformed by multiplying each coordinate point to be drawn by a 4x4 matrix*. 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.

4.2.3.1 The Identity Transformation

PICTURE SYSTEM 2 initialization subroutine, PSINIT, initializes the Picture Processor's Transformation Matrix to a 4x4 identity matrix of the form:

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

PICTURE SYSTEM 2 subroutines which alter the transformation matrix do so by matrix concatenation. Initializing to the identity 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 a non-zero scalar quantity without changing its graphic effect. Thus, PICTURE SYSTEM 2 automatically scales (or normalizes) 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. In fact, subroutine PSINIT uses the value 16384 in their place.

---

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

4.2.3.2 Simple Linear Transformation

All transformations performed by the graphics subroutines (i.e., WINDOWing, ROTation, TRANslation and SCAL(E)ing) are simple linear transformations; each is expressable as a 4x4 matrix. When called, the subroutines create a 4x4 matrix to perform the required linear transformation (e.g., ROTate 90 degrees, etc.) and concatenate 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-concatenated matrix.

Figure 4.2-6 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']. For data to be displayed without transformation in any manner, the Transformation Matrix must contain the identity matrix as shown in Figure 4.2-7.

4.2.3.3 Compound Transformations

A compound transformation may be thought of as a series of two or more 4x4 matrices multiplied together as illustrated in Figure 4.2-8.

Typically, all transformations that are to be applied to a given set of PICTURE SYSTEM 2 data are concatenated into one matrix, as in Figure 4.2-8, 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 4.2-6*

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

---

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

All 4x4 matrices will be represented by the notation:

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

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 by PICTURE SYSTEM 2 graphics software, see Reference 1.

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.

$$[x\ y\ z\ w]\begin{bmatrix} & I^* & \end{bmatrix} = [x'y'z'w'] = [x\ y\ z\ w]$$

(1) (2) (3)

Figure 4.2-7*

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 4.2-8

Three Simple Transformations and an Equivalent Compound Transformation.

---

*In this discussion, the identity 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 4.2-9. 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.

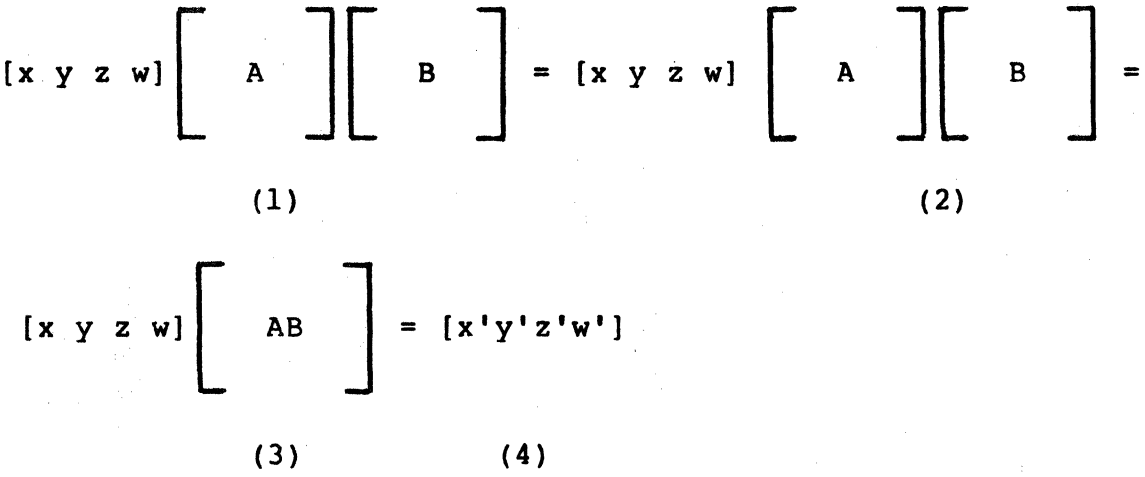


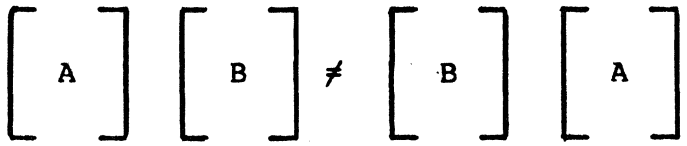
Figure 4.2-9

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 4.2-9 is indicative of the technique used to specify the transformations used in PICTURE SYSTEM 2 application programs. The transformations to be performed upon a given set of data are determined and diagrammed, IN THE ORDER THAT THE DATA ENCOUNTERS THE TRANSFORMATIONS TO BE PERFORMED*. A suggested order in which transformations may be performed is:

---

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



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 4.2-10 illustrates this order of transformation in the matrix notation previously defined.

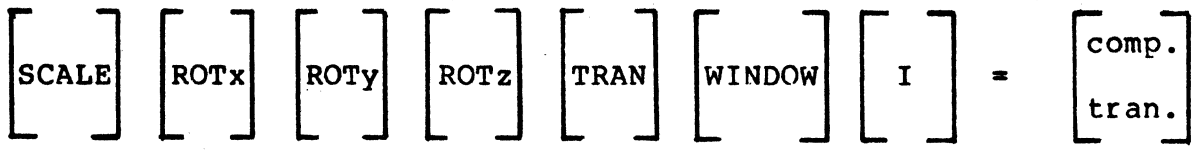


Figure 4.2-10

A Suggested Order in which Transformations may be Performed.

It should be noted that inclusion of all the transformations is not necessary and, 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. A suggested order of transformations for two-dimensional data is shown in Figure 4.2-11.

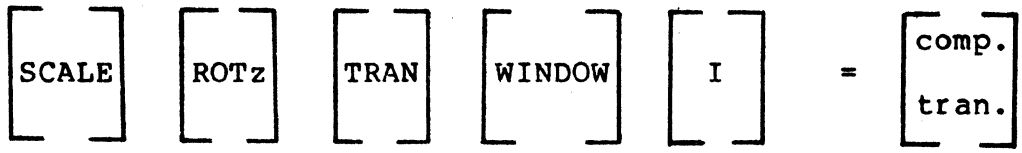
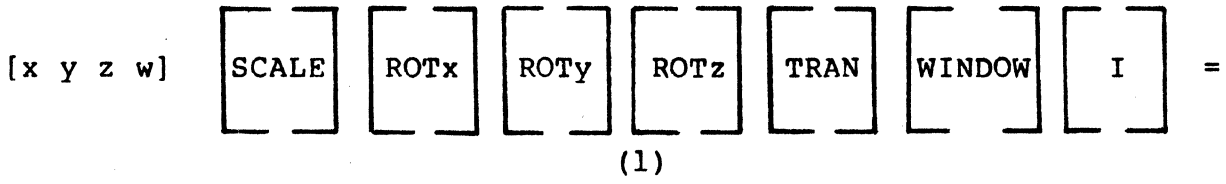


Figure 4.2-11

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

A comparison of Figures 4.2-10 and 4.2-11 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 4.2-12 shows the transformation of a data point by the transformations of Figure 4.2-10.



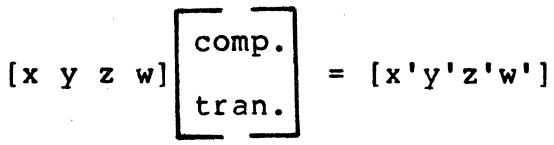
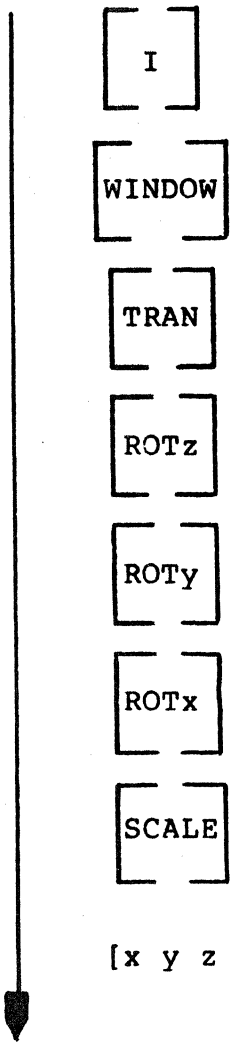


Figure 4.2-12

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

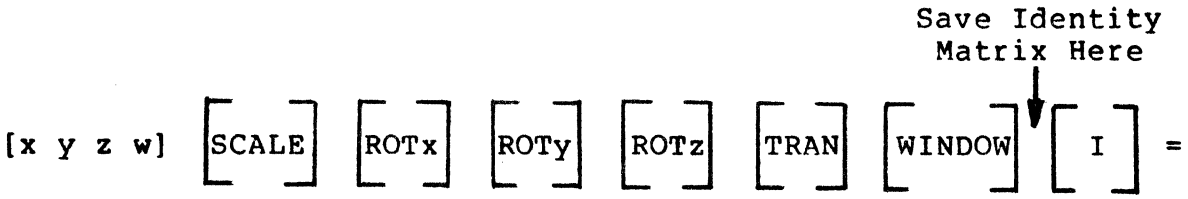
Once the transformations to be performed on a set of data have been diagrammed as in Figure 4.2-12, it is relatively simple to implement them in a graphics application program. Since the matrix concatenation implemented in hardware premultiplies the existing transformation matrix by the new component 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, translation, rotation and scaling. Note that this is the reverse order in which the transformations will be effectively applied to the drawn data. The most recent transformation applies first! This is illustrated in Figure 4.3-13 along with the FORTRAN subroutine calls required to implement this transformation sequence in a user program. Usually, the transformation sequence would be repeatedly executed by changing the parameters in the transformations to produce a dynamic picture. The PUSH and POP operations facilitate multiple use of compound matrices. For example, the identity matrix might be saved prior to the concatenation 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 4.2-14. 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 4.2-14 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 identity matrix. This is shown in Figure 4.2-15.



[I]	CALL PSINIT(2,0)
[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 4.2-13

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

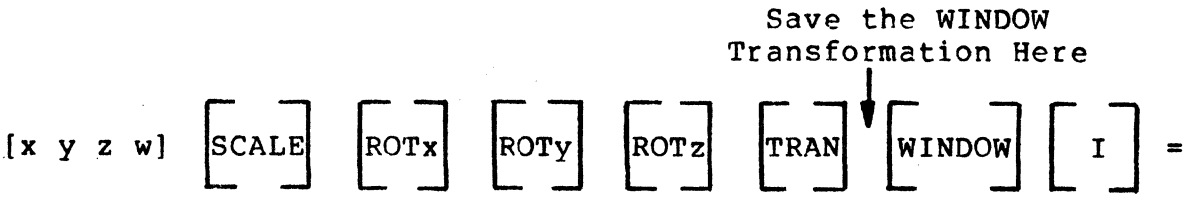


[x'y'z'w']

```
C   INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
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(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 IDENTITY MATRIX AND DISPLAY
C
          CALL POP
          CALL NUFRAM
          GO TO 100
```

Figure 4.2-14

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



[x'y'z'w']

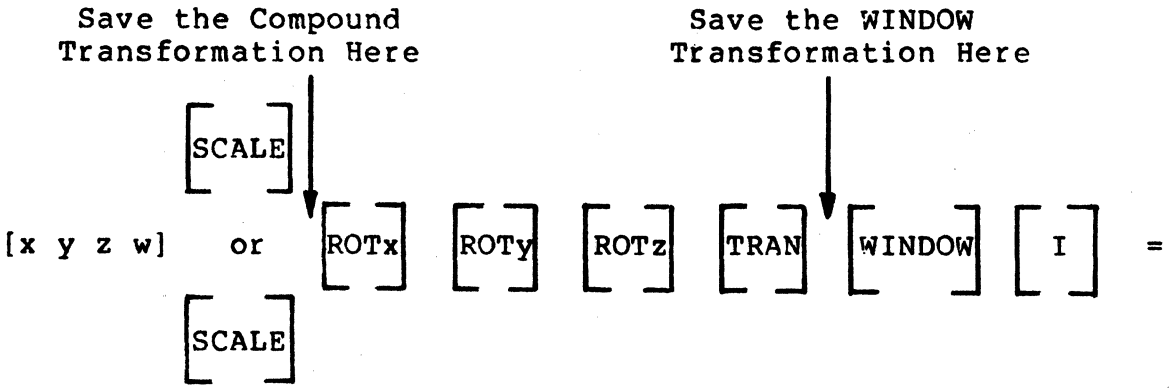
```
C    INITIALIZE PICTURE SYSTEM 2
C
           CALL PSINIT(2,0)
           CALL WINDOW(IWL,IWR,IWB,IWT,IH,IY,IE)
C
C    SAVE THE WINDOW TRANSFORMATION & BEGIN THE DISPLAY LOOP
C
100        CALL PUSH
C
C    MODIFY OR OBTAIN NEW TRANSFORMATION PARAMETERS
C
                .
                .
                .
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
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 4.2-15

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 4.2-15. 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 4.2-16. This ability to nest or stack transformations is available to eight levels in hardware, and may be extended by the software to any level required.

The capability of merely calling 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 associated linear transformations a powerful feature of PICTURE SYSTEM 2.




```
C    INITIALIZE PICTURE SYSTEM 2
C
           CALL PSINIT(2,0)
C
C    SET AND SAVE THE WINDOWING TRANSFORMATION
C
           CALL WINDOW(IWL,IWR,IWB,IWT,IH,IY,IE)
100        CALL PUSH
C
C    MODIFY OR OBTAIN NEW TRANSFORMATION PARAMETERS
                .
                .
                .
C    CONCATENATE THE TRANSFORMATIONS & DISPLAY THE DATA TWICE
C
           CALL TRAN(ITX,ITY,ITZ)
           CALL ROT(IAX,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 4.2-16

Diagrammed Nesting of Compound Transformations
and the Corresponding FORTRAN Code.

4.3 PROGRAM INITIALIZATION [PSINIT]

Program initialization for PICTURE SYSTEM 2 graphics applications programs consists of:

1. Calling the subroutine PSINIT to initialize the PICTURE SYSTEM 2 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.

4.3.1 Initialization of PICTURE SYSTEM 2 Hardware and Software

Typically, the first statement in a user applications program is the call to PSINIT* to initialize the hardware and software of PICTURE SYSTEM 2. A typical call to PSINIT is shown in Example 4.3-1:

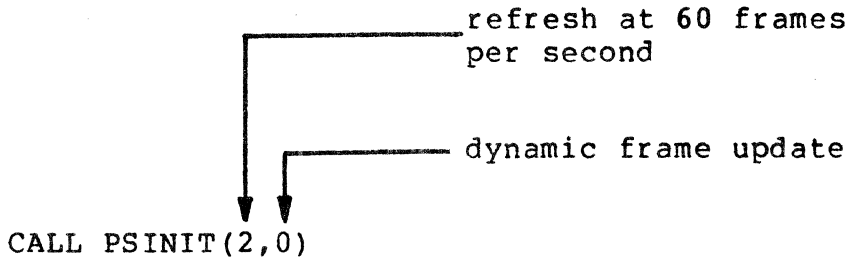


Figure 4.3-1

In this example, typical parameter values have been chosen: a refresh rate of 60 frames per second and the specification of a dynamic frame update rate.

---

*See Section 6.1 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 6.1:

```
[EXTERNAL ERRSUB]
CALL PSINIT(IFTIME,INRFSH,[ICLOCK],[ERRSUB],[ISTKCT]
1           ,[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, the frame refresh will occur upon the next 1/120 second interval after the frame refresh is completed. Table 4.3-1 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, INRFSH provides this capability. Example 4.3-2 demonstrates the calling sequence which specifies that frame update be done no sooner than every 30th of a second.

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

Example 4.3-2

TABLE 4.3-1

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 4.3-2 as follows:

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

Example 4.3-2

If a longer interval of time is required to generate a new frame, the update rate will automatically extend allowing 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 upon 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 4.3-3. 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-per-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 4.3-3

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 is called to report the occurrence of the user error. Typically, the system error subroutine is specified by default as shown in Examples 4.3-2 and 4.3-3. 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 4.3-4 demonstrates the use of a user error subroutine which avoids the loading of the system error subroutine, PSERRS.

```
EXTERNAL PSERRS
CALL PSINIT(2,0,,PSERRS)
  .
  .
  .
END


SUBROUTINE PSERRS
STOP
RETURN
END
```

Example 4.3-4

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 8, the number available within the Picture Processor that are supported by the Graphics Software Package. If this is required, ISTKCT is the number of additional levels of matrix stack space that are required.

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

IFMCNT is an optional parameter which, if specified, will be incremented by one each 1/120 second.

TABLE 4.3-2

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(2,0,,,,,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 & THEN CONTINUE
C
            X=120./IFMCNT
            PRINT 2000, X
2000        FORMAT('FRAME UPDATE RATE=',F5.2,
                   'FRAMES PER SECOND')
            GO TO 100
```

Example 4.3-5

4.3.2 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 the full screen.

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.

4.3.2.1 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 4.3-6:

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

Example 4.3-6

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.

4.3.2.2 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 4.3-7:

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

Example 4.3-7

In this example, the parameters IX and IY are the variables containing 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.

4.3.2.3 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.
2. CALL TABLET to initiate automatic tablet update.
3. CALL CURSOR to initiate automatic cursor display.

Example 4.3-8 shows the use of automatic tablet and cursor modes.

```
C
C   INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
C
C   INITITATE AUTOMATIC TABLET AND CURSOR MODES
C
          CALL TABLET(1,IX,IY,IPEN)
          CALL CURSOR(IX,IY,1)
C
C   BEGIN DISPLAY LOOP
C
            .
            .
            .
```

Example 4.3-8

4.3.3 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 restarted or has a programmed restart 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 4.3-1 illustrates a suggested placement of the user variable initialization process.

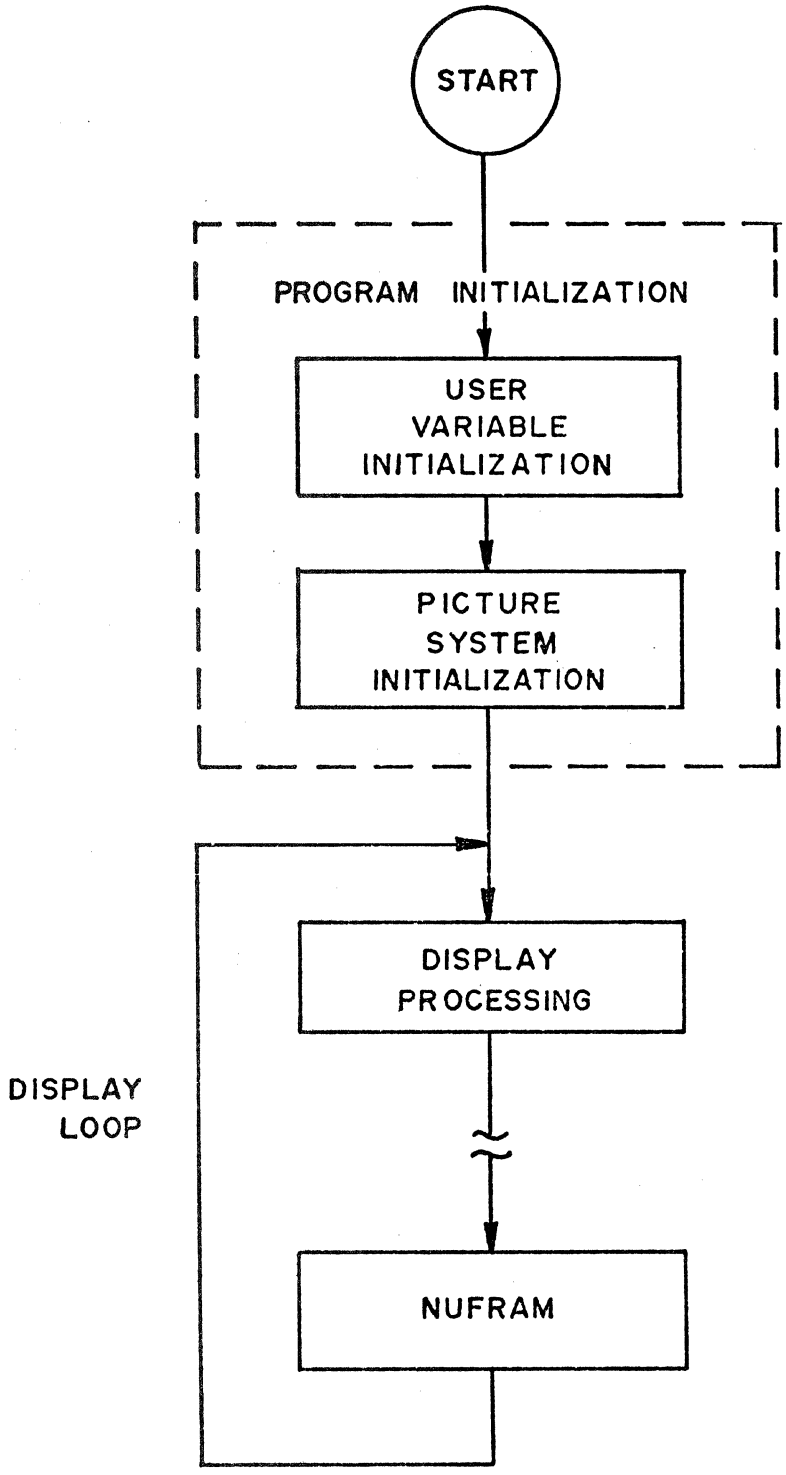


Figure 4.3-1

Suggested Program Initialization Structure

4.4 VIEWPORTS [VWPORT]

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

Figures 4.4-1a and b illustrate the two- and three-dimensional display of data which is mapped into the viewport. A viewport is specified for PICTURE SYSTEM 2 by calling the VWPORT subroutine. The following is the VWPORT calling sequence specification of Section 6.1:

```
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).

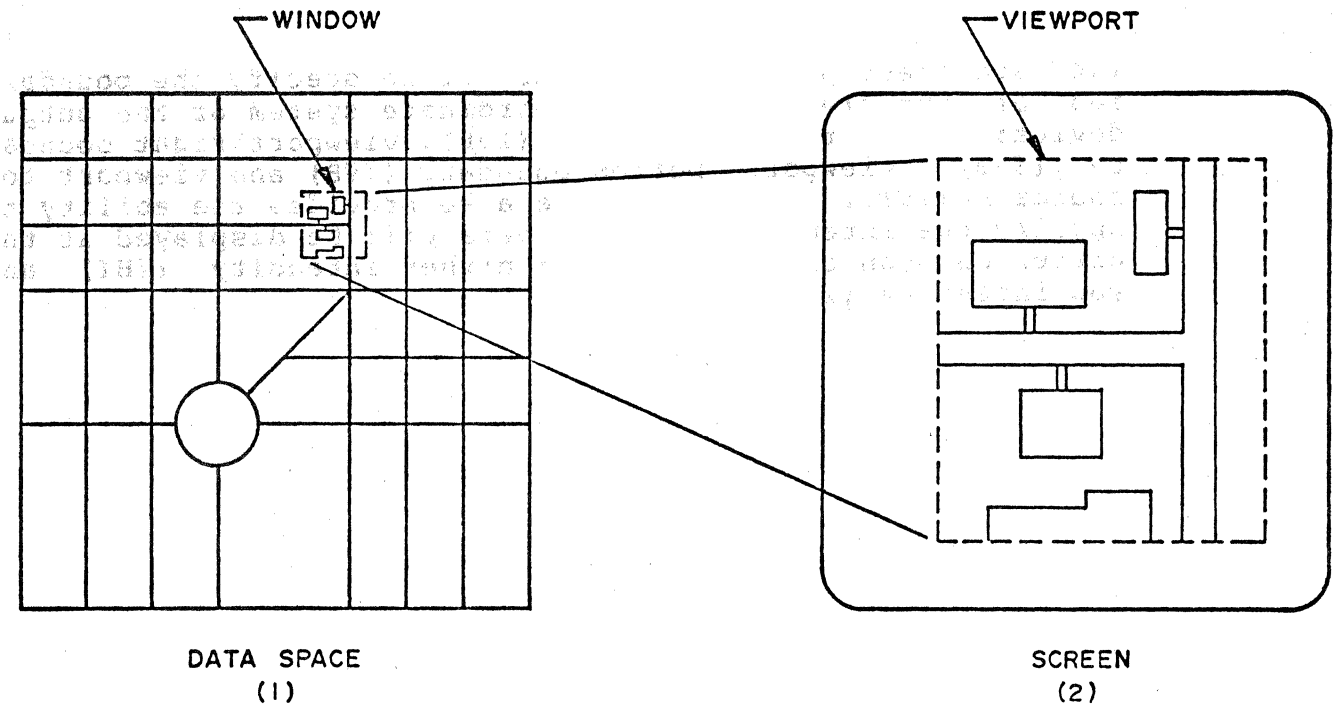
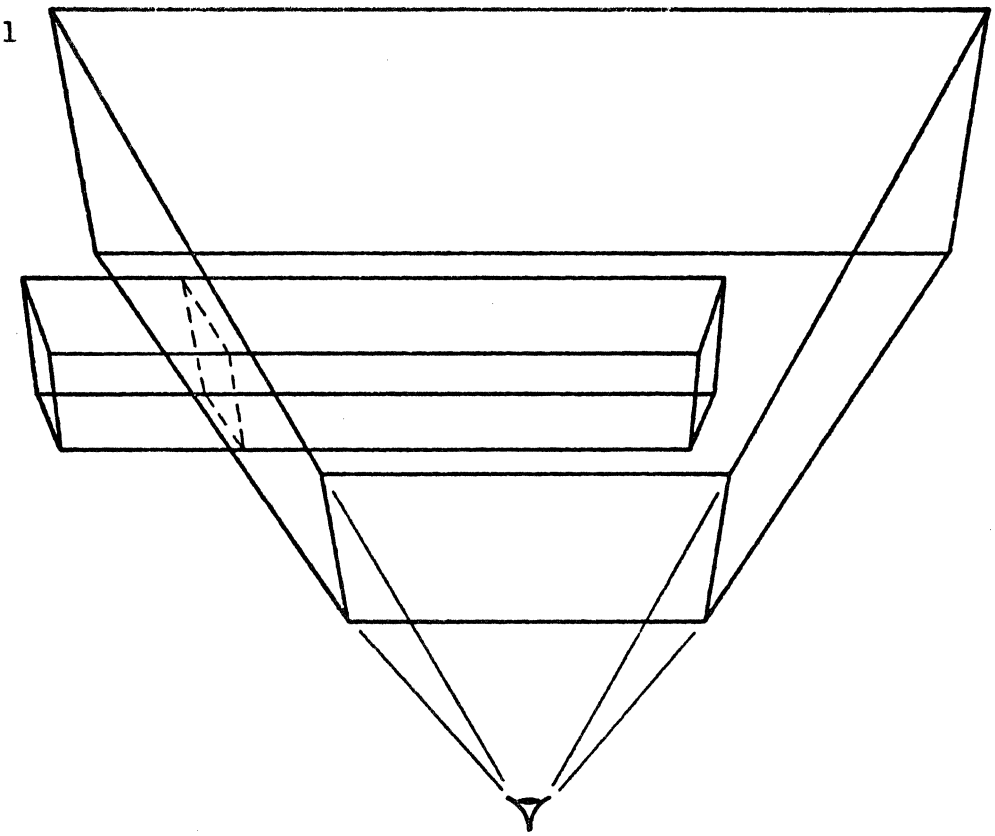
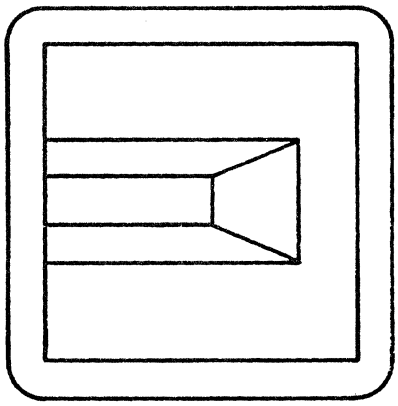


Figure 4.4-1a

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 4.4-1b

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.

4.4.1 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 4.4-1:

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

Example 4.4-1

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 4.4-2. 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 4.5.

4.4.2 Multiple Viewports

The Picture Display may be used for simultaneous display of different pictures. For example, the screen could be used to simultaneously display the entire street map of Figure 4.4-1a and the magnified portion of the map as illustrated in Figure 4.4-3. The statements used to accomplish this are shown in Example 4.4-2. The use of multiple viewports on one display is a powerful feature of PICTURE SYSTEM 2. The ability of PICTURE SYSTEM 2 to display data on up to six 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.

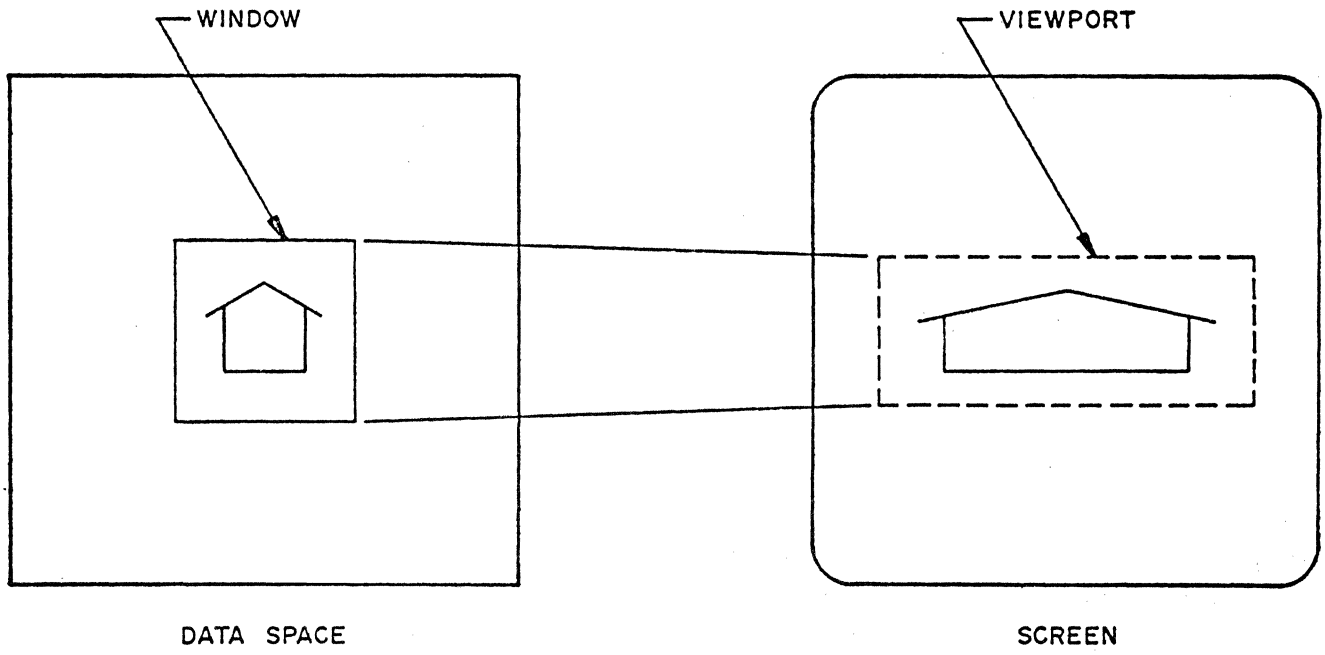


Figure 4-4.2

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

```
C
C   INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
C
C   SAVE THE INITIAL TRANSFORMATION
C
100       CALL PUSH
C
C   SET THE VIEWPORT FOR THE TEXT & ANNOTATE THE DISPLAY
C
          CALL VWPORT(-2048,2047,-2048,2047,255,255)
          CALL MOVETO(1000,16384)
          CALL TEXT(17,'ENTIRE STREET MAP')
          CALL MOVETO(8192,0)
          CALL TEXT(13,'MAGNIFICATION')
C
C   DISPLAY THE ENTIRE MAP
C
          CALL VWPORT(-2048,0,0,2048,255,255)
          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 & 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
          END
```

Example 4.4-2

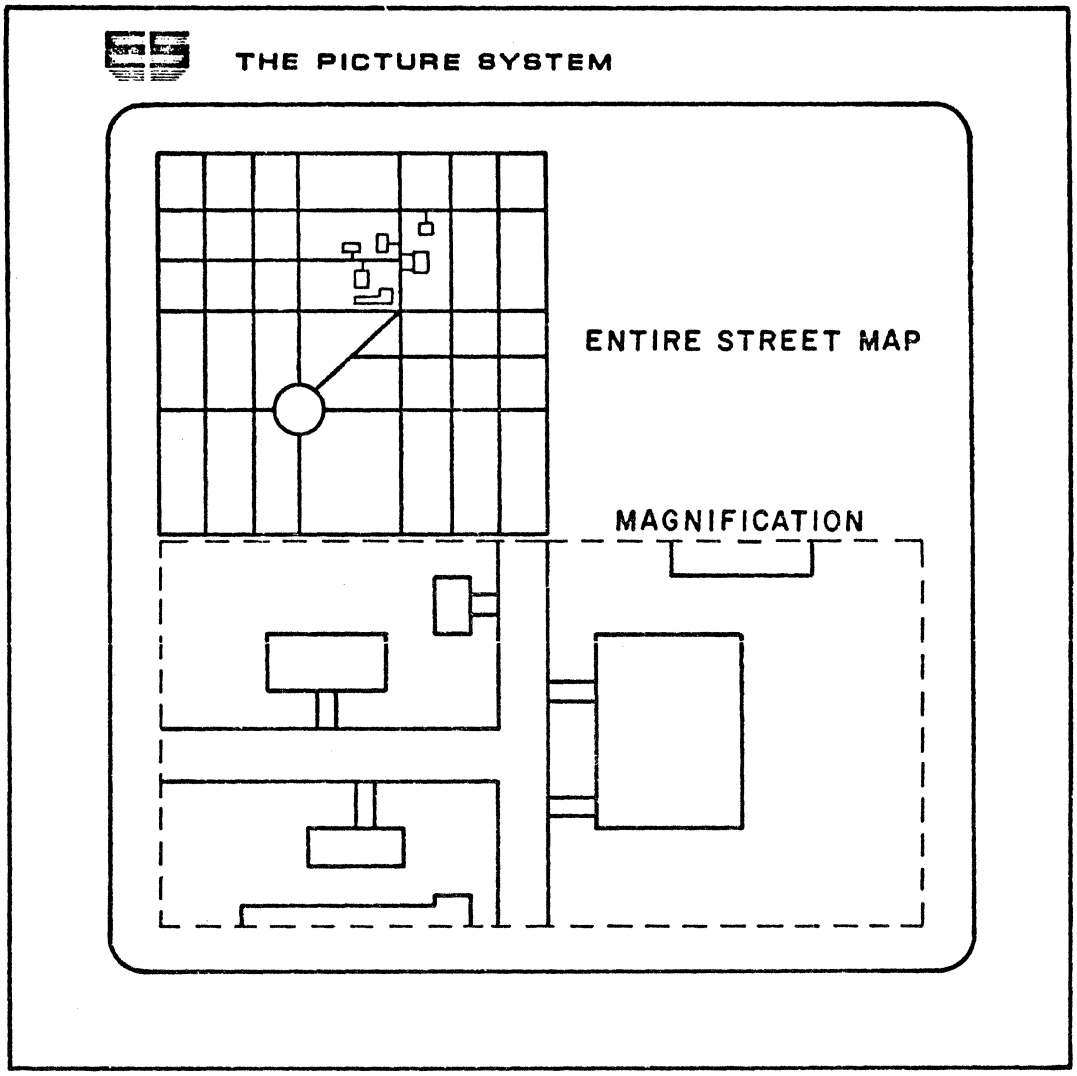


Figure 4.4-3

Simultaneous use of the screen to display an entire street map, a portion of the map magnified and text annotation using three viewports to specify the portion of the screen to be used.

4.4.3 Depth-cueing

A heightened sense of perspective may be imparted to three-dimensional objects by specifying that depth-cueing be performed. By specifying differing hither and yon intensities when calling the VWPORT subroutine, depth-cueing provides the ability to vary the intensity of lines as the lines become "further away". The maximum depth-cueing effect is obtained by specifying a maximum hither intensity (255) and a minimum yon intensity (0). Example 4.4-3 shows the use of the VWPORT subroutine to specify depth-cueing.

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

Example 4.4-3

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

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

IHI,IYI: 255 to 0

4.5 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 (i.e., the position of the observer as shown in Figure 4.5-1a, 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 6.1:

```
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.

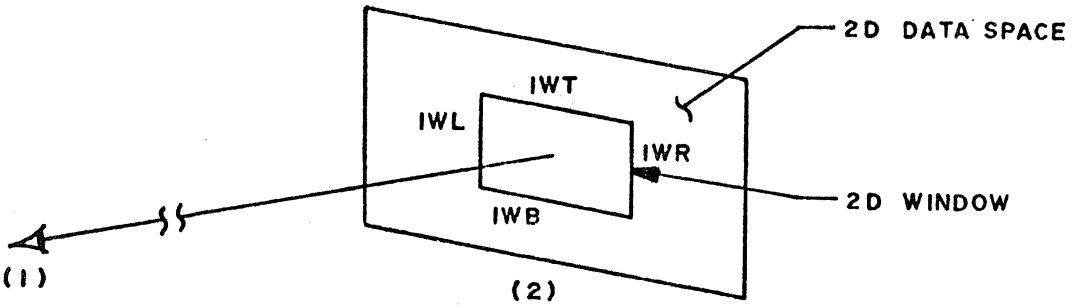


Figure 4.5-1a

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.

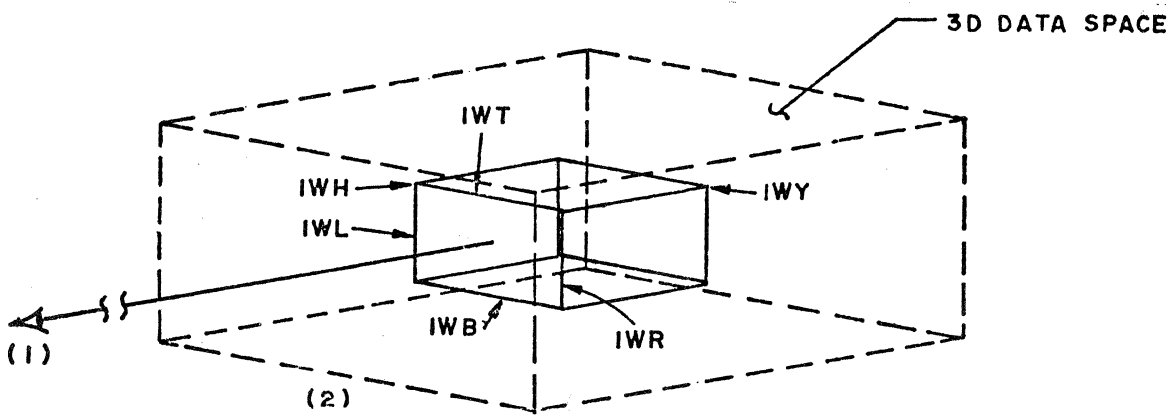


Figure 4.5-1b

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 three-dimensional 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).

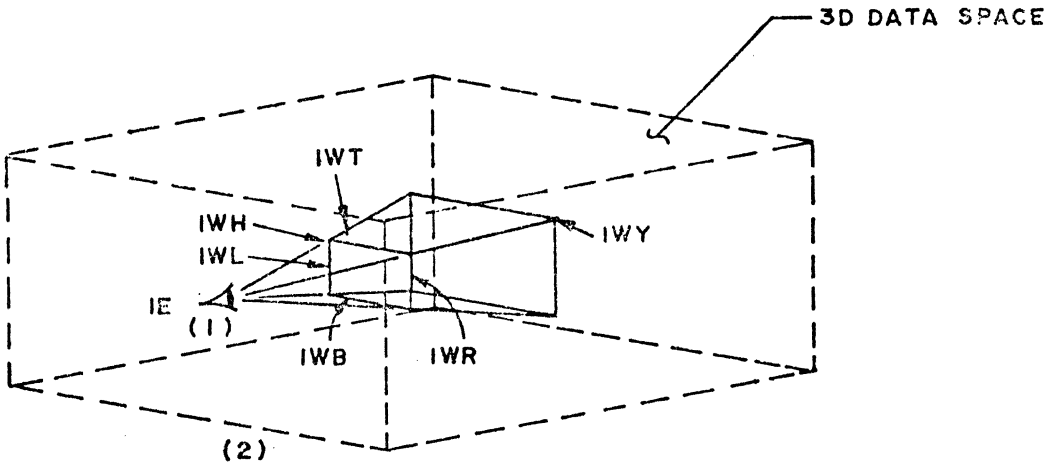


Figure 4.5-1c

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 three-dimensional 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).

4.5.1 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 4.5-1 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 4.5-1

4.5.2 Three-Dimensional Orthographic Views

An extension of the two-dimensional WINDOW call is to provide six parameters for the definition of a rectangular parallelopiped (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. Note that only the x and y coordinates of the displayed lines and dots affect the picture. The z coordinate has no effect except perhaps in determining 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 4.5-2.

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

Example 4.5-2

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 4.5-3.

```
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,-20000,-10000,-5000,
     1            5000,-5000,16384)
```

Example 4.5-3

4.5.3 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 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 normally 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 4.5-2. 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*:

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

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 degrees) or a fish eye camera lens (viewing angle > 40 degrees).

---

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

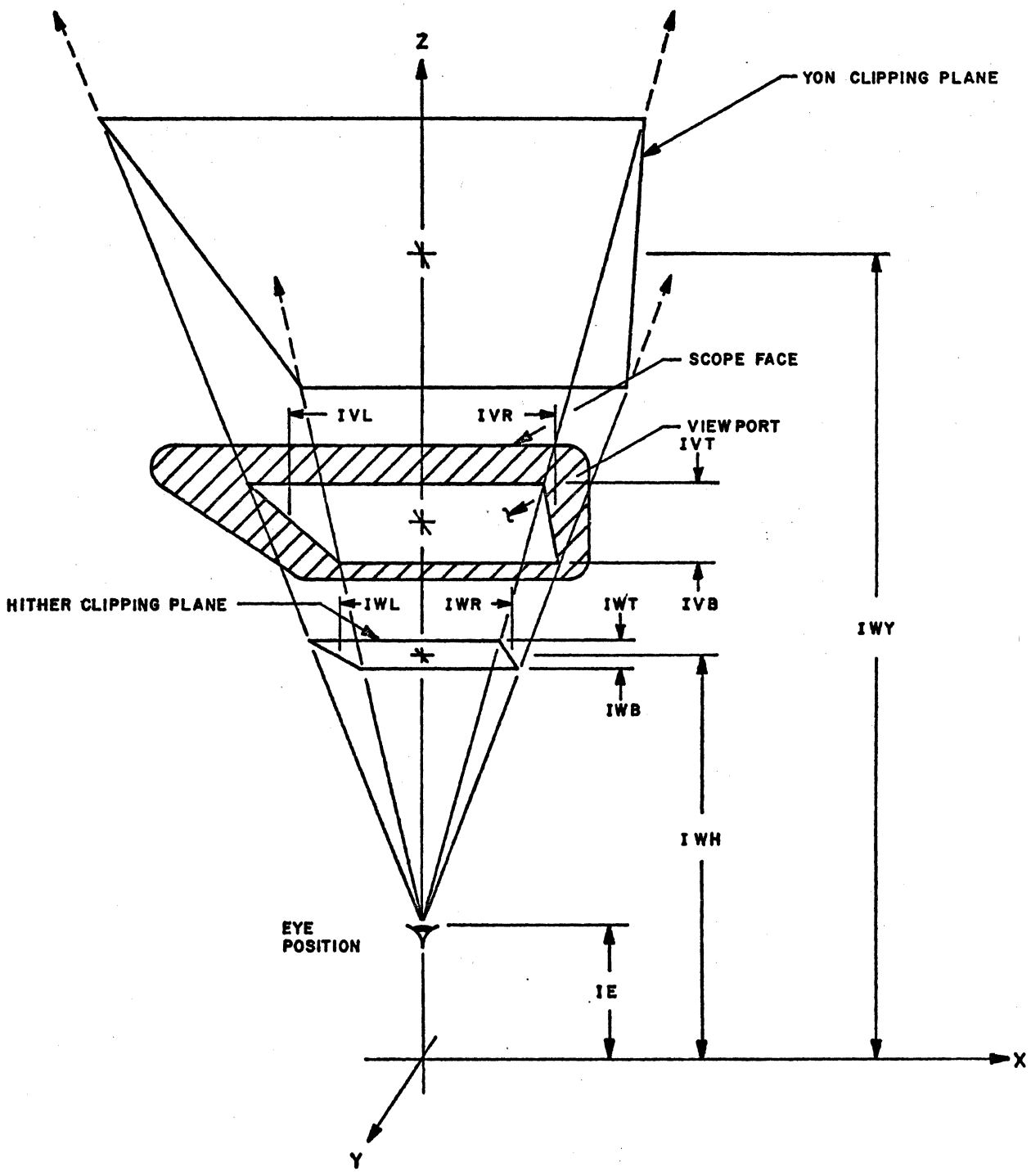


Figure 4.5-2

The Frustrum of Vision as defined
by the WINDOW subroutine

4.5.4 Non-Square Windows and Viewports

In specifying a WINDOW (square or non-square) the user should display data within a viewport of a corresponding shape to ensure that the data are 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:

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

The user should maintain this equality for all types of windowing-- two-dimensional, three-dimensional orthographic and three-dimensional perspective views. Note that the angular width of the frustrum of vision must be approximately equal to the angle through which the viewport is observed. 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 4.5-3. 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 corresponding ratio:

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

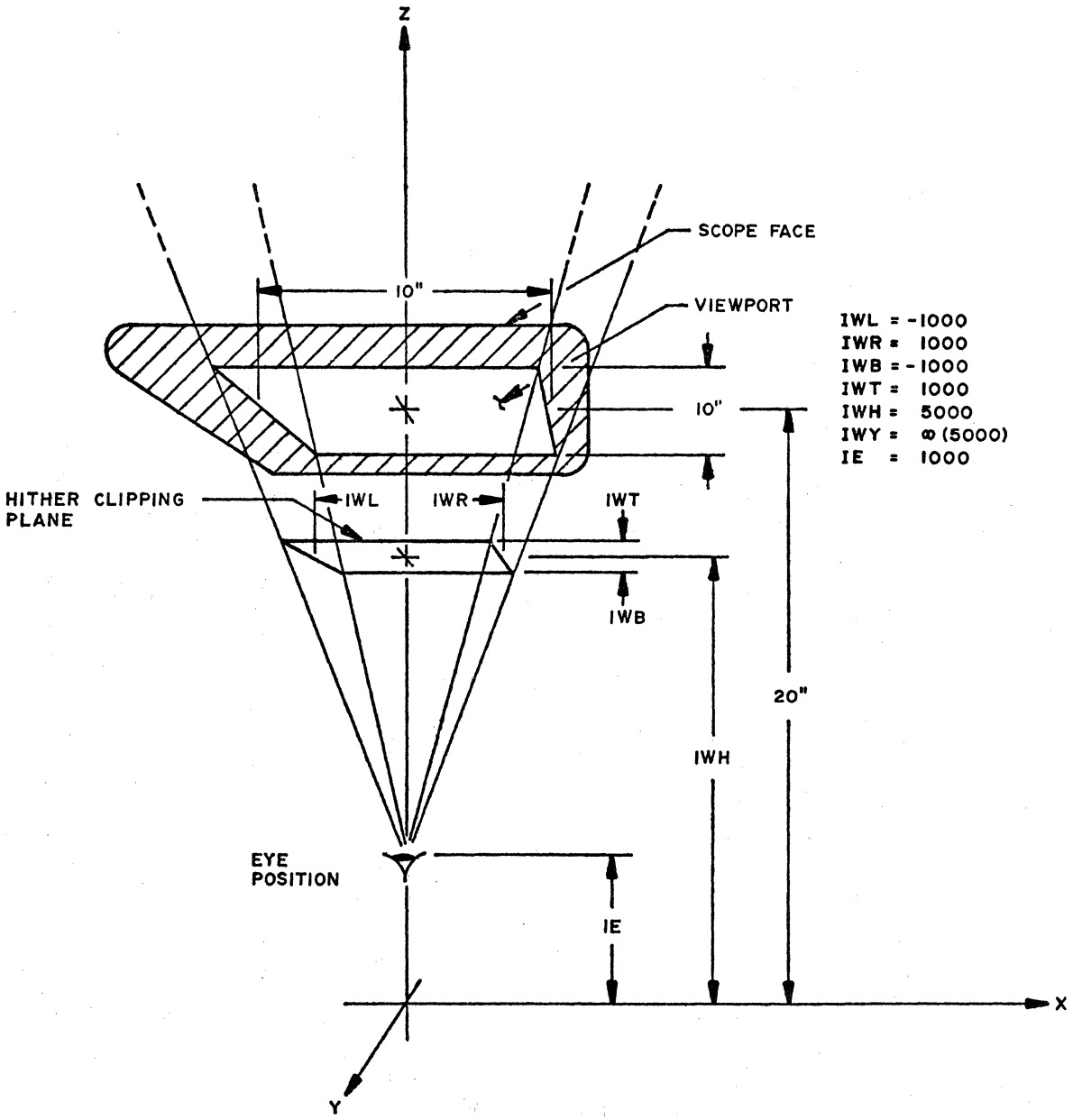


Figure 4.5-3

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 4.5-4.

```
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 TRANS(ITX,ITY,ITZ)
             .
             .
             .
```

Example 4.5-4

4.5.5 Sectioning

For most applications, it is undesirable 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 4.5-4, 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 by simply 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 equaling on it, and the above difficulty is thus avoided.

4.5.6 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 4.5-5, 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 4.5-5

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

Both sectioning and 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 2 feature allows data to be displayed as it would appear when illuminated by a light source, allowing data to decrease rapidly in intensity with increase in apparent distance.

4.5.7 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 4.5-6.

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 4.5-7. 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 4.5-8.

```
            .
            .
            .
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 VIEWPOINT 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 4.5-6

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

Example 4.5-7

```
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 4.5-8

4.5.8 Placement of the Hither and Yon Planes

The normal relationship of the three z-values appearing in the WINDOW calling sequence is (in order of increasing z-value):

IWE (eye)
IWH (hither clipping plane)
IWY (yon clipping plane)

For two-dimensional windowing, the hither plane is arbitrarily placed at z=0 and the yon plane at IW (normally 32767) with the eye assumed to be at minus infinity. 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 explictly specified by the arguments in the WINDOW call. As described in the previous section, reversing the order of the above z-values provides a view in the negative direction in z. There exist, however, several variations to the above z-value ordering convention, each of which has its own unique set of advantages. These are detailed below.

4.5.8.1 Reversing the Hither and Yon Planes

To maintain utmost precision of transformed data, the hither and yon planes should not be given unnecessarily extreme positions (i.e., the hither plane should ordinarily not be placed immediately in front of the eye position). Maximal precision is maintained if the distance between the hither and yon planes (and between the hither plane and the eye) is

in the same order of magnitude as the width and height of the hither boundaries.

There are times, however, when hither clipping is desired very close to the eye. This may be easily accomplished by simply reversing the roles of the hither and yon planes (i.e., by placing the yon plane between the eye and the hither plane, thereby using the yon plane for hither clipping). This method works properly without a loss of precision since under these circumstances the distance from the hither plane to the eye approximately equals the distance from the hither to the yon plane. This method, however, necessitates two changes in procedure. First, the WINDOW boundaries (IWL, IWR, IWB and IWT) are still measured at the hither clipping plane (IWH), which is now at the back of the frustrum of vision. Second, because the roles of the hither and yon clipping planes are reversed, the user must also reverse the hither and yon intensity values (IHI and IYI) in the VWPORT subroutine call.

4.5.8.2 Superimposing the Yon Plane on the Hither Plane

By convention, if IWY is specified to be equal to IWH, it is set so as to be at infinity on the side of IWH opposite the eye. Thus, no yon clipping will result and objects will still be visible at any distance from the eye, although they may be dim if depth-cueing is used.

4.5.8.3 Superimposing the Eye on the Hither Plane

As described in Section 4.5.2, if IWE is made equal to IWH, the eye is set so as to be at minus infinity, thus producing an orthographic view.

4.5.8.4 Superimposing the Yon Plane on the Eye

When IWY is set exactly equal to IWE, no hither or yon clipping will occur. Thus, an unclipped perspective view from the eye position to infinity may be obtained. However, under these circumstances no depth-cueing is possible and all objects are displayed at the VWPORT hither intensity. Example 4.5-9 illustrates the use of this technique. If, in addition, the eye position parameter (IWE) is also set equal to IWH, the view is then orthographic with no hither or yon clipping.

The effects of the techniques described above are summarized in Table 4.5-1.

```
C
C    PLACEMENT OF HITHER AND YON PLANES TO AVOID
C    HITHER CLIPPING
C
           CALL WINDOW(-1000,1000,-1000,1000,
         1              10000,-4000,-4000)
C
C    YON CLIPPING WILL OCCUR AT 10000!
C
                .
                .
                .
```

Example 4.5-9

Table 4.5-1

Section	Action	Result
4.5.8.1	IWE<IWY<IWH	Allows clipping close to the eye with no display degradation.
4.5.8.2	IWH=IWY	No yon clipping (i.e., yon plane at infinity).
4.5.8.3	IWE=IWH	Orthographic view.
4.5.8.4	IWE=IWY	No hither or yon clipping (i.e., visibility from the eye to infinity and no depth-cueing).
4.5.8.4	IWE=IWH=IWY	Orthographic view with no hither or yon clipping and no depth-cueing.

4.6 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 an 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.

Following is the ROT calling sequence specification of Section 6.1:

```
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 degrees 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 greater precision without the need for special floating-point hardware or increased execution time due to software floating-point calculations.

Table 4.6-1 shows some common angles and their corresponding IANGLE values. Example 4.6-1 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 round-off 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 of the left-handed coordinate system.

Table 4.6-1

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

```
C    ONE DEGREE
     DATA I/182/
C
C    INITIALIZE  PICTURE SYSTEM 2
C
           CALL PSINIT(2,0)
           IANGLX = 0
           IANGLY = 0
           IANGLZ = 0
C
C    PERFORM THE PERSPECTIVE TRANSFORMATION
C
           CALL WINDOW(-1000,1000,-1000,1000,
          1            -1000,10000,-5000)
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 4.6-1

4.7 TRANSLATION [TRAN]

A translation transformation is applied to coordinate data using the TRAN subroutine, to cause a translation in the X, Y and Z directions of the data space of all subsequent data drawn. Following is the TRAN calling sequence specification of Section 6.1:

```
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 before the ROT subroutine*. This order is illustrated by Example 4.7-1.

```
          .
          .
          .
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 4.7-1

---

*See Section 4.2.3 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 (+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 +2**30.

Example 4.7-2 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 4.7-2

4.8 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. Following is the SCALE calling sequence specification of Section 6.1:

```
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 are to be scaled. For example, if an object were to be decreased in size by 1/2 in the X, Y and Z axes, 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 4.8.3.

4.8.1 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 of the object the scaling should be performed before the object has been rotated about its origin. This means that the SCALE subroutine should be called after the ROT subroutine*. This order is illustrated by Example 4.8-1.

---

*See Section 4.2.3 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 4.8-1

4.8.2 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 symmetrical 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 4.8-2.

```
            .
            .
            .
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 4.8-2

4.8.3 Scaling Using the Homogenous 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 4.8. 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 ±2**30.

Example 4.8-3 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 4.8-3

4.9 DATA DISPLAY

Data that is displayed on PICTURE SYSTEM 2 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 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.

NOTE

The Picture Display is protected by phosphor protect circuitry to help prevent burning the face of the CRT. If the electron beam does not move appreciably in the X and Y axes of the display, then this circuitry will not allow the intensification of the beam. Thus, to ensure the display of data (both lines and characters), the beam must be moved from one corner of the screen diagonally to the other corner. For example, if only a single line of text were to be displayed, this beam deflection would be required.

4.9.1 Display of Lines and Dots [MOVETO,MOVE,LINETO,LINE,DOTAT,DOT,DRAW2D,DRAW3D,DRAW4D]

The display of a single two- or three-dimensional point or line is accomplished by calling the MOVETO, MOVE, LINETO, LINE, DOTAT and DOT subroutines. Following are the calling sequence specifications for these subroutines of Section 6.1:

```
CALL MOVETO(IX,IY[,IZ])
```

```
CALL MOVE(IX,IY[,IZ])

CALL LINETO(IX,IY[,IZ])

CALL LINE(IX,IY[,IZ])

CALL DOTAT(IX,IY[,IZ])

CALL DOT(IX,IY[,IZ])
```

These subroutines provide the means for positioning to, drawing a line to, or drawing a dot at a specified x,y and optionally z, position. These subroutines are very useful for positioning for text display or for a small number of lines. However, if many lines or dots are to be drawn then they should be defined or created as a data set. The display of two- or three-dimensional data sets as lines or dots is accomplished by calling the DRAW2D, DRAW3D or DRAW4D subroutines. Following are the DRAW2D, DRAW3D and DRAW4D calling sequence specifications of Section 6.1:

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

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

CALL DRAW4D(IDATA,NUM,IF1,IF2)
```

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.
3 = set base-offset-offset-offset-etc.
4 = offset 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 4.9-1 shows the simplistic data set:

```
X1,Y1 = 0,1
X2,Y2 = -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.

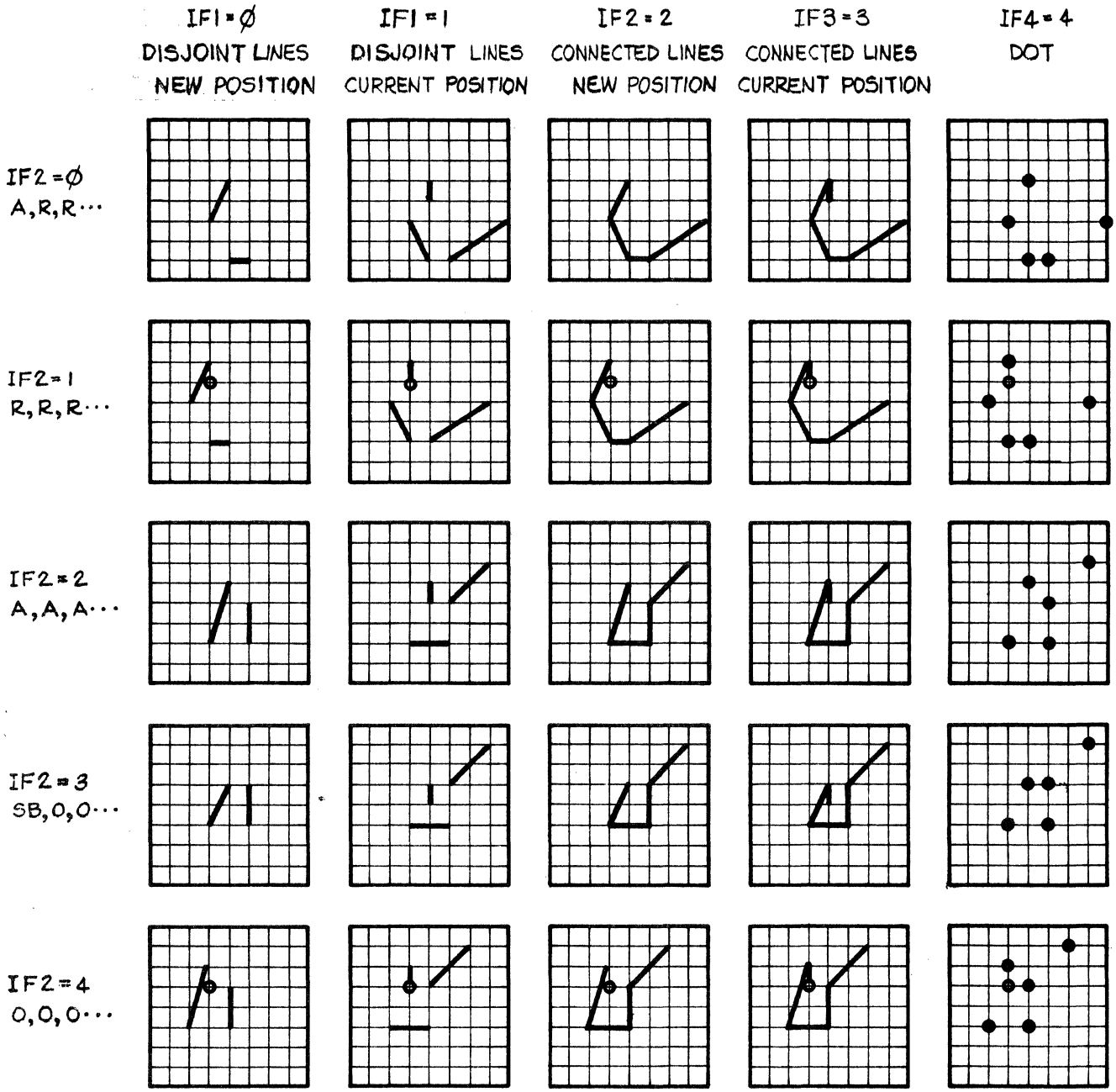


Figure 4.9-1

The Drawing Functions of the DRAW2D and DRAW3D Subroutines
NOTE: A Setpoint is Indicated by the O Symbol.

It is assumed for each of these drawings that the current position before the draw begins will be 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.

4.9.1.1 Drawing Two-Dimensional Data

Two-dimensional data points are 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 are really three-dimensional data which resides in a constant plane and may therefore be ROTated, TRANslated, etc. as a three-dimensional data set. Example 4.9-1 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,-10000,
     1 10000,-10000,10000,10000/
        .
        .
        .
C
C   NOW DRAW THE 2D DATA
C
      CALL DRAW2D(IDATA,5,2,2,16384)
        .
        .
        .
```

Example 4.9-1

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*. When viewed through a two-dimensional window, IZ = 0 will cause the data to be displayed at the maximum intensity** 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 64 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>IL>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 4.9-2 would be required.

---

*This assumes that a viewport was specified which allows intensity variation (i.e., depth-cueing).

**This is because data are 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 4.9-2

4.9.1.2 Drawing Three-Dimensional Data

Three-dimensional data are defined within a data set as a series of x,y,z coordinates with a constant (or default) w coordinate. Example 4.9-3 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,
     2            10000,10000,16384/
        .
        .
        .
C
      CALL DRAW3D(IDATA,5,2,2)
```

Example 4.9-3

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 of the data displayed*. When viewed orthographically, the intensity of the data displayed varies linearly from the hither to yon clipping planes. When viewed in perspective, however, the intensity of the data displayed varies reciprocially from the hither to yon clipping planes.

4.9.2 Display of Characters

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

1. Calling the CHARSZ subroutine to specify the size and orientation in which the characters are to appear.

2. Calling the MOVETO, DRAW2D, or DRAW3D subroutine to position to where the text is to be displayed.

3. Calling the TEXT subroutine to output the characters to be displayed.

---

*This example is equivalent to the two-dimensional case of Example 4.9-1 and would produce the same image if displayed.

Following are the CHARSZ and TEXT calling sequence specifications of Section 6.1:

```
CALL CHARSZ(ISIZE,ITILT)

CALL TEXT(NCHARS,ITEXT)
```

4.9.2.1 Character Size and Orientation [CHARSZ]

The CHARSZ subroutine may specify a total of 8 character sizes and 2 orientations for text display. The character sizes available are shown in Figure 4.9-2. As this figure illustrates, the sizes may range from .08 cm (.03 inches) to 1.88 cm (.74 inches). The characters which are displayed may be oriented either horizontally or vertically depending upon the value of the ITILT parameter as shown in Figure 4.9-3a and b. The CHARSZ 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 CHARSZ subroutine. Therefore, if the default character specification (horizontal .68 cm (.27 inches) characters) is sufficient for the user's application, the CHARSZ subroutine need never be called. 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. Thus, the default values may no longer be relied upon, but must be explicitly specified.

Figure 4.9-2

Standard Character Sizes in Inches.

HORIZONTAL

Figure 4.9-3a

Horizontal Character Orientation (ITILT=0)

VERTICAL

Figure 4.9-3b

Vertical Character Orientation (ITILT≠0)

4.9.2.2 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, characters are displayed at the maximum intensity available and a two-dimensional window is used to make positioning and intensity specification simple. An example of a two-dimensional window is the one which is initialized by PSINIT. The boundaries of this window* 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 is then 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 4.9-4 shows how this may be done.

---

*This two-dimensional window is merely an identity matrix.

```
C
C    INITIALIZE PICTURE SYSTEM 2
C
           CALL PSINIT(2,0)
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 MOVETO(-32767,0)
           CALL TEXT(16,'PICTURE SYSTEM 2')
C
C    SET THE VIEWPORT AND 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 4.9-4

4.9.2.3 Text Output [TEXT]

Text is output for display on PICTURE SYSTEM 2 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 4.9-5.

```
CALL TEXT(16,'PICTURE SYSTEM 2')
```

Example 4.9-5

This will cause the ASCII character string "PICTURE SYSTEM 2" 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 CHARSZ 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 with their ASCII codes, in Table 4.9-1. When these codes are encountered during the refresh cycle, the Character Generator is called to stroke the character relative to the current position of the beam on the Picture Display.

The character is always stroked (in horizontal or vertical mode) relative to the lower-left position of the area in which any given character is defined. Figures 4.9-4a and b show the area in which a character is displayed by the Character Generator.

Table 4.9-1
95 Displayable ASCII Characters

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	del

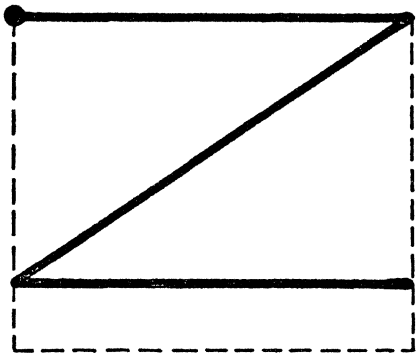


Figure 4.9-4a

Horizontal Character Stroking Relative to the Current Position of the Beam

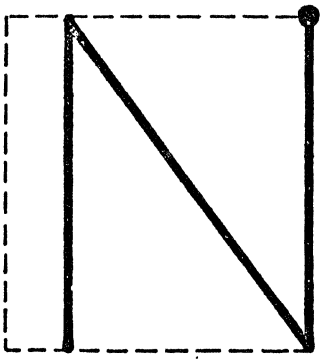


Figure 4.9-4b

Vertical Character Stroking Relative to the Current Position of the Beam

4.9.3 Instancing [INST,MASTER]

As pointed out previously, the data which comprises a scene is made up of a set of primitives (i.e., lines, dots and strings of text). These primitives are used, either singularly or collectively, to generate objects which comprise the scene. The ability to create and easily position more complex primitives greatly simplifies construction of a scene. As an example, let a transistor symbol be 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. 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, is called instancing. This constitutes one of the more powerful tools 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 degrees from its original position, etc.). This call may be omitted otherwise.

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

Following is the INST calling sequence specification of Section 6.1:

```
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-argument parameter list such as that shown in Example 4.9-6.

```
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 4.9-6

If the instance is three-dimensional, INST is called with a six-argument parameter list such as that shown in Example 4.9-7. Note that if any calls to ROT immediately follow the INST call, the center of rotation is defined to be at the center of the hither surface of the instance space.

```
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 4.9-7

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 4.9-8 shows the FORTRAN subroutine, HOUSE2 referenced previously in Example 4.9-6, 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,
     1           -32000,-32000, -32000,16000, 0,32000/
C
C AND A DOOR
C
      DATA DOOR/4000,-16000, 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,6,2,2,0)
C
C AND THE DOOR
C
      CALL DRAW2D(DOOR,5,2,20)
C
C NOW RESTORE THE TRANSFORMATION MATRIX AND RETURN
C
      CALL POP
      RETURN
      END
```

Example 4.9-8

The following discussion describes each of the parts, illustrated by Example 4.9-8 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 identical). This enclosure is used as 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. Following are the MASTER subroutine calling sequence specifications of Section 6.1:

```
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 centered in 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 MASTER enclosure is not defined as large as possible. 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 to fill the entire 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 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.

While the MASTER call transformation is identical to that of a WINDOW call, data points are not clipped at the master boundaries as 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 preceded 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 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.

Using this technique, two- and three-dimensional primitives may be defined and libraries of these "master copies" maintained. Example 4.9-9 shows a simple program which uses the primitive defined by Example 4.9-8. Figure 4.9-5 shows the mapping performed by PICTURE SYSTEM 2 during instancing.

```
C
C INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
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 DEGREES)
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 4.9-9

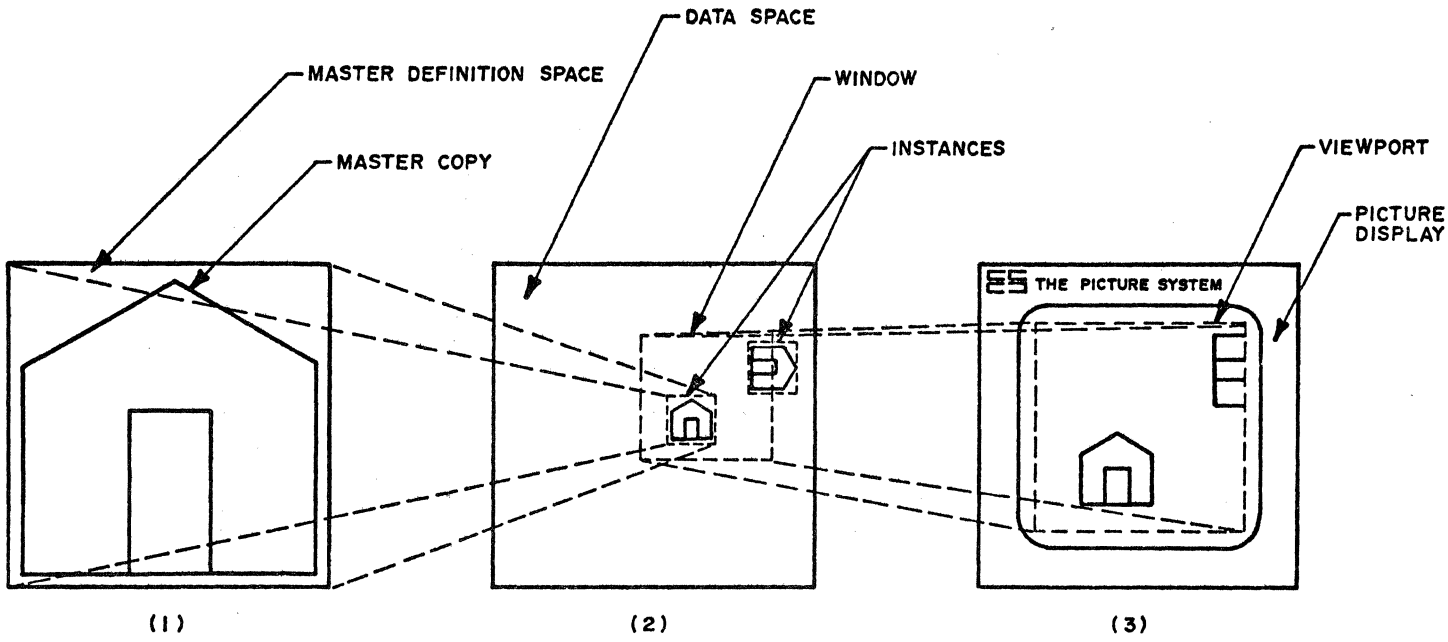


Figure 4.9-5

Illustration of the Mapping of the Instances to the Window and the Window to the Viewport of Example 4.9-11 Showing (1) the master copy definition of Example 4.9-10, (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.

4.9.4 Display Modes

All lines, dots, characters and instances may be displayed in blinking display mode on one to six Picture Displays (or any combination thereof) simultaneously. Also, lines may be displayed in any of seven line textures. 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.

4.9.4.1 Textured Display Mode [TXTURE]

When PSINIT is called to initialize PICTURE SYSTEM 2, 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 TXTURE subroutine to initiate a dashed line mode. Following is the TXTURE calling sequence specification of Section 6.1:

```
CALL TXTURE(ISTAT)
```

The parameter passed to this subroutine specifies the mode in which all subsequent lines will be drawn, until PSINIT is called to re-initialize the system or TXTURE is called to reset the line status. If TXTURE is called with ISTAT=0, all subsequent data output will be drawn in solid line mode. A call to TXTURE with ISTAT = 1 - 6 causes one of the six line textures available to be selected. Figure 4.9-6 shows the various line textures that are available by calling TXTURE.

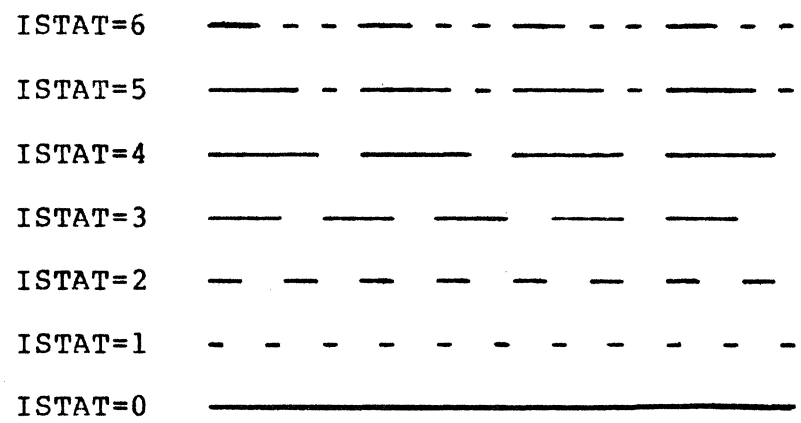


Figure 4.9-6

PICTURE SYSTEM 2 Line Textures

4.9.4.2 Blink Display Mode [BLINK]

When PSINIT is called to initialize PICTURE SYSTEM 2, the display mode is set allowing all subsequent data output to be drawn in non-blink mode. This mode will remain in effect until the user calls the BLINK subroutine to initiate blink display mode. Following is the BLINK calling sequence specification of Section 6.1:

```
CALL BLINK(ISTAT)
```

The parameter passed to this subroutine specifies the mode in which all subsequent lines will be drawn, 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 non-blinking. If BLINK is called with ISTAT≠0, all subsequent data output will blink at the approximate rate of 30 times per minute. Example 4.9-10 shows how blink mode may be used.

```
C
C   INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
C
C   BEGIN THE DISPLAY LOOP, SET THE BLINK MODE
C
100      CALL BLINK(1)
C
C   NOW DISPLAY THE BLINKING DATA
C
          CALL DRAW2D(IARRAY,12,2,2,0)
           .
           .
           .
C
C   RESET THE BLINK MODE
C
          CALL BLINK(0)
C
C   NEW FRAME
C
          CALL NUFRAM
          GO TO 100
```

Example 4.9-10

4.9.4.3 Scope Selection [SCOPES]

When PSINIT is called to initialize PICTURE SYSTEM 2, all Picture Displays (Scopes 0-5) are selected for output. This selection will remain in effect until the SCOPES subroutine is called to deselect the Picture Displays to which output is not desired. Following is the SCOPES calling sequence specification of Section 6.1:

```
CALL SCOPES(IVALUE)
```

The parameter passed to this subroutine specifies the Picture Displays to be selected/deselected. IVALUE is interpreted as a 6-bit binary value where each bit that is set will select the corresponding scope and each bit that is not set will deselect the corresponding scope. Thus, the value 1 will select scope 0; 2, scope 1; 4, scope 2; 8, scope 3; 16, scope 4; and 32, scope 5. The values are additive so that 1+2+4+8+16+32 (=63) will select all scopes for display. Example 4.9-11 shows the use of the SCOPES subroutine to select scope 1, 3 and 5.

```
C
C INITIALIZE PICTURE SYSTEM 2
C
         CALL PSINIT(2,0)
           .
           .
           .
C
C NOW SELECT ONLY SCOPES 1, 3 AND 5
C
         CALL SCOPES(42)
           .
           .
           .
```

Example 4.9-11

If the user's particular PICTURE SYSTEM configuration has fewer than six 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 components of the Picture Generator.

4.10 LINEAR DISPLAY LISTS

The PICTURE SYSTEM typically functions in a mode of operation where a Graphics Subroutine is called by a user program to perform a given operation (i.e., draw 3D data, concatenate a rotation matrix to the current transformation matrix, etc.) and returns to the calling program as soon as the operation has been initiated. All of the subroutines discussed previously in this chapter function in this manner. However, for each subroutine call there is a certain amount of software overhead implicit in checking the parameters passed by the user and setting up and initiating the data transfer from the Picture Controller to the PICTURE SYSTEM. For objects which do not change dynamically, and are basically static in nature, this software overhead may be eliminated by building the objects as a linear display list. A linear display list is a collection of PICTURE SYSTEM commands* and associated data stored in an array that may be output in a single DMA transfer. A linear display list may be created a command or data element at a time, but usually is generated using the Graphics Subroutines and the linear display list support subroutines MAKEOB and STOPOB.

---

*See Reference 1, Section 2.3.2 for a detailed description of the PICTURE SYSTEM 2 commands available.

4.10.1 Linear Display List Generation [MAKEOB,STOPOB]

The subroutine MAKEOB is called to set the PICTURE SYSTEM Graphics Software into a mode where all subsequent commands and data are buffered in a user-supplied array rather than being output directly to the Picture Processor. The collection of the data placed into the user-supplied buffer is transparent to the user, but occurs for all calls to PICTURE SYSTEM output subroutines (i.e., VWPORT, WINDOW, MOVETO, DRAW3D, etc.) until the STOPOB subroutine is called to terminate the generation of the linear display list. The following is the MAKEOB calling sequence specification of Section 6.1:

```
[EXTERNAL FULSUB]
CALL MAKEOB(IARRAY,MAX,LEN[,FULSUB])
```

The parameters passed to MAKEOB are detailed below.

IARRAY is a user-supplied array (MAX words in length) into which the generated linear display file is to be buffered.

MAX is an integer which specifies the maximum length in words of the user-supplied buffer, IARRAY.

LEN is an integer variable where the number of buffer words thus far actually used will be maintained. After STOPOB is called, LEN will contain the actual number of words of the buffer (IARRAY) that have been used.

FULSUB is a subroutine which, if specified, will automatically be called when IARRAY becomes full. If supplied, the calling sequence will be:

```
CALL FULSUB(IARRAY,LEN)
```

When the user FULSUB subroutine is called by the linear display list generation software, it expects the buffer to be empty upon return. The LEN variable is reset to 1 when MAKEOB is called and after each call to a FULSUB routine.

The STOPOB subroutine is called to terminate the generation of a linear display list. The following is the STOPOB calling sequence specification of Section 6.1:

```
CALL STOPOB
```

When STOPOB is called, the linear display list is completed by appending a Picture Processor HALT command and the total number of PUSHes and POPs that are contained within the

list. These values are examined when the linear display list is actually output to insure that there is sufficient matrix stack area in the Picture Processor to perform the functions. The user subroutine, FULSUB, if one was supplied, is then called. Examples 4.10-1 and 4.10-2 show the generation of linear display lists.

```
      INTEGER IARRAY(50)
C
C   INITIALIZE PICTURE SYSTEM 2
C
      CALL PSINIT(2,0)
C
C   GENERATE A LINEAR DISPLAY LIST
C   TO DRAW A SQUARE
C
      CALL MAKEOB(IARRAY,50,LEN)
      CALL MOVETO(0,0)
      CALL LINETO(10000,0)
      CALL LINETO(10000,10000
      CALL LINETO(0,10000)
      CALL LINETO(0,0)
      CALL STOPOB
C
C   CONTINUE
C
        .
        .
        .
```

Example 4.10-1

```
      INTEGER IARRAY(100)
C
C   INITIALIZE PICTURE SYSTEM 2
C
      CALL PSINIT(2,0)
C
C   GENERATE A LINEAR DISPLAY LIST
C   WHICH DRAWS A SQUARE WHICH IS ROTATED
C   90 DEGREES ABOUT ITS X AXIS WITHIN A VIEWPORT
C   WHICH IS THE UPPER RIGHT HAND QUADRANT
C
      CALL MAKEOB(IARRAY,100,LEN)
      CALL VWPORT(0,2047,0,2047,255,0)
      CALL PUSH
      CALL ROT(16384,1)
      CALL MOVETO(0,0)
      CALL LINETO(10000,0)
      CALL LINETO(10000,10000)
      CALL LINETO(0,10000)
      CALL LINETO(0,0)
      CALL POP
      CALL STOPOB
C
C   CONTINUE
C
        .
        .
        .
```

Example 4.10-2

4.10.2 Drawing Linear Display Lists [DRAWOB]

A linear display list is output by calling the DRAWOB subroutine. The following is the DRAWOB calling sequence specification of Section 6.1:

```
CALL DRAWOB(IARRAY)
```

IARRAY is the user-supplied array in which the linear display list previously generated by the MAKEOB and STOPOB routines is located.

When DRAWOB is called, it initiates the transfer of the data from the Picture Controller to PICTURE SYSTEM 2 using the DMA channel. Thus when the DRAWOB subroutine is returned from, user computations may proceed while the linear display list is being processed by the Picture Processor. Example 4.10-3 shows the drawing of a linear display list.

```
            INTEGER IARRAY(700)
C
C   INITIALIZE PICTURE SYSTEM 2
C
            CALL PSINIT(2,0)
C
C   GENERATE A CIRCLE IN A  LINEAR DISPLAY LIST
C
            CALL MAKEOB(IARRAY,700,LEN)
            CALL MOVETO(32767,0)
            DO 10 I=4,360,4
             R=I
            IX=32767*COS(R/(3.1416*2))
            IY=32767*SIN(R/(3.1416*2))
10          CALL LINETO(IX,IY)
            CALL STOPOB
C
C   DISPLAY LOOP
C
            I=0
100         CALL PUSH
C
C   ROTATE ABOUT X AND Y AXES AND THEN DRAW THE OBJECT
C
            CALL ROT(I,1)
            CALL ROT(I,2)
            CALL DRAWOB(IARRAY)
            CALL POP
            CALL NUFRAM
            I=I+182
            GO TO 100
            END
```

Example 4.10-3

During the generation of a linear display list, it is also possible to include a previously generated linear display list as part of the object being created. Thus, calls to the DRAWOB subroutine are valid when MAKEOB is in process. This will cause the data normally output by the DRAWOB routine to be accumulated into the linear display list currently being generated. Example 4.10-4 illustrates this capability.

Linear display lists may contain Picture Generator status commands such as are generated by calls to the SCOPES, TXTURE, CHARSZ, BLINK and COLOR subroutines, as well as any drawing subroutines which produce dots. The Graphics

Software Package does not keep track of these status changes embedded in the linear display list. Thus, it is the user's responsibility to return the Picture Generator status to the desired status before calling STOPOB. This is done by the appropriate calls to the above subroutines. To reset from dot mode, simply issue a call such as "CALL MOVETO(0,0)".

```
            .
            .
            .
C
C   CREATE ANOTHER LINEAR DISPLAY LIST
C
          CALL MAKEOB(IARRAY,1000,LEN)
            .
            .
            .
C
C   INCLUDE OBJECT 1
C
          CALL DRAWOB(IOB1)
          CALL STOPOB
C
C   CONTINUE
C
            .
            .
            .
```

Example 4.10-4

Of course, linear display lists may be created and output as a data file to a mass storage device. A data file of this type may then be read by another program for display thereby eliminating the need to repeatedly recreate the linear display list each time the program is executed. If the file was generated as a series of smaller buffers by calls to the user's FULSUB routine, these must be recombined into a single array before being sent to DRAWOB. This is done by simply placing them end-to-end in an array.

4.10.3 Updating Linear Display Lists [GETROT,GETTRN,GETSCL,COSSIN]

While linear display lists are extremely efficient for static objects, they are similarly useful for dynamic objects and scenes in which only the transformations that are performed (but not the data displayed) are changing. Several subroutines, analogous to the ROT, TRAN and SCALE subroutines, are provided to facilitate the modification of the transformations embedded within a linear display list. These routines, GETROT, GETTRN and GETSCL, have the same input parameters as the ROT, TRAN and SCALE subroutines but, instead of concatenating the transformation created to the current Transformation Matrix, return the 4x4 matrix in a 16-word array supplied by the user. The calling sequence specifications of Section 6.1 for the subroutines are listed below.

```
CALL GETROT(IARRAY,IANGLE,IAXIS)
```

IARRAY is a 16-word array in which the 4x4 matrix rotation transformation is to be returned.

IANGLE and IAXIS are the same input parameters as passed to the ROT subroutine (see Section 4.6 for details).

```
CALL GETTRN(IARRAY,ITX,ITY,ITZ[,IW])
```

IARRAY is a 16-word array in which the 4x4 matrix translation transformation is to be returned.

ITX, ITY, ITZ and IW are the same input parameters as passed to the TRAN subroutine (see Section 4.7 for details).

```
CALL GETSCL(IARRAY,ISX,ISY,ISZ[,IW])
```

IARRAY is a 16-word array in which the 4x4 matrix scaling transformation is to be returned.

ISX, ISY, ISZ and IW are the same input parameters as passed to the SCALE subroutine (see Section 4.8 for details).

Also provided is a subroutine, COSSIN, which returns a cosine and sine for a given input angle. These values may then be used directly to update the individual terms of a rotation matrix*. The following is the COSSIN calling sequence specification of Section 6.1:

CALL COSSIN(IANGLE,ICOS,ISIN)

IANGLE is an integer which specifies the angle of rotation. This parameter is the same as would be passed to the ROT or GETROT subroutines to create a rotation matrix. 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.

ICOS is an integer variable in which the computed cosine (scaled for insertion in a rotation matrix) will be returned.

ISIN is an integer variable in which the computed sine (scaled for insertion in a rotation matrix) will be returned.

To use the GETROT, GETTRN, GETSCL and COSSIN subroutines, one must be able to determine where within a linear display list a transformation resides. This position may be programably determined by using the "LEN" variable passed as an argument in the parameter list to the MAKEOB subroutine. As described previously in Section 4.10.1, LEN is an integer variable where the number of words of the user-supplied buffer actually used will be maintained. Thus at any time during the generation of a linear display list (provided that FULSUB has not been called), this variable will contain a value which may be used as an index into the linear display list for the next PICTURE SYSTEM operation to be performed. For example, if during the generation of a linear display list the contents of the LEN variable were copied to another variable (LENA) before a call to the ROT subroutine was made, then the rotation matrix could be updated using the GETROT subroutine at a later time. This would be done by a call to the GETROT subroutine of the following form:

CALL GETROT(IARRAY(LENA+2),IANGLE,2)

In this call, LENA+2 is an index from the beginning of the linear display list (IARRAY(1) in this case) to the beginning of the 4x4 rotation matrix. Thus, GETROT will return a new rotation matrix in the location where the original matrix resided. Using GETTRN and GETSCL in a similar manner allows transformations within linear display lists to be updated easily without reconstructing the entire list.

---

*Chapter 4 of the PICTURE SYSTEM 2 Reference Manual details the rotation matrices generated for rotation about each of the x, y and z axes. The values returned by COSSIN may be used directly to update the individual elements of these matrices.

Example 4.10-5 shows the updating of a linear display list in this manner.

```
            .
            .
            .
C
C   INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
C
C   CREATE THE LINEAR DISPLAY LIST
C
          CALL MAKEOB(IARRAY(1),1000,LEN)
          CALL PUSH
          LENA=LEN
          CALL TRAN(ITX,ITY,ITZ)
          LENB=LEN
          CALL ROT(IANGLE,1)
          LENC=LEN
          CALL SCALE(ISV,ISV,ISV)
          CALL DRAW3D(IDATA1,200,2,2)
          CALL DRAW3D(IDATA2,100,2,0)
          CALL STOPOB
C
C   BEGIN DISPLAY LOOP BY UPDATING DYNAMIC VALUES
C
100       ITZ=ITZ+100
          IANGLE=IANGLE+182
          ISV=ISV+200
C
C   UPDATE THE LINEAR DISPLAY LIST AND THEN DRAW IT
C
          CALL GETTRN(IARRAY(LENA+2),ITX,ITY,ITZ)
          CALL GETROT(IARRAY(LENB+2),IANGLE,1)
          CALL GETSCL(IARRAY(LENC+2),ISV,ISV,ISV)
          CALL DRAWOB(IARRAY)
          CALL NUFRAM
          GO TO 100
          END
```

Example 4.10-5

4.11 DATA DISPLAY [NUFRAM,SETBUF]

The manner in which frames are displayed on PICTURE SYSTEM 2 is dependent upon the requirements of the user's application. Several modes of refresh buffer utilization are available, each with its own advantages. These are:

1. Display of data without refresh buffering.
2. The refresh buffer used in single-buffer mode.
3. The refresh buffer used in double-buffer mode.
4. The refresh buffer used in segmented-buffer mode.

The use of the system in each of these modes is detailed in the following sections.

4.11.1 Display Of Data Without Refresh Buffering

During initialization, the PICTURE SYSTEM 2 Graphics Software Package dynamically determines whether there is a Refresh Controller available and the amount of PICTURE Memory that is to be used for refresh buffering. If there is no Refresh Controller or there is no PICTURE Memory available, then the system is configured to channel the output of the Picture Processor directly to the Line Generator for display. When used in this manner, there is no refresh buffering of data and the refresh rate is the same as the update rate. The update rate should be sufficient to avoid flicker, a constraint not present when a refresh buffer is used. However, program development may proceed in this configuration without regard to the fact that there is no refresh buffer. Refresh buffer specific subroutines (e.g., NUFRAM, SETBUF) perform no function when called in this configuration other than ensuring synchronization of the update (and refresh) with the line frequency.

4.11.2 Display Of Data In Single-Buffer Mode

PICTURE SYSTEM 2 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 the last portion of data from the picture, when attempting to use double-buffer mode. During the output of data, if the capacity of the Refresh Buffer is exceeded, then all further output is inhibited until the NUFRAM subroutine is called. The user selects the single-buffer mode of the Refresh Buffer by calling the SETBUF subroutine as illustrated in Example 4.11-1.

```
C
C INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
C
C AND SET THE REFRESH BUFFER TO SINGLE-BUFFER MODE
C
          CALL SETBUF(1)
C
C   BEGIN THE DISPLAY LOOP
C
             .
             .
             .
```

Example 4.11-1

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 4.11-2 illustrates, in single-buffer mode, data displayed on the Picture Display are identical to the data 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) and a portion of the new data (new frame). This is of little importance where the data may not change drastically from frame to frame or where there are many refresh cycles between frame updates. The structure of a program which uses single-buffer mode is, nonetheless, the same as if the program were using double-buffered mode. In either case, the user draws* all of the data 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 4.11-2.) If no subsequent data are 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 to be viewed. This data will continue to be refreshed until an "erase" of the refresh buffer is initiated by the user. The equivalent of an "erase" is provided by calling the NUFRAM

---

*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.

subroutine twice in succession. Therefore, a user could write an ERASE subroutine similar to Example 4.11-3 which could then be called to "erase" the Picture Display. The user is free to utilize the single-buffered refresh buffer as previously described. Example 4.11-4 illustrates this with the use of the ERASE subroutine of Example 4.11-3 only between major frame changes.

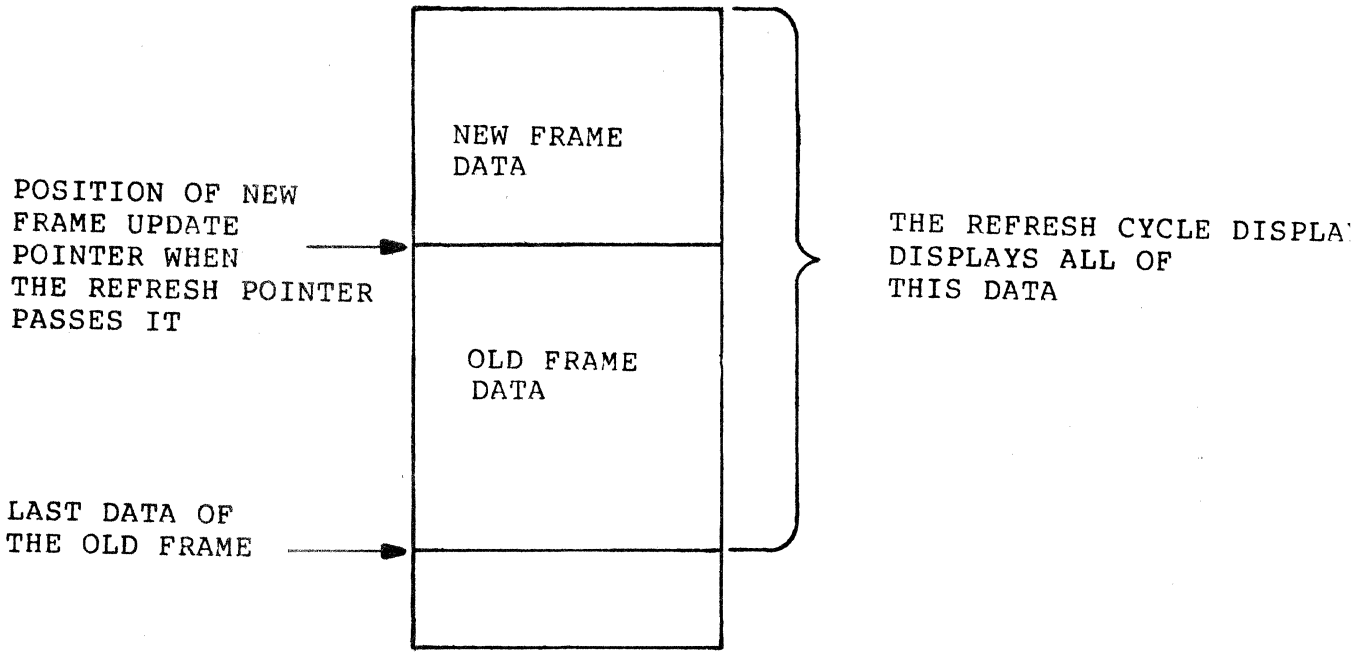


Figure 4.11-1

A Single-Buffered Refresh Buffer

```
C
C   INITIALIZE THE PICTURE SYSTEM
C
          CALL PSINIT(2,0)
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,
     1                4000,8000)
C
C   SAVE WINDOWING TRANSFORMATION AND BEGIN THE
C   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 4.11-2

```
      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 4.11-3

```
            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
C    EXIT
1000        CALL POP
            CALL ERASE
            RETURN
            END
```

Example 4.11-4

4.11.3 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 this state by PSINIT when called by a user program. In this mode, the user fills a buffer with data to be displayed, calls the NUFRAM subroutine to initiate its display and may then proceed to fill the opposite buffer with new frame data. This is illustrated by Figure 4.11-2. This method of frame display frees the user to create a new frame without worry of degradation of the picture. Example 4.11-5 shows the display loop structure of a typical application program which utilizes the double-buffer mode. If the user's application required the refresh buffer to be used in single-buffer mode for a given set of frames and subsequently returned to double-buffer mode, it would be done as shown in Example 4.11-6.

```
C INITIALIZE PICTURE SYSTEM 2
C
    CALL PSINIT(2,0)
C
C SET THE WINDOWING TRANSFORMATION
C
         CALL WINDOW(-4000,4000,-4000,4000,-4000,
       1              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
C DATA TWICE
C
         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 4.11-5

```
C ENTER THE DISPLAY SECTION WHICH MUST USE
C SINGLE BUFFER
C
          CALL SETBUF(1)
C
C BEGIN DISPLAY OF 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 4.11-6

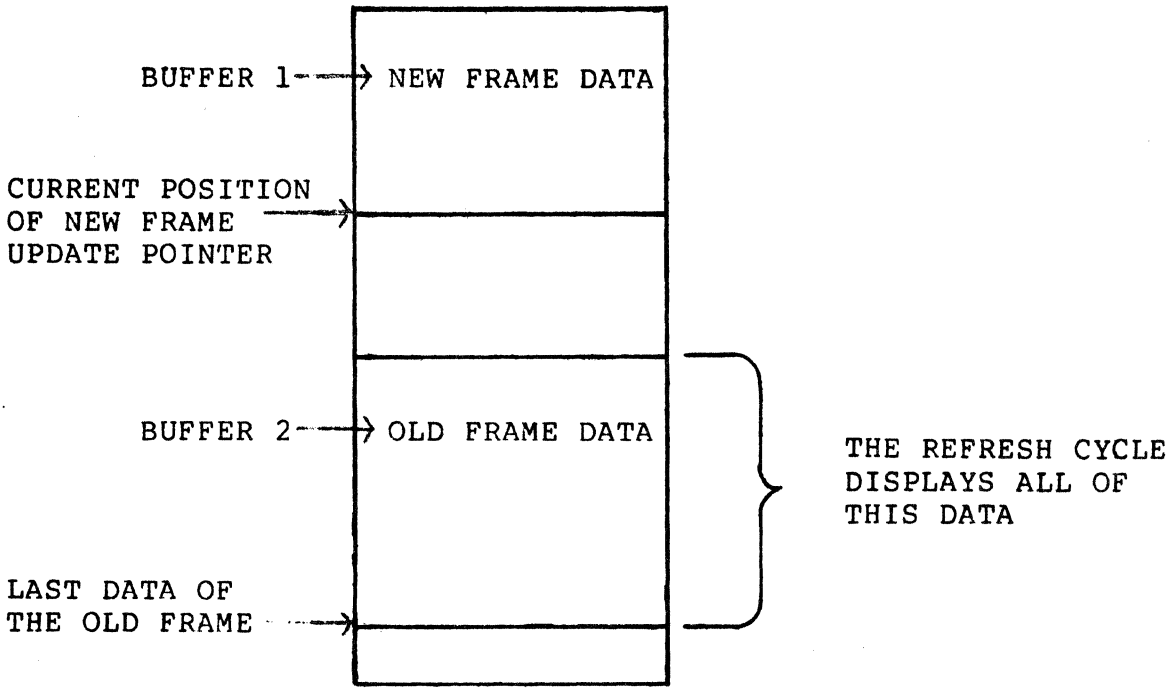


Figure 4.11-2

A Double-Buffered Refresh Buffer

4.11.4 Display Of Data In Segmented-Buffer Mode [CLEARS,OPENS,CLOSES,DELETS,BLANKS,SYNCS]

Use of the refresh buffer in segmented-buffer mode provides the most general use of the memory for the display and updating of data. When used in this manner, the user may create a display as many separate portions, or segments, each of which may be updated independently. This has a great inherent advantage over single- or double-buffered modes in that only the portion of a picture that has changed need be updated. In single- or double-buffered modes, the entire picture must be updated even if only a very small portion of it has changed.

As in single- or double-buffered mode, data output to the refresh buffer proceeds linearly from the beginning of the memory. As segments are created or updated, they are placed at the end of the last segment in the refresh buffer. As segments are deleted, or replaced by updated segments of the same name, the segments are compacted during the course of the refreshing of the buffer using special purpose segmentation hardware built into the Refresh Controller. This is illustrated by Figures 4.11-3a and b. This method of frame display frees the user to update only those portions of the picture that change providing an increased update rate and a corresponding decrease of load on the host Picture Controller.

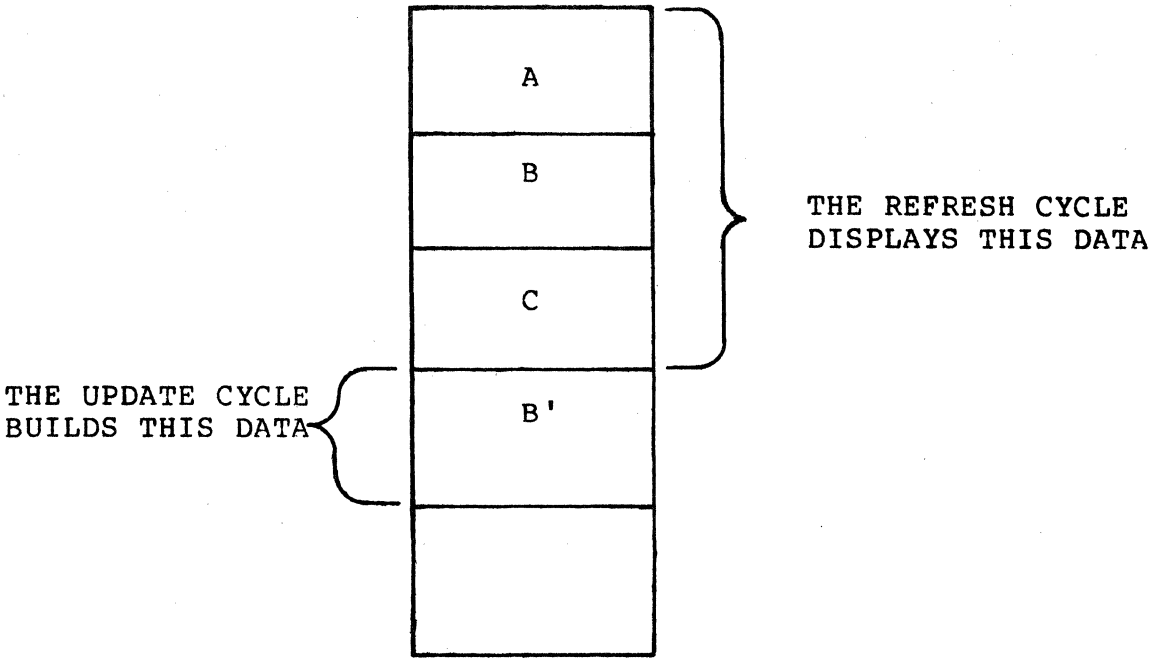


Figure 4.11-3a
Segmented Refresh Buffer showing
segments A, B, C and new segment B'
which is to replace current segment B.

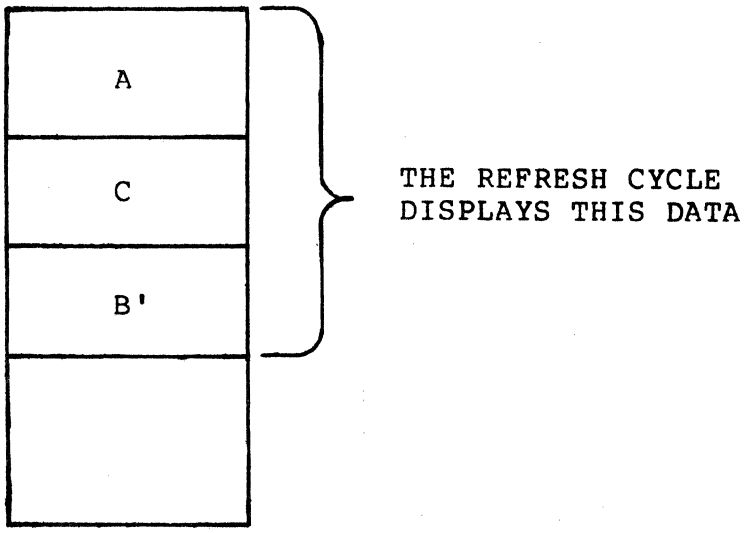


Figure 4.11-3b
Segmented Refresh Buffer showing
segment position after memory compaction.

Segmented-buffer mode is initiated by calling the CLEARS subroutine. Following is the CLEARS calling sequence specification of Section 6.1:

```
CALL CLEARS(IARRAY,ISIZE)
```

IARRAY is an array ISIZE words in length that is used to contain the data associated with each segment.

ISIZE is an integer which specifies the size, in words, of IARRAY. This value should be at least 3*(m+n)+6 in length where m is the maximum number of segments that are to be created and n is the maximum number of segments which may be active during a refresh cycle. ISIZE must be less than 4064 (4096-32), thus limiting the total number of segments which may exist and be updated to 1352.

When CLEARS is called it, in effect, "erases" the contents of the refresh buffer. Any image that was being displayed will cease to be displayed. Once CLEARS has been called, the user is free to create and update his picture as individual segments. Each segment created is identified by a non-zero integer number (or "name"), by which each subsequent reference to the segment (update, delete, etc.) is made. A segment is opened for creation (or update) by calling the OPENS subroutine. Following is the OPENS calling sequence specification of Section 6.1:

```
CALL OPENS(N)
```

The parameter N passed to this routine is the "name" by which this segment is to be referenced. Creation of the segment then proceeds just as if the user were in single- or double-buffer mode. The user is free to call any of the graphics subroutines to perform ROTations, TRANslations, DRAW3Ds, etc. However, all data output by the Picture Processor will become part of segment "N", and hence may be updated independently at a later time. Example 4.11-7 illustrates the opening of a segment.

```
              .
              .
              .
C    INITIALIZE PICTURE SYSTEM 2
C
           CALL PSINIT(2,0)
C
C    INITIALIZE SEGMENTATION
C
           CALL CLEARS(IARRAY,30)
C
C    CREATE SEGMENT 1
C
           CALL OPENS(1)
C
C    DRAW 100 LINES
C
           DO 10 I=1,30001,300
           CALL MOVETO(0,I)
           CALL LINETO(30000,I)
10         CONTINUE
              .
              .
              .
```

Example 4.11-7

A segment which has been opened will receive all data output by the Picture Processor until the segment is implicitly closed by the opening of another segment or explicitly closed by a call to the CLOSES subroutine. The following is the CLOSES calling sequence specification of Section 6.1:

```
CALL CLOSES
```

The CLOSES subroutine is called to close the current segment which is open (if any) and to cause all of the segments which have been implicitly closed since the last call to CLOSES (or CLEARS) to be displayed during the next refresh cycle. This allows many segments to be updated ensuring that the visual effect of the sequence of updates occurs at the same time. Without this capability, it would be possible to apply the same transformation to a series of objects with some of the objects being displayed in the new position during one refresh cycle and the rest of the objects during a subsequent refresh cycle. This would make the objects appear to move out of synchronization with one another, a very undesirable result. Example 4.11-8 illustrates the implicit

and explicit closing of segments.

```
                .
                .
                .
C    INITIALIZE PICTURE SYSTEM 2
C
           CALL PSINIT(2,0)
C
C    INITIALIZE SEGMENTATION
C
           CALL CLEARS(IARRAY,18)
                .
                .
                .
C
C    OPEN SEGMENT 1
C
           CALL OPENS(1)
           CALL PUSH
           CALL ROT(IANGLE,1)
           CALL DRAW3D(IARRAY,25,2,2)
                .
                .
                .
C
C    OPEN SEGMENT 2 (IMPLICITLY CLOSING SEGMENT 1)
C
           CALL OPENS(2)
           CALL PUSH
           CALL TRAN(ITX,ITY,IZ)
           CALL DRAW3D(IARRAY,25,2,2)
                .
                .
                .
C
C    EXPLICITLY CLOSE SEGMENTS 1 AND 2
C
           CALL CLOSES
                .
                .
                .
```

Example 4.11-8

A segment which has been created may be deleted by (1) replacing it with another segment of the same name, or (2) calling the DELETS subroutine to delete it entirely from the refresh buffer. The following is the DELETS calling sequence specification of Section 6.1:

```
CALL DELETS(N)
```

N is an integer which specifies the "name" by which the segment was opened.

A segment which has been deleted frees the area of refresh memory that it formerly occupied for use by another segment. When a segment is deleted, the remaining segments in the refresh buffer are automatically compacted, thereby providing the maximum area possible for the creation of new segments. Often, however, it is desirable to cause a segment (or segments) to not be displayed but retained within the refresh buffer for display at a later time. This may be done by calling the BLANKS subroutine to cause a segment to be "blanked" (refreshed but not displayed) for all subsequent refresh cycles until the segment is either updated or BLANKS is called to "unblank" the segment. The following is the BLANKS calling sequence specification of Section 6.1:

```
CALL BLANKS(N,STATE)
```

N is an integer which specifies the "name" by which the segment was opened.

STATE is a logical value which specifies whether the segment is to be "blanked" or "unblanked". STATE=.FALSE. for unblanked and .TRUE. for blanked.

All segments are created, by default, as unblanked segments. If is possible, however, to modify the default blank/unblank status by a call to the BLANKS subroutine with N=0. When BLANKS is called in this manner, all subsequently opened segments will open with the default blank/unblank status as defined by the STATE variable. Thus, "CALL BLANKS(0,.TRUE.)" would cause the default blank/unblank status to be set such that all subsequently opened segments would be blanked. Similiarly, "CALL BLANKS(0,.FALSE.)" would cause the default to return to unblanked.

If BLANKS is called to blank a segment which is currently open, then the newly created segment will be blanked, otherwise, the segment of the name specified which is currently being displayed will be blanked/unblanked. Example 4.11-9 shows how BLANKS may be used to blank and unblank segments. As with the OPENS subroutine, the DELETS and BLANKS subroutines must be followed by a call to CLOSES to have an effect

upon the segments currently being refreshed. This ensures that the user is able to perform all of the functions (i.e., OPENS, DELETS, BLANKS) upon the current segments during the same refresh cycle.

```
C    INITIALIZE PICTURE SYSTEM 2
C
           CALL PSINIT(2,0)
C
C    INITIALIZE SEGMENTATION
C
           CALL CLEARS(IARRAY,18)
C
C    CREATE SEGMENT 1 (INITIALLY BLANKED)
C
           CALL OPENS(1)
           CALL BLANKS(1,.TRUE.)
             .
             .
             .
           CALL CLOSES
C
C    CREATE SEGMENT 2
C
           CALL OPENS(2)
             .
             .
             .
           CALL CLOSES
C
C    BEGIN DISPLAY LOOP
C
100        CONTINUE
             .
             .
             .
C
C    NOW UNBLANK SEGMENT 1 AND BLANK SEGMENT 2
C
           CALL BLANKS(1,.FALSE.)
           CALL BLANKS(2,.TRUE.)
           CALL CLOSES
             .
             .
             .
```

Example 4.11-9

When using segmentation, is is possible to begin the creation of a new segment before the compaction of deleted or replaced segments has completed. If this occurs repeatedly, each subsequent segment will tend to "creep" towards the end of the refresh buffer. Eventually, a segment will meet the limit of the refresh buffer. When this occurs, the condition is detected by the segmentation software and the incomplete segment will be compacted at the end of the next refresh cycle and the segment output then allowed to continue to completion. Thus, the user is freed from concern about the physical configuration and allocation of the refresh buffer. However, in displaying objects in which smooth movement is essential, this asynchronous, "occasional", segment compaction operation can cause a marked difference in update rate. Such a difference may be apparent to the user as a jerkiness in the otherwise smooth motion of the object. To avoid this phenomenon, the user may elect to synchronize the creation of new segments with the completion of segment compaction. This may by done by a call to the SYNCS subroutine. The following is the SYNCS calling sequence specification of Section 6.1:

```
CALL SYNCS
```

When this subroutine is called, it will not return to the calling program until all pending refresh buffer compaction is complete. Example 4.11-10 illustrates the use of this subroutine.

```
C   INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
          CALL CLEARS(IARRAY,18)
C
C   CREATE STATIC SEGMENTS 1 AND 2
C
          CALL OPENS(1)
           .
           .
           .
          CALL OPENS(2)
           .
           .
           .
          CALL CLOSES
C
C   BEGIN DISPLAY LOOP ENSURING SYNCHRONIZATION WITH RB
C
100       CALL SYNCS
C
C   UPDATE DYNAMIC SEGMENT 3
C
          CALL OPENS(3)
           .
           .
           .
          CALL CLOSES
          GO TO 100
          END
```

Example 4.11-10

In many applications, the use of segmentation is to create one or more segments which are relatively static (i.e., do not change often, if at all), and then continually update a constantly changing dynamic segment. An example of this might be a menu output as a static segment with the simulation of a real-time event as the dynamic segment. The optional manner in which such program might be written is to create the static segments and then divide the remaining refresh buffer in half so that the dynamic segment might be double buffered as an unnamed segment. The PICTURE SYSTEM 2 Graphics Software provides this flexibility. Once a segment has been explicitly closed, any further call to PICTURE SYSTEM graphics output routines (i.e., DRAW2D, MOVETO, TXTURE, etc.) without a call to the OPENS subroutine causes the remaining refresh buffer to be divided in half and set to a default double-buffer mode before data output proceeds. The

user is then free to create his picture as if he were using normal double-buffer mode. All data will be output to the "new" buffer until the NUFRAM subroutine is called to cause the buffers to be swapped. This mode of operation allows frame update to proceed at the maximum possible rate without the need to synchronize with refresh buffer compaction. Example 4.11-11 illustrates the use of the refresh buffer in this manner.

```
            .
            .
            .
C   INITIALIZE PICTURE SYSTEM 2
C
            CALL PSINIT(2,0)
C
C   INITIALIZE SEGMENTATION
C
            CALL CLEARS(IARRAY,9)
C
C   CREATE THE STATIC SEGMENT
C
            CALL OPENS(1)
C
C   NOW OUTPUT THE MENU
C
            CALL MOVETO(-20000,-20000)
            CALL TEXT(5,'POINT')
             .
             .
             .
            CALL CLOSES
C
C   BEGIN DISPLAY LOOP USING UNNAMED, DYNAMIC SEGMENT
C
100         CALL PUSH
            CALL ROT(IANGLE,2)
            CALL DRAW3D(IDATA,100,2,2)
             .
             .
             .
            CALL POP
            CALL NUFRAM
            GO TO 100
            END
```

Example 4.11-11

The unnamed dynamic segment may also be used in a single-buffer mode by calling the SETBUF subroutine after the last static segment has been explicitly closed. Example 4.11-12 shows this capability.

```
            .
            .
            .
C   INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
C
C   INITIALIZE SEGMENTATION
C
         CALL CLEARS(IARRAY,9)
C
C   CREATE THE STATIC SEGMENT
C
          CALL OPENS(1)
          CALL MOVETO(-20000,-20000)
          CALL TEXT(10,'POINT HERE')
            .
            .
            .
          CALL CLOSES
C
C   SET FOR SINGLE BUFFER MODE
C
          CALL SETBUF(1)
C
C   BEGIN DISPLAY LOOP FOR UNNAMED, SINGLE-BUFFERED SEGMENT
C
100       CALL PUSH
          CALL ROT(IANGLE,2)
          CALL DRAW3D(IDATA,100,2,2)
            .
            .
            .
          CALL POP
          CALL NUFRAM
          GO TO 100
          END
```

Example 4.11-12

After data has been output to an unnamed single- or double-buffered segment, further calls to the segmentation routines will cause the unnamed segment to be deleted, returning the memory to the normal segmented buffer mode. In many applications, this can be most useful. Such a case is when a new static segment is created for each basic menu area. After an explicit close, an unnamed segment may again be created. Due to the nature of this operation, the unnamed segment may flicker if this process is done for each new frame to be displayed.

For all of the previously discussed refresh buffer modes, if more refresh data are output than there is physically room for in the allocated refresh buffer, the PICTURE SYSTEM Graphics Software will automatically inhibit the output of more data to the refresh buffer until more room is made available. Frame creation will proceed, but all data will be "clipped" from the scene. Thus the user may proceed with data output which has no visual effect upon the image displayed. This will continue in this manner until the user calls NUFRAM to "swap" buffers, CLEARS to reinitialize the entire refresh buffer or DELETS to delete a segment from the refresh buffer.

4.12 DATA WRITE-BACK TO MEMORY [WBTMEM,STOPWB]

In some applications, it is desirable to write data back to Picture Controller memory rather than output it to PICTURE Memory for display. The PICTURE SYSTEM Graphics Software Package provides this capability in the write-back to memory subroutines, WBTMEM and STOPWB. These subroutines are used to obtain transformed coordinate data in full 16-bit precision. This data may be used in applications such as high resolution plotting or where further processing of the data is required after transformation (e.g., hidden line removal, etc.).

The subroutine WBTMEM is called to set the PICTURE SYSTEM Graphics Software into a mode where all subsequent data output by the Picture Processor are transferred back to Picture Controller memory and stored in a user supplied buffer. This transfer is transparent to the user but occurs for all calls to PICTURE SYSTEM output subroutines (i.e., MOVETO, DRAW3D, TXTURES, etc.) until the STOPWB or NUFRAM subroutine is called to terminate the write-back of data.

The following is the WBTMEM calling sequence specification of Section 6.1:

```
CALL WBTMEM(IARRAY,MAX,LEN,ITYPE[,FULSUB])
```

IARRAY is a user-supplied write-back buffer (MAX words in length) in which data written back to memory are stored.

MAX is an integer which specifies the maximum size of IARRAY, in words.

LEN is an integer variable where the number of buffer words thus far actually used will be maintained. After STOPWB (or NUFRAM) is called, LEN will contain the actual number of words of the buffer (IARRAY) that have been used.

ITYPE is an integer which specifies the type of data to be written back. Since the processing of data by the Picture Processor is entirely digital, it is possible to obtain the data at various stages in the processing. The stages of the processing are, transformation, clipping and viewpoint mapping. ITYPE specifies at which of these stages of processing that data are to be written back.

ITYPE = 1 for data transformed only.
ITYPE = 2 for data transformed and clipped.
ITYPE = 3 for data transformed, clipped and viewport mapped.

For transformed only data, x, y, z and w coordinates are written back to memory for each two-, three- or four-dimensional coordinate point output to the Picture Processor. No data other than coordinate data are written back (i.e., status or characters). This allows the Picture Processor to be utilized as a general purpose 4x4 matrix multiplier, however, the x, y, z and w coordinate data are normalized*.

For data which has been clipped (ITYPE = 2 or 3), a command code is also written back to indicate what the coordinate data represents (i.e., MOVETO, DRAWTO, etc.). Status and characters are also written back on these modes. Section 6.1 details the specific data formats for each of the types of data that may be written back.

FULSUB is a subroutine which, if specified, will automatically be called when IARRAY becomes full. If supplied, the calling sequence will be:

```
CALL FULSUB(IARRAY,LEN)
```

When the user FULSUB subroutine is called by the write-back to memory software, it expects the buffer to be empty upon return. The LEN variable is zeroed when WBTMEM is called and should also be zeroed by the user FULSUB routine.

Write-back to memory of all data output by the Picture Processor will proceed until either the STOPWB or NUFRAM subroutine is called. At that time, the user FULSUB routine will be called (if specified) and the system will be returned to the state where all subsequent data output by the Picture Processor will be for display. Example 4.12-1 shows the use of the WBTMEM and STOPWB subroutines.

---

*It is possible to set the Picture Processor to a mode where the transformed coordinate data are not normalized, but this requires that 24 bits be written back for each x, y, z and w value to ensure that the proper sign of each coordinate is obtained. See Reference 1, Section 2.3.3 for further details.

```
            .
            .
            .
C
C    NOW WRITE-BACK TO MEMORY THE COORDINATE DATA
C
          CALL WBTMEM(IARRAY,2000,LEN,1)
          CALL DRAW3D(IDATA,500,2,2)
          CALL STOPWB
C
C    NOW REPROCESS THE DATA FOR THIS APPLICATION
C
            .
            .
            .
            .
```

Example 4.12-1

A particular use of the write-back to memory software is to obtain absolute coordinate data after a series of transformations have been applied to the orginal data. This allows several basic objects to be manipulated to form a new object. An example of this may be a house where each wall of the house is merely and instance of the same "master". One could continually apply the transformations necessary to place each of the walls, but alternately, the entire house could be written back to memory to form a new "transformation free" object. In applications where one is creating a library of objects, this can free the objects from carrying the transformations required to position each of the subobjects along with the library. Example 4.12-2 illustrates the use of the write-back to memory software in this manner.

```
            .
            .
            .
C
C   NOW PERFORM THE INSTANCING TRANSFORMATIONS
C   AND WRITE THE DATA BACK TO MEMORY
C
          CALL WBTMEM(IARRAY,2000,LEN,2)
C
          CALL INST(-10000,-9000,-10000,10000)
          CALL WALL
C
          CALL INST(9000,10000,-10000,10000)
          CALL WALL
C
          CALL INST(-10000,10000,-10000,-9000)
          CALL WALL
C
          CALL INST(-10000,10000,9000,10000)
          CALL WALL
C
C   NOW STOP THE WRITE-BACK AND STORE THE DATA
C
          CALL STOPWB
            .
            .
            .
```

Example 4.12-2

4.13 USER INTERACTION

The user may interact with PICTURE SYSTEM 2 in a variety of ways. It may be done using any of the peripherals that are standard with the Picture Controller (e.g., console terminal, etc.) or any of the graphic input devices available from Evans & Sutherland. The standard graphic input devices available for PICTURE SYSTEM 2 include:

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

The subroutines provided by the PICTURE SYSTEM Graphics Software Package to support these devices are detailed in the following sections.

4.13.1 Tablet and Cursor Use [TABLET,CURSOR,ISPCHD]

The Data Tablet serves as the standard, general-purpose graphic input device for PICTURE SYSTEM 2, performing those interactive functions usually reserved for light pens, joysticks, etc. This section discusses in detail how the Tablet may be used to perform pointing, positoning, and other miscellaneous functions required for flexible interactive data input.

Data points are input from the Tablet within a user application program by calling the TABLET subroutine. Following is the TABLET calling sequence specification of Section 6.1:

```
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 and y coordinates are returned in the IX and IY parameters as scaled* 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 4.13-1.

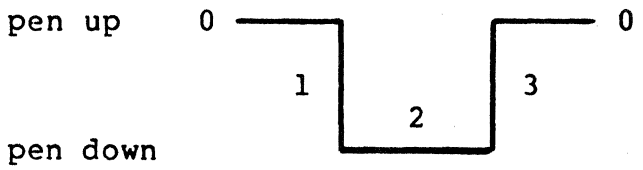
---

*The x and y cordinate values are scaled from the actual tablet coordinate range (0-2040 octal) to the approximate data space range ±32700 (±77700 octal).

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, allowing the pen information to be determined by the user. The returned status is information being read directly from the tablet. The returned status information is shown in Figure 4.13-2.

If the IX, IY and IPEN parameters are not passed to the TABLET subroutine, then the default IX, IY, and IPEN variables in the named COMMON block "PSCOM" are updated with the new pen data. See Section 6.1, subroutine PSINIT, for specific details of this COMMON block.

As Figure 4.13-2 illustrates, the user may determine when the pen is down (i.e., pressed against the surface of the tablet) by testing bit 4 of the pen status word. Since bit testing capabilities are not provided directly by FORTRAN, the integer function subroutine ISPCHD is available to the FORTRAN programmer to determine the pen status. This subroutine allows the pen transitions to be determined. The pen often transitions from a tracking position (up) to down and back up during pointing and picking operation. This subroutine allows the pen status to be considered as a state transition which can be graphed from call to call as:



The routine returns: 0 if the pen is up and was also up the last time the routine was called; 1 if the pen is down but was up last time the routine was called; 2 if the pen is down and was down last time the routine was called; 3 if the pen is up but was down last time the routine was called. This allows leading and trailing "edge" pen transitions to be tested for. This is illustrated by Example 4.13-1.

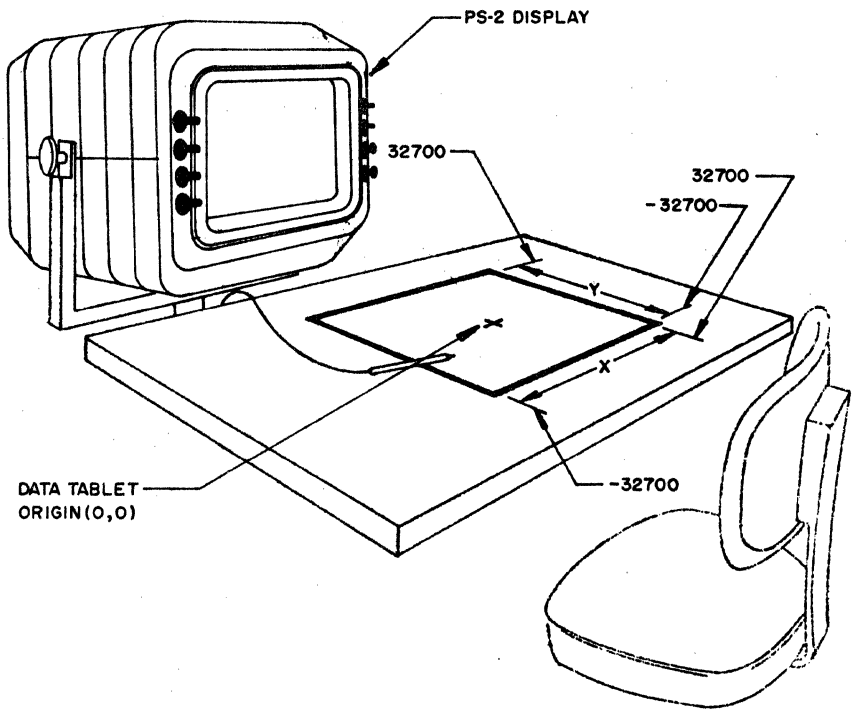


Figure 4.13-1
The Two-dimensional Coordinate System of the Tablet

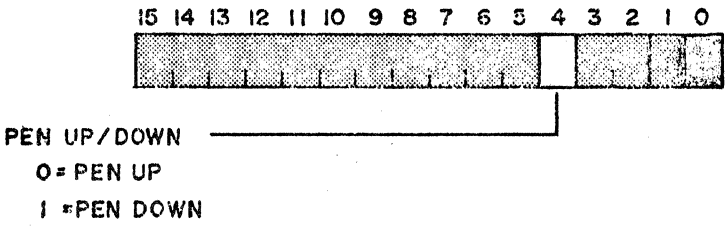


Figure 4.13-2

The Pen Status as Returned by the TABLET Subroutine

```
                .
                .
                .
C
C    READ THE TABLET VALUES AND PEN STATUS
C
           CALL TABLET(0,IX,IY,IPEN)
C
C    IF THE PEN JUST WENT DOWN GO TO 100
C
           IF(ISPCHD(IPEN).EQ.1) GO TO 100
C
C    ELSE JUST CONTINUE
C
                .
                .
                .
```

Example 4.13-1

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 and y coordinate and status are then updated automatically upon each refresh interrupt. In this way, the user constantly has available the most recent tablet values without explicitly calling the TABLET subroutine. Example 4.13-2 shows how the tablet may be used in automatic mode.

```
            INTEGER IX,IY,IPEN
              .
              .
              .
C
C   INITIALIZE THE PICTURE SYSTEM
C
            CALL PSINIT(2,0)
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(ISPCHD(IPEN).EQ.0 .OR. ISPDCHD(IPEN).EQ.3)
      1     GO TO 200
C
C   THE PEN WAS DOWN...
C
              .
              .
              .
```

Example 4.13-2

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, the tablet 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 will always be in the most recent position (x,y) where the pen was set down. Example 4.13-3 illustrates the clearing of the IPEN parameter after the pen position has been determined.

```
              .
              .
              .
C
C    IF THE PEN HAS NOT JUST BEEN LIFTED, GO TO 200
C
100         IF(ISPCHD(IPEN).NE.3)  GO TO 200
C
C    THE PEN WAS 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 4.13-3

It is often convenient to provide a visual feedback 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. Following is the CURSOR calling sequence specification of Section 6.1:

```
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 specified x,y coordinate. The pen status (IPEN) is an optional argument which brightens the cursor when the pen is down. If the pen is not in proximity of the tablet, the cursor will not be displayed. If this argument is not specified, the cursor will always be displayed at maximum intensity. As with the TABLET subroutine, if the IX, IY

and IPEN parameters are not specified in the call to the CURSOR subroutine, then the values in the default IX, IY and IPEN variables in the named COMMON block "PSCOM" will be used to position the cursor. See Section 6.1, subroutine PSINIT, for specific details of this COMMON block. Example 4.13-4 shows the use of the TABLET and CURSOR subroutines using the default variables.

```
C
C READ THE TABLET VALUES
C
        CALL TABLET(0)
C
C AND DISPLAY THE CURSOR
C
        CALL CURSOR(0)
          .
          .
          .
```

Example 4.13-4

As with the TABLET subroutine, the user may optionally choose to display a cursor in 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 displays 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 4.13-5 shows how the automatic mode of the TABLET and CURSOR subroutines may be specified.

```
C   DEFINE THE DEFAULT COMMON BLOCK
C
          COMMON/PSCOM/ICLOCK,IFRMCT,IX,IY,IPEN
C
C   INITIALIZE PICTURE SYSTEM 2
C
          CALL PSINIT(2,0)
C
C   "TURN ON" AUTOMATIC MODE FOR THE TABLET AND CURSOR
C
          CALL TABLET(1)
          CALL CURSOR(1)
C
C   NOW BEGIN THE DISPLAY LOOP WITHOUT WORRYING
C   ABOUT TABLET UPDATE AND CURSOR DISPLAY.
C
              .
              .
              .
```

Example 4.13-5

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 $\pm$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 4.13-6.

```
            INTEGER ZERO
            COMMON /PSCOM/ICLOCK,IFRMCT,IX,IY,IPEN
            DATA ZERO/0/
C
C   INITIALIZE PICTURE SYSTEM 2
C
            CALL PSINIT(2,0)
              .
              .
              .
C
C   SET CURSOR ALWAYS AT Y=0
C
            CALL TABLET(0)
            CALL CURSOR(IX,ZERO,0)
             .
             .
             .
```

Example 4.13-6

The cursor is defined in a window running from -32768 to +32767 in x and y, which coincides approximately with the range of tablet values.

When used in manual mode, the cursor will always be displayed within a viewport which is specified by the default variables "IVL,IVR,IVB,IVT" in the PICTURE SYSTEM COMMON block, PSCOM. In Example 4.13-6, this means that the cursor will always be displayed in a viewport which is the entire screen. However, as Example 4.13-7 shows, a cursor can be displayed within a dynamically-changing viewport merely by modifying those variables which define the viewport. This feature proves useful in menu and data pointing functions. Note that when used in automatic CURSOR mode, the cursor is always displayed within a viewport which is the entire screen.

```
      COMMON/PSCOM/ICLOCK,IFMCNT,IX,IY,IPEN
     1  IVL,IVR,IVB,IVT,IHI,IYI
C
C   INITIALIZE THE PICTURE SYSTEM
C
      CALL PSINIT(2,0)
C
C   BEGIN THE DISPLAY LOOP...
C
        .
        .
        .
C
C   CALL TABLET (DEFAULT IX,IY,IPEN)
C
      CALL TABLET(0)
      IPOINT=0
        .
        .
        .
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
C
210   CALL CURSOR(0)
        .
        .
        .
```

Example 4.13-7

4.13.1.1 Pointing [HITWIN,HITEST]

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 to a particular data element. However, the implementations of the two pointing functions are typically different. The following describes the use of the Tablet in performing these two pointing functions.

A menu item is a symbol, usually text, which when selected by the user issues a command to the user's program. Figure 4.13-3 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 is necessary only if the pen is down. Example 4.13-8 illustrates how this may be done.

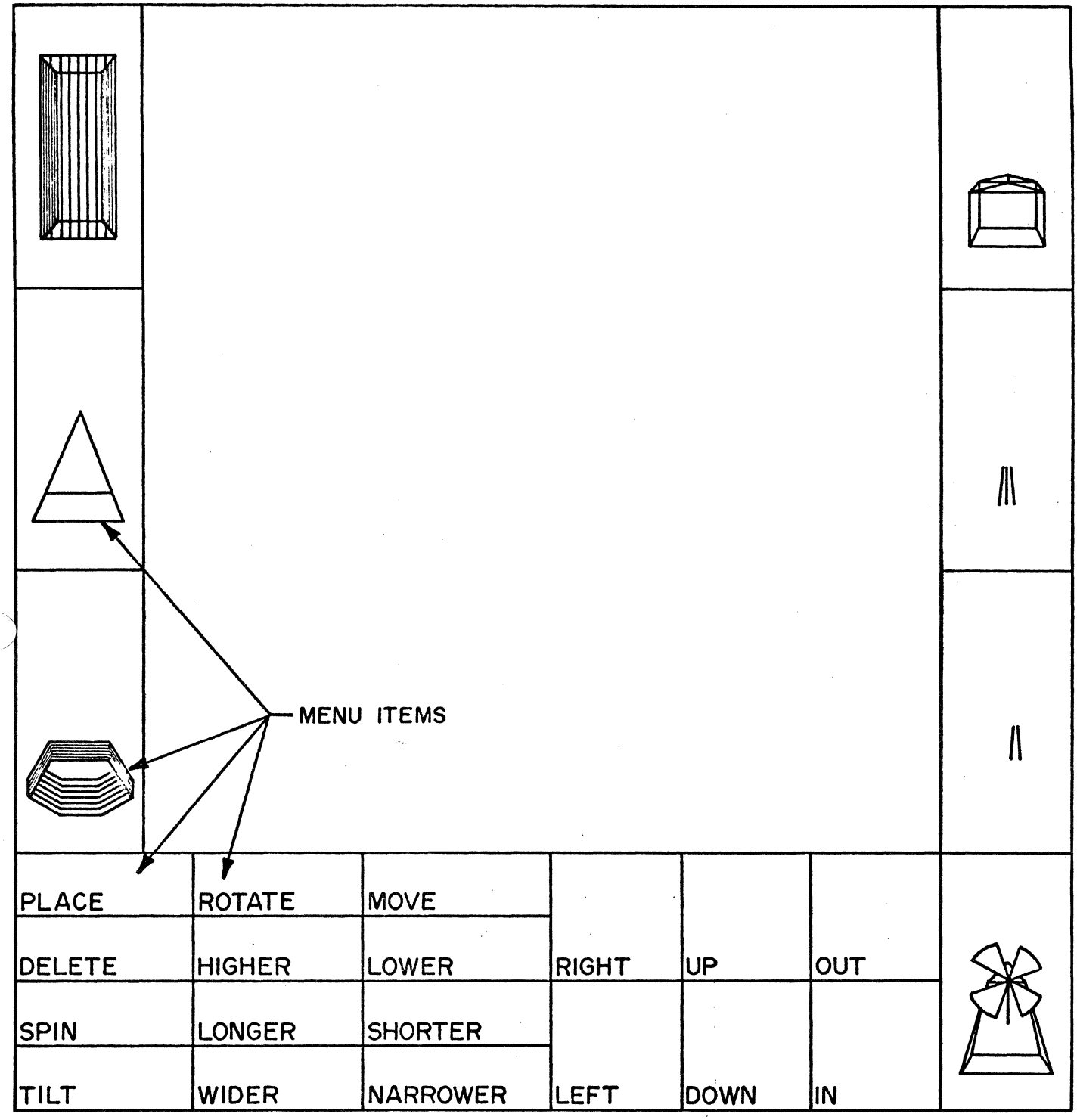


Figure 4.13-3

A Menu which Includes both Text and Other Symbols

```
              .
              .
              .
C
C    IF THE PEN WAS DOWN, ENTER THE MENU SELECTION CODE,
C    OTHERWISE, BRANCH TO 300
C
           IF(ISPCHD(IPEN).NE.3)  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
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 4.13-8

Note that the menu may be only a paper overlay which is placed on the tablet. A menu may also be displayed on the screen corresponding to the menu areas on the tablet, enabling the user to point with the cursor to the menu area on the screen when selecting 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 4.13-4.

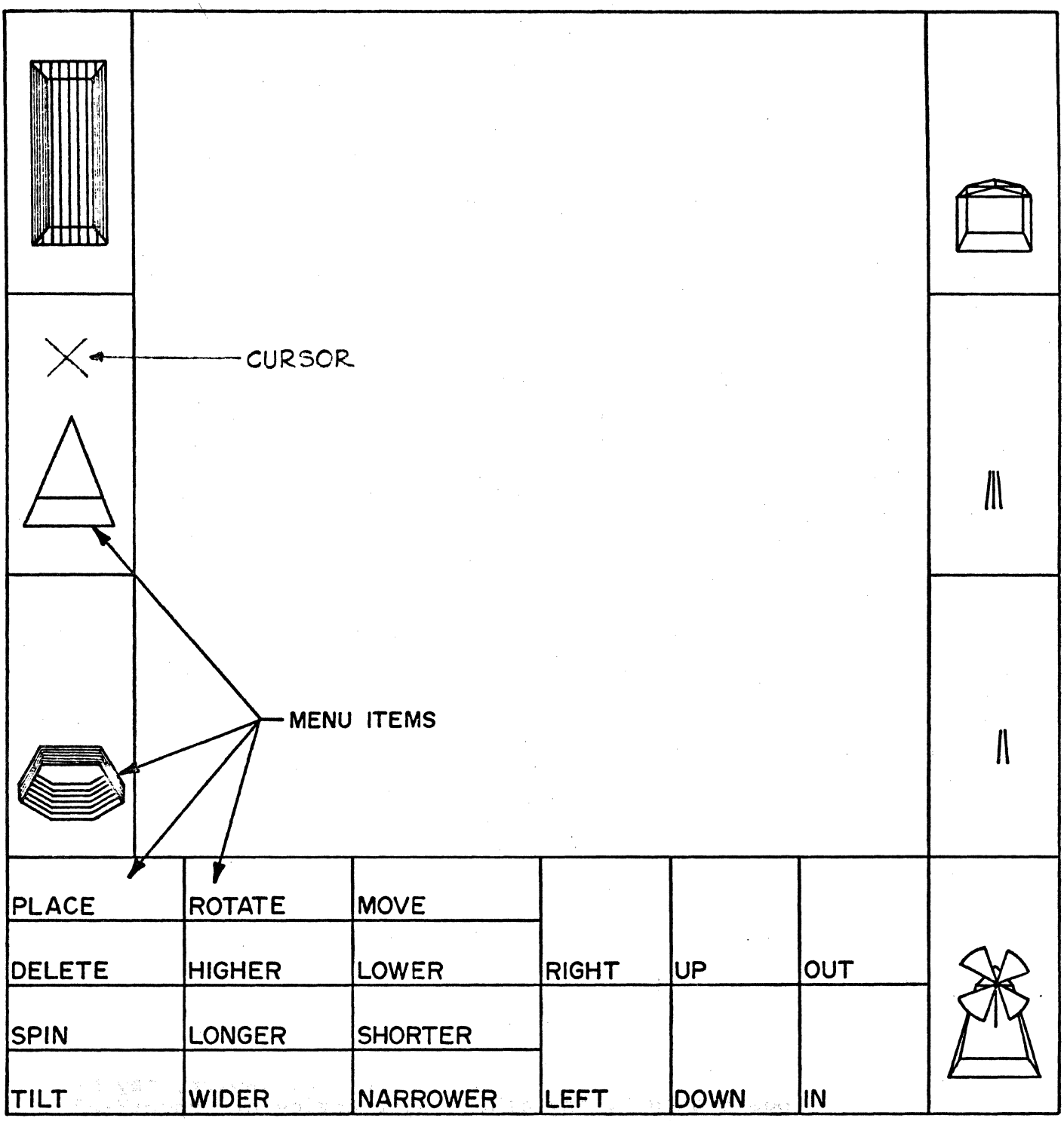


Figure 4.13-4

A Displayed Menu Illustrating Pointing at a Menu Item with the Cursor and also Data which has been Clipped and Mapped to the Viewport Boundaries

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, that data element. Such functions might include deletion of the data element, stress computation on the element in relation to its neighboring elements, indication of a bond about which a portion of a molecular model is to be rotated, or any other function which may be desired for a particular application. This pointing function, often erroneously considered to be strictly a light pen operation, is performed with PICTURE SYSTEM 2 by what is known as "hit testing". This function, facilitated by the Picture Processor's clipping process, allows a "hit window" to be defined through which all data in question may be processed whether any of the data was "hit"; i.e., whether any point (visible or not), or any part of any line was within the "hit window".

This process differs from the analogous function of the light pen if data pointed at with the light pen are part of a transformed display file. A transformed display file may bear little or no resemblance to the original data, especially if clipping has occurred. Thus, trace-back within the display file to determine which data element was pointed at can be quite complicated, unless care is taken to properly segment the transformed display file. In using hit testing, no trace-back to the data base is required since the hit testing is performed on the original data and not upon the transformed display file. The user also controls the level within the data base to which hit testing is to be performed. Hit testing also provides additional flexiblity in the following ways:

1. The hit testing feature, when coupled with the TABLET and CURSOR subroutines, allows the user to point at and identify data elements, with the added capability 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 positioned by any other input device or specified arithmetically, allowing data which is not displayed to be "hit".

3. Hit testing may be extended to three dimensions allowing data elements which lie within a given distance to a x,y,z position to be identified.

The "hit testing" capability is provided within the Graphics Software Package by the HITWIN and HITEST subroutines. Following are the HITWIN and HITEST calling sequence specifications of Section 6.1:

```
CALL HITWIN(IX,IY,SIZE[,IW])

CALL HITEST(IHIT,ISTAT)
```

The HITWIN subroutine is called to specify a "hit window" through which data may be processed to determine if 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, resulting 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 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 4.13-5. 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" 1 inch in width:

$$\frac{\text{ISIZE}}{32767} = \frac{1}{10} \text{ or } \text{ISIZE} = 3276$$

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, then a picture element which appears near the cursor will be hit. If the picture occupies a viewport other than the full screen, the cursor should also be confined to that viewport if hit testing is to be performed.

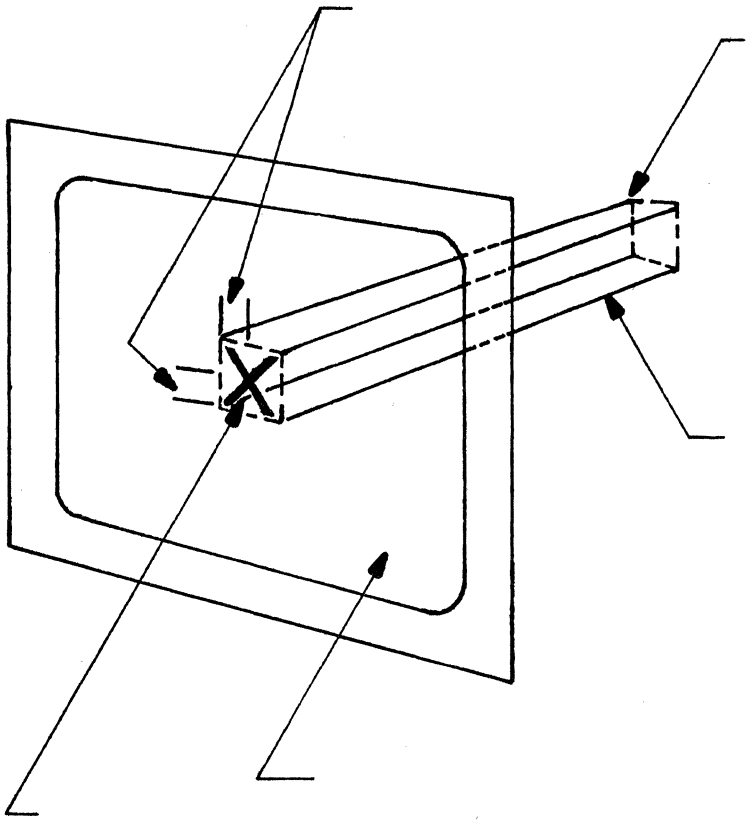


Figure 4.13-5

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

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 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 coordinates 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 testing" is to be performed (the data is not actually drawn, but is processed for hit testing purposes only).

3. Determine whether a "hit" has been made upon the data element or data set by calling the HITEST subroutine and testing 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 argument, ISTAT, set to a non-zero value to restore the transformation in effect when the HITWIN subroutine was called and to allow subse-

quent data drawn to be written into the Refresh Buffer.

The previous steps (1-4) specify the manner in which hit testing 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 testing". These transformations include WINDOWing, ROTation, TRANslation, SCAL(E)ing and INSTancing. This simplifies the "hit testing" process since it may be done within the logical flow of the program, while (or as if) a new frame is being created. Example 4.13-9 illustrates how "hit testing" may be used to determine if an object, in this case a "HOUSE", has been hit.

```
      COMMON/PSCOM/ICLOCK,IFMCNT,IX,IY,IPEN
C
C   INITIALIZE THE PICTURE SYSTEM AND TURN ON
C   AUTOMATIC TABLET AND CURSOR (DEFAULT VARIABLES)
C
      CALL PSINIT(2,0)
      CALL TABLET(1)
      CALL CURSOR(1)
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
200   CONTINUE
       .
       .
```

Example 4.13-9

4.13.1.2 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, in 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. Example 4.13-10 shows how rubber-banding of lines may be done.

```
      INTEGER IARRAY(2,500)
C
      DATA MAX,LEN/500,0/
C
      COMMON/PSCOM/ICLOCK,IFMCNT,IX,IY,IPEN
C
      CALL PSINIT(2,0)
      I=1
C
C   READ THE TABLET AND DISPLAY THE CURSOR
C
100   IPEN=0
      CALL TABLET(0)
      CALL CURSOR(0)
C
C   IF THE PEN HAS NOT JUST BEEN LIFTED, GO TO 200
C
      IF(ISPCHD(IPEN).NE.3) GO TO 200
C
C   IF THERE'S ROOM, FIX THE LAST VALUES
C
      IF(I+1.GT.MAX) GO TO 300
      I=I+1
200   IARRAY(1,I)=IX
      IARRAY(2,I)=IY
C
C   DRAW THE LINES
C
300   CALL DRAW2D(IARRAY,I,2,2,0)
      CALL NUFRAM
      GO TO 100
```

Example 4.13-10

4.13.2 Control Dials Use [ANALOG]

The Control Dials option for PICTURE SYSTEM 2 consists of a Control Dial box with 8 stopless dials which may be used to input values for positioning, rotation angles, scaling, etc. The dials are interfaced to PICTURE SYSTEM 2 by means of an Analog to Digital (A/D) converter. This provides high precision data input and facilitates the connection of other analog input devices (e.g., joystick, trackballs, etc.) by means of this standard PICTURE SYSTEM 2 interface. Software support of the Control Dials is provided by the ANALOG subroutine. This subroutine is called to read and return the relative value that the specified dial has changed since the last time ANALOG was called to read that channel. Thus, the use of the Control Dials is independent of the initial physical position of the individual dials. This allows programs to present the same initial view of an object regardless of the Control Dial settings when the program is executed. The following is the ANALOG calling sequence specifiecation of Section 6.1:

```
CALL ANALOG(ICHANL,IVALUE)
```

ICHANL is an integer which specifies the device channel number which is to be read. This value may be in the range 0-31, providing support for up to 4 sets of Control Dials. If ANALOG is called to read a channel which has no Control Dial connected, an indeterminate value will be returned.

IVALUE is an integer variable in which the value read from the specified channel is to be returned. This value is in the range of approximately $\pm$32700 and is the relative value that the Control Dial has changed since the previous call to this routine. IVALUE will be returned with a value of zero the first time that ANALOG is called.

Example 4.13-11 illustrates how this subroutine may be used to obtain values that can be used directly in positioning objects for display, (translation, rotation, etc.).

```
            INTEGER IVALUE(6)
C
C   INITIAL DATA VALUES
C
            DATA IVALUE/6*0/
C
C   INIALIZE PICTURE SYSTEM 2
C
            CALL PSINIT(2,0)
            CALL WINDOW(5000,-5000,5000,-5000,
      1                 -5000,5000,-20000)
C
C   BEGIN DISPLAY LOOP
C   READING THE FIRST 6 ANALOG CHANNELS (0-5)
C
100         DO 110 I=1,6
            CALL ANALOG(I-1,J)
            IVALUE(I)=IVALUE(I)+J
110         CONTINUE
C
C   NOW UPDATE THE TRANSFORMATIONS
C
            CALL PUSH
            CALL TRAN(IVALUE(1),IVALUE(2),IVALUE(3))
            CALL ROT(1,IVALUE(4))
            CALL ROT(2,IVALUE(5))
            CALL ROT(3,IVALUE(6))
C
C   DRAW THE OBJECT (VIA SUBROUTINE)
C
            CALL OBJECT
            CALL POP
            CALL NUFRAM
            GO TO 100
            END
```

Example 4.13-11

4.13.3 Function Switches & Lights Use [ISWSET,SWITCH,SETLIT,LIGHTS]

Available for PICTURE SYSTEM 2 are two Function Switches & Lights options. One is a unit with 16 paddle switches each with 3 positions (OFF/ON/momentary ON) and a seperate incandescent light associated with each switch. The other is a unit with 32 back lighted buttons (1 row of 4, 4 rows of 6, and 1 row of 4) and an optional overlay sensor. Both of these options are supported by the same PICTURE SYSTEM Graphics Software subroutines.

The Function Switches & Lights may be used in two ways by the user. The switches and lights may be tested and set individually, or they may be read and set in banks of 16 switches and lights. Subroutines are provided to utilize these devices in both of these ways. The following are the calling sequence specifications of Section 6.1 for the subroutines provided to test and set individual Function Switches & Lights.

```
I = ISWSET(N)
```

ISWSET is an integer function subroutine which returns a value = 0 if the specified switch (N) is not set and = 1 if the switch is set. N is an integer which specifies which switch is to be used. N = 0-63 providing for support of up to 64 switches.

```
CALL SETLIT(N,ISTAT)
```

SETLIT is called to set or clear an individual light. N is an integer which specifies which light is to be set or cleared. N = 0-63 providing for support of up to 64 lights.

ISTAT is an integer which specifies whether the light is to be set or cleared. ISTAT = 0 to clear an individual light. ISTAT $\neq$ 0 to set the specified light on.

Example 4.13-12 shows how these two routines may be used together to test and indicate the status of a switch.

```
            .
            .
            .
C
C   GET SWITCH 0 STATUS (SET THE LIGHT ACCORDINGLY)
C
          I=ISWSET(0)
          CALL SETLIT(0,I)
C
C   WAS THE SWITCH SET (IF NOT GO TO 200)
C
          IF (I.EQ.0) TO TO 200
            .
            .
            .
```

Example 4.13-12

The following are the calling sequence specifications of Section 6.1 for the subroutines provided to utilize the Function Switches & Lights in banks of 16* switches and lights.

```
CALL SWITCH(IVALUE[,ISET])
```

SWITCH is called to return a 16-bit value from a set of function switches. IVALUE is an integer variable in which the 16-bit value is returned. Each bit of this value indicates the status of the switch corresponding to that bit (i.e., bit 0=1 indicates that switch 0 is set).

```
CALL LIGHTS(IVALUE[,ISET])
```

LIGHTS is called to set a 16-bit value into a set of 16 lights. IVALUE is an integer which specifies the value to be placed in to the lights. Each bit of this value represents whether the light corresponding to that bit is to be set or cleared (i.e., bit 0=1 indicates that light 0 is to be set).

---

*For the 32 button box, the banks of 16 are established as: buttons 15-0 = set 1. buttons 31-16 = set 2.

For both of the two subroutines above, an optional parameter, ISET, may be specified if a bank of Function Switches & Lights other than set 1 is to be used. If this parameter is omitted, Function Switch & Lights set 1 (buttons 15-0 for the 32 button box) is assumed.

Example 4.13-13 illustrates how these subroutines may be used.

```
            .
            .
            .
C
C   READ THE FUNCTION SWITCHES
C
          CALL SWITCH(I)
C
C   SET THE LIGHTS ACCORDINGLY
C
          CALL LIGHTS(I)
C
C   NOW TEST THE INDIVIDUAL SWITCHES
C
C   SWITCH 0
C
          IF(1.AND.I .NE.0) GO TO 200
C
C   SWITCH 1
C
          IF(2.AND.I .NE.0) GO TO 210
C
C   SWITCH 2
C
          IF(4.AND.I .NE.0) GO TO 220
            .
            .
            .
```

Example 4.13-13

4.13.4 Alphanumeric Keyboard Use [GETCHR]

The Alphanumeric Keyboard option for PICTURE SYSTEM 2 is a standard 61-key, 128-character keyboard which may be used to enter ASCII character data. Software support of the Alphanumeric Keyboard is provided by the GETCHR subroutine. This subroutine is called to return all characters that have been input since the last line terminator (i.e., Carriage Return, Line Feed or Form Feed). The count of the number of characters returned and a status indicator whether the character string is complete (i.e., a line terminator entered) are also returned. The following is the GETCHR calling sequence specification of Section 6.1:

```
CALL GETCHR(ICOUNT,IBUFF,ISTAT[,MAX])
```

ICOUNT is an integer variable where the count of the number of characters in the user supplied buffer (IBUFF) will be returned. If ICOUNT = 0 then no characters have been input since the previously entered line terminator.

IBUFF is an integer array into which all characters which have been input, since initialization or the last line terminator, will be placed. This buffer should be 80 bytes in length (or MAX bytes in length, if the MAX parameter is specified). Note that the line terminator is not returned to the user buffer.

ISTAT is an integer variable which is set by GETCHR to indicate the status of the character string returned in the user supplied buffer. ISTAT = 0 to indicate the character string is not complete. ISTAT = 1 to indicate the character string is complete. If ISTAT = 1, the next time GETCHR is called, only those characters entered after the last line terminator encountered will be returned.

MAX is an integer which, if specified, is the maximum number of characters to return in the user supplied buffer. If MAX is not specified, then up to 80 characters may be returned. MAX must be < 80.

The use of the GETCHR subroutine provides the capability of polling the character input buffer without the necessity of waiting for the entire character string to be input before control is returned to the program. This allows frame updates to proceed while character input takes place. GETCHR also provides user editing functions without application program concern. The striking of the "delete" (or rubout) key causes the previous character to be deleted from the input buffer. Control U (^U) causes the entire line being input to be deleted from the buffer.

The character count and the user buffer contents can be used directly in the TEXT subroutine call to cause the characters input to be displayed. Example 4.13-14 illustrates how this may be done.

```
          .
          .
          .
C
C  GET CHARACTER INPUT AND DISPLAY IT
C
          CALL GETCHR(ICOUNT,IBUFF,I)
          CALL TEXT(ICOUNT,IBUFF)
           .
           .
           .
```

Example 4.13-14

The characters input, however, need not be processed until a line terminator is encountered. Example 4.13-15 illustrates this.

```
          .
          .
          .
C
C GET CHARACTER INPUT
C
          CALL GETCHR(ICOUNT,IBUFF,I)
          CALL TEXT(ICOUNT,IBUFF)
C
C  INPUT COMPLETE?  (IF EQ NO)
C
          IF(I.EQ.0) GO TO 300
C
C  INPUT COMPLETE...DECODE THE CHARACTERS
C
          DECODE(ICOUNT,101,IBUFF) X
101       FORMAT(F7.5)
           .
           .
           .
```

Example 4.13-15

CHAPTER FIVE

5. EXAMPLE PROGRAMS

The implementation of a graphics program is very similar to the writing and debugging of any other type of computer program. One must carefully design, code and debug the program created. However, there is one substantial difference which makes the debugging phase of the implementation much easier. Once a picture is displayed, it can be visually inspected and interacted with to determine if the program is working properly. Thus, when implementing a graphics program, it is often best to display a simple picture and then proceed to progressively more difficult program structures and pictures. The example programs contained in the chapter may be used as a guide in the creation of a graphics program for PICTURE SYSTEM 2.

5.1 A SIMPLE PROGRAM

This program illustrates how simple it is to display an object which is moving dynamically, using the PICTURE SYSTEM 2 Graphics Software Package. The object that is displayed is a two-dimensional square which rotates continuously about the Z axis. Since the object is two-dimensional, no WINDOW is established and the object is transformed only by the rotation matrix generated.

```
C  SIMPLE DISPLAY PROGRAM
C
      INTEGER IARRAY(2,5)
      DATA /25000,25000, -25000,25000, -25000,-25000,
     1      25000,-25000, 25000,25000/
C
C  INITIALIZE PICTURE SYSTEM 2
C
      CALL PSINIT(2,0)
      IANGLE = 0
C
C  BEGIN THE DISPLAY LOOP BY SAVING THE
C  CURRENT TRANSFORMATION MATRIX
C
100   CALL PUSH
C
C  GENERATE THE ROTATION MATRIX
C
      CALL ROT(IANGLE,3)
C
C  OUTPUT THE OBJECT
C
      CALL DRAW2D(IARRAY,5,2,2,0)
      CALL POP
C
C  DISPLAY THE FRAME
C
      CALL NUFRAM
C
C  UPDATE THE CURRENT ANGLE AND CONTINUE
C
      IANGLE = IANGLE+182
      GO TO 100
      END
```

5.2 LINEAR DISPLAY LIST GENERATION

This section details a subroutine used to create and display a sphere using the linear display list software. This sphere is created using 1468 individual sine and cosine calculations (using the FORTRAN SIN and COS function subroutines). The time required for the SIN and COS calculations is prohibitive to allow dynamic motion of the sphere if the calculations were necessary for each new frame. However, the sphere can be generated once using the linear display list software and displayed thereafter without the need to recompute the object.

The following is the FORTRAN source listing for this subroutine.

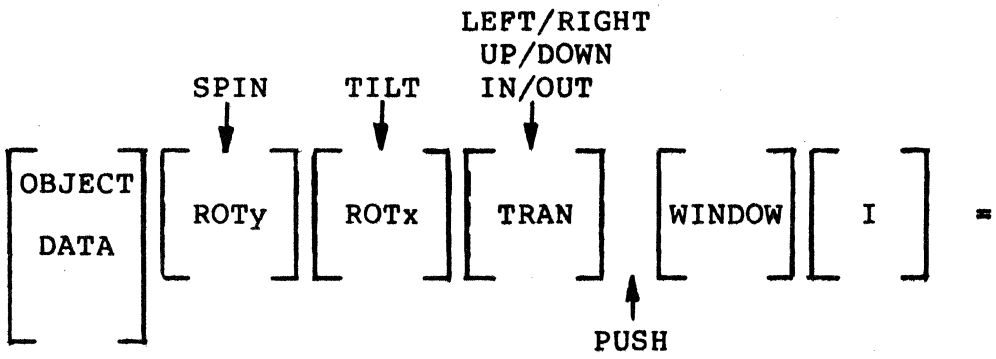
```
      SUBROUTINE OBJECT
C
C  THIS PROGRAM GENERATES A SPHERE CONSISTING OF
C  17 "LATTITUDE" LINES AND 36 "LONGITUDE LINES.
C
      REAL RADIAN,RADIUS,THETA,PHI,T,X,Y,Z
      INTEGER IARRAY(5000),IBUFF(300),FIRST
C
      DATA RADIAN,RADIUS/.0175,10000./
      DATA FIRST/0/
C
      IF(FIRST.NE.0) GO TO 100
      FIRST = 1
      CALL MAKEOB(IARRAY,5000,LEN)
C
      DO 30 IQ=1,4
C
C  CREATE "LONGITUDE" LINES
C
       DO 10 I=1,9
       THETA = RADIAN*10*(I-1)
       X = RADIUS*COS(THETA)
       Y = RADIUS*SIN(THETA)
       K = 0
        DO 9 J=1,19
        PHI = RADIAN*10*(J-10)
        T = COS(PHI)
        IBUFF(K+1) = X*T
        IBUFF(K+2) = Y*T
        IBUFF(K+3) = RADIUS*SIN(PHI)
 9      K = K+3
10     CALL DRAW3D(IBUFF,K/3,2,2)
C
C  CREATE "LATTITUDE" LINES
C
       DO 20 J=1,17
       PHI = RADIAN*10*(J-9)
       T = RADIUS*COS(PHI)
       Z = RADIUS*SIN(PHI)
       K = 0
        DO 19 I=1,10
        THETA = RADIAN*10*(I-1)
        IBUFF(K+1) = T*COS(THETA)
        IBUFF(K+2) = T*SIN(THETA)
        IBUFF(K+3) = Z
19      K = K+3
20     CALL DRAW3D(IBUFF,K/3,2,2)
30    CALL ROT(16384,3)
C
      CALL STOPOB
```

```
C
C  DISPLAY THE SPHERE
C
100     CALL DRAWOB(IARRAY)
        RETURN
        END
```

5.3 TABLET CONTROL PROGRAM

This program illustrates how a basic viewing program could be implemented which could easily be used to view a three-dimensional object, such as the sphere of the previous section. This program uses only the Data Tablet to control the viewing of the object. The menu is created as a segment and the object is then continuously updated as an unnamed dynamic segment.

The menu functions that may be performed are: TILT, SPIN, UP, DOWN, IN, OUT, LEFT, RIGHT. TILT and SPIN are rotations about the X and Y axes respectively. The remaining functions are merely performed by means of translation. Thus, the matrix transformations that are applied may be diagrammed as follows:



The following is the FORTRAN source listing for this program.

```
C TABLET CONTROL PROGRAM
C
      INTEGER MENU(3,8), SEGBUF(9)
      INTEGER ITX, ITY, ITZ, IRX, IRY
C
      COMMON /PSCOM/ICLOCK,IFRMCT,IX,IY,IPEN
C
      DATA MENU/'LE','FT','  ', 'RI','GH','T ',
     1          'UP','  ','  ', 'DO','WN','  ',
     2          'IN','  ','  ', 'OU','T ','  ',
     3          'SP','IN','  ', 'TI','LT','  '/
      DATA ITX,ITY,ITZ,IRX,IRY/5*0/
C
C  INITIALIZE PICTURE SYSTEM 2
C
      CALL PSINIT(2,0)
C
C  CREATE THE MENU AS SEGMENT 1
C
      CALL CLEARS(SEGBUF,9)
      CALL OPENS(1)
      CALL CHARSZ(4,0)
      J=27000
      DO 10 I=1,8
      CALL MOVETO(28000,J)
      CALL TEXT(6,MENU(1,I))
      J=J-8000
10    CONTINUE
      CALL CLOSES
C
C  SET THE WINDOW AND VIEWPORT FOR THE OBJECT
C
      CALL WINDOW(-6000,6000,-6000,6000,
                  -6000,6000,-30000,8192)
      CALL VWPORT(-1750,1750,-1750,1750,255,20)
C
C  DISPLAY LOOP
C
100   CALL TABLET(0)
      IF(IX.GE.28000) CALL CURSOR(0)
      I=ISPCHD(IPEN)
C
C  IF THE PEN IS NOT DOWN, JUST DISPLAY THE OBJECT
C
      IF(I.EQ.0 .OR. I.EQ.3) GO TO 200
C
C  ELSE PERFORM THE SELECTED FUNCTION
C
      I=IABS(IY)/8000 + 1
      IF(IY.GE.0) GO TO(140,130,120,110) I
      GO TO(150,160,170,180) I
```

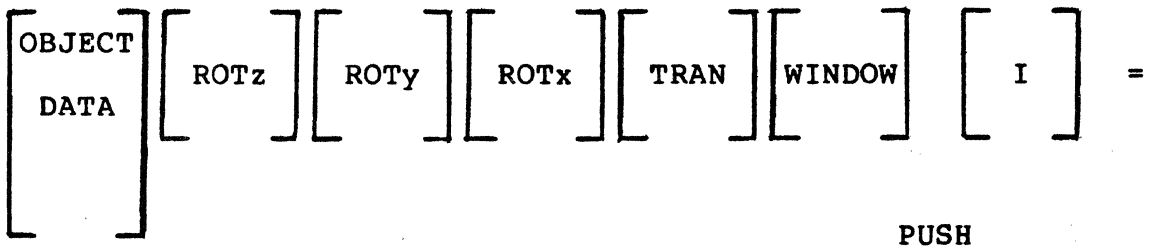
```
C
C  LEFT
C
110     ITX=ITX - 450
        GO TO 200
C
C  RIGHT
C
120     ITX=ITX + 450
        GO TO 200
C
C  UP
C
130     ITY=ITY + 450
        GO TO 200
C
C  DOWN
C
140     ITY=ITY - 450
        GO TO 200
C
C  IN
C
150     ITZ=ITZ - 450
        GO TO 200
C
C  OUT
C
160     ITZ=ITZ + 450
        GO TO 200
C
C  SPIN
C
170     IRY=IRY + 182
        GO TO 200
C
C  TILT
C
180     IRX=IRX + 182
C
C  PERFORM THE TRANSFORMATIONS AND DISPLAY THE OBJECT
C
200     CALL PUSH
        CALL TRAN(ITX,ITY,ITZ)
        CALL ROT(IRX,1)
        CALL ROT(IRY,2)
        CALL OBJECT
        CALL POP
        CALL NUFRAM
        GO TO 100
        END
```

5.4 CONTROL DIALS PROGRAM

This program is similar to the program of the previous section in that it is a general program which can be used to view a three-dimensional object, such as the sphere of Section 5.2. However, this program uses the Control Dials to manipulate the object viewing parameters. No menu is displayed, since all interaction is performed using the Control Dials. In this program, each of the eight Control Dials have an individual function to perform. Those functions are:

1. Rotation about X, Y and Z axes.
2. Translation in X and Y.
3. Zooming.
4. Positioning of the hither clipping plane.
5. Setting of the thickness from the hither to yon clipping planes. This, in conjunction with 4 above, allows the object to be viewed as a variable thickness cross-section.

The matrix transformations that are applied to perform these functions may be diagrqammed as follows:



The following is the FORTRAN source listing for this program.

```
C      EVANS & SUTHERLAND COMPUTER CORPORATION       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  CONTROL DIALS OBJECT MANIPULATION PROGRAM
C
C  THIS PROGRAM USES THE CONTROL DIALS TO MANIPULATE A
C  THREE-DIMENSIONAL OBJECT.  THE CONTROL DIAL ASSIGN-
C  MENTS ARE:
C
C       0. X AXIS ROTATION
C       1. Y AXIS ROTATION
C       2. Z AXIS ROTATION
C       3. HITHER PLANE POSITION
C       4. X TRANSLATION
C       5. Y TRANSLATION
C       6. ZOOM (WINDOW BOUNDARY POSITION)
C       7. YON PLANE POSITION FROM THE HITHER PLANE
C
C
C  PROGRAMMER:  L. KNAPP
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
       INTEGER CDIALS(8),HITHER,YON,SIZE,IHI,IYI
C
C  SET INITIAL VIEW VALUES
C
       DATA CDIALS/0,0,0,-32767,0,0,0,32767/
C
C  INITIALIZE PICTURE SYSTEM 2
C
       CALL PSINIT(2,0)
C
100    CALL PUSH
C
C  READ THE CURRENT CONTROL DIAL VALUES
C
       DO 110 I=1,3
       CALL ANALOG(I-1,J)
       CDIALS(I)=CDIALS(I) + J
110    CONTINUE
C
C  DON'T ALLOW ADDITION OVERFLOW FOR THE REST
C
       DO 120 I=4,8
       CALL ANALOG(I-1,J)
       CALL ADDNOV(J,CDIALS(I))
120    CONTINUE
```

```
C
C  DETERMINE CURRENT WINDOW BOUNDARIES
C  (HITHER AND YON FIRST)
C
      HITHER = CDIALS(4)/4
      YON= HITHER + CDIALS(8)/4 + 8196
      IF(YON.GT.8188) YON = 8188
C
C  WINDOW SIZE
C
      J = -(CDIALS(7)/10) + 3326
      SIZE = J*(FLOAT(HITHER)/24575. + 1.333)
C
C  SET VIEWPORT BOUNDARIES
C
      IHI = -HITHER/64 + 127
      IYI = -YON/64 + 127
      CALL VWPORT(-2048,2047,-2048,2047,IHI,IYI)
C
C  NOW PERFORM ALL THE TRANSFORMATIONS
C
      CALL WINDOW(-SIZE,SIZE,-SIZE,SIZE,
     1            HITHER,YON,-32767,4096)
      CALL TRAN(CDIALS(5),CDIALS(6),0)
      DO 130 I=1,3
      CALL ROT(CDIALS(I),I)
130   CONTINUE
C
C  NOW DRAW THE OBJECT (VIA THE OBJECT SUBROUTINE)
C
      CALL OBJECT
      CALL NUFRAM
      CALL POP
      GO TO 100
      END
```

Note: The ADDNOV subroutine is a special purpose subroutine written to allow the addition of two numbers without allowing overflow (e.g., 32000 + 1000 = 32767 instead of -32535).

5.5 THE ARCHITECTURE PROGRAM

The Architecture Program is a FORTRAN program written using PICTURE SYSTEM 2 Graphics Software subroutines. When executed the program display is that shown in Figure 5.5-1.

As Figure 5.5-1 shows, the screen is partitioned into three main areas:

1. The three-dimensional platform area (A).
2. The object areas (B).
3. The menu commands (C).

The three-dimensional platform is a plane which may be manipulated in 3-space and upon which objects (selected from the MENU) may be placed. The arrow in the rear-right corner of the platform is shown to indicate the direction in which the platform should be pointing for the PICTURE SYSTEM 2 tablet to map properly onto the platform. This allows the cursor to move right on the screen when the pen is moved right on the tablet. Note that as the cursor moves from the MENU to the platform, it too becomes a three-dimensional entity.

The object areas of the menu show a three-dimensional view of the eight objects which, when the "PLACE" command has been selected, may be pointed at to indicate which object is to be placed on the platform.

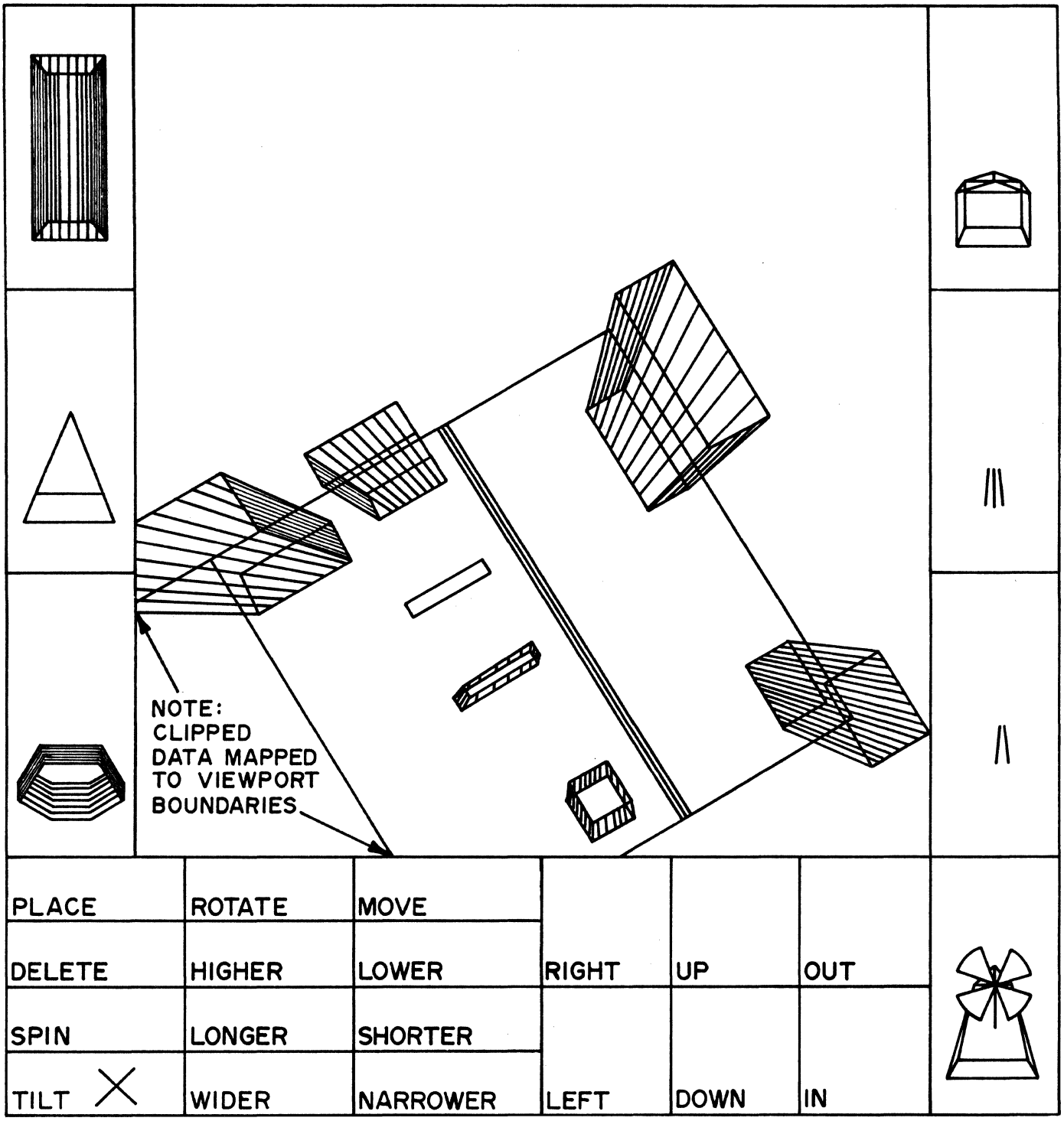


Figure 5.5-1

The Architecure Program

The menu commands are those commands which may be performed (1) on the objects or (2) on the platform and all of the objects on it. The top three rows of commands are those commands which may be performed on the objects:

PLACE - When this command is selected, an object may be pointed at and the cursor can then be moved to indicate where the object is to be placed. All commands selected remain in effect until another command is selected; thus, several objects may be placed without reselecting the PLACE command.

ROTATE - When this command is selected, the cursor can be moved to the platform and the pen pressed beneath an object to cause the object to rotate about its vertical axis. Note that the direction (- or +) and the amount the object is rotated may be selected on the "-<<<0>>>+" row as described below.

MOVE - When this command is selected, the cursor can be moved to the platform and the pen pressed beneath an object to select it to be moved to the next place the pen is pressed down. Note that the object selected follows the cursor until the pen is pressed down again.

DELETE - When this command is selected, the cursor can be moved to the platform and the pen pressed beneath an object to delete that object.

HIGHER or LOWER -
When this command is selected, the cursor can be moved to the platform and the pen pressed beneath an object to make that object grow higher or lower.

LONGER or SHORTER -
When this command is selected, the cursor can be moved to the platform and the pen pressed beneath an object to make that object grow longer or shorter.

WIDER or NARROWER -
When this command is selected, the cursor can be moved to the platform and the pen pressed beneath an object to make that object grow wider or narrower.

RAISE or DESCEND -
When this command is selected, the cursor can

be moved to the platform and the pen pressed beneath an object to make that object raise from or descend to the platform.

It should be noted that for the above commands (HIGHER, LOWER, LONGER, SHORTER, WIDER, NARROWER, RAISE and DESCEND) the rate at which the command is performed is determined by the absolute value of the current value selected on the "-<<<0>>>+" row as described below.

The next two rows of commands are those commands which may be performed for the platform and all of the objects on it:

SPIN - When this command is selected, the platform will spin about its vertical axis at the rate and in the direction indicated by the current value selected from the "-<<<0>>>+" row.

TILT - When this command is selected, the platform will tilt about the x-axis at the rate and in the direction indicated by the current value selected from the "-<<<0>>>+" row.

ZOOM - When this command is selected, the platform will increase (+) or decrease (-) in size and at the rate indicated by the current value selected from the "-<<<0>>>+" row.

RIGHT/LEFT - When this command is selected, the platform will move right (+) or left (-) at the rate indicated by the current value selected from the "-<<<0>>>+" row.

UP/DOWN - When this command is selected, the platform will move up (+) or down (-) at the rate indicated by the current value selected from the "-<<<0>>>+" row.

IN/OUT - When this command is selected, the platform will move out (+) or in(-) at the rate indicated by the current values selected from the "-<<<0>>>+" row.

"-<<<0>>>+" - When this command is selected, the value is used to perform the current function selected if the function is SPIN, TILT, ZOOM, LEFT/RIGHT, UP/DOWN or IN/OUT. Thus, for example, the SPIN command may be selected and

then this row may be used to dynamically vary the rate and direction of rotation without reselecting the SPIN command. The last value selected on this row is also used to control the rate object commands are performed.

5.5.1 Implementation

The Architecture program is implemented as three FORTRAN modules; a main program and two subroutines. The functions of these modules are:

Main Program:

The MENU Subroutine is called to display the menu and to determine what menu area the pen is in. Hit testing is performed on each object, if required by the command selected.

All platform transformations are performed and the platform (and "3-D cursor" if necessary) is displayed. Each object's transformations are performed and the OBJCTS subroutine is called to display the individual objects.

Subroutine MENU:

This subroutine is called to (initially create and then) display the command menu and all object menu areas. The area in which the pen resides is determined and returned to the main program as a value 0-10 where:

- 0 = pen not in any menu area (i.e., on the platform).
- 1-8 = object number currently selected.
- 9 = pen down in menu command area.
- 10 = pen not down, but in the menu area.

If the value returned is 1-10, then a "2-D cursor" is displayed.

Subroutine OBJCTS:

This subroutine is called to (initially create and then) display the object indicated by the number passed as an argument to the subroutine. One of eight objects may be displayed during any given call. Note that the "blank COMMON" variables COUNT, ANGLE and BLDPT are used to facilitate the updating of the angle of rotation for the windmill blades (object 8).

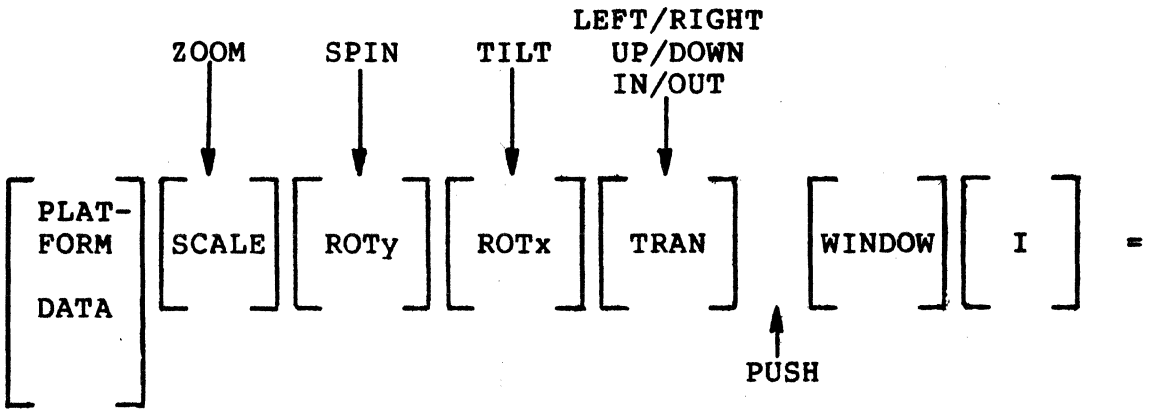
This subroutine has been structured to facilitate easy modification by any given user to substitute any object required for a given application for any current object definition. Each object (1-8) is defined by data statements and then incorporated as a linear display list using the MAKEOB, STOPOB and DRAWOB subroutines(1). Close inspection of the statements of this subroutine will reveal a FORTRAN WRITE and FORMAT statement associated with each object creation. These statements were used to determine the array size required for each linear display list and were made into comment statements to ease the modification of the subroutine by users. Note that each WRITE and FORMAT statement has a "C" in column one to make it only a comment statement.

To substitute a new object for a predefined object, the user should: (1) define the coordinates for the object so that the object's vertical axis (as it would be when resting on the platform) is the axis with Z = 0 as the base of the object and Z = -32767 as the top of the object, (2) modify the size of the array OBn (n=the object number, 1-8) so that the array will be just large enough for the linear display list, (3) modify the size of the array IARRAY in the MENU subroutine so that it is just large enough to hold the modified menu, (4) modify the calls to the MAKEOB subroutine to reflect the increased (or decreased) array sizes. Note that if the array specified to the MAKEOB subroutine is not large enough an "ERROR 3 IN GRAPHICS SUBROUTINE 33" will be output to the console terminal. The use of the "commented" WRITE and FORMAT statements may be useful in determin-

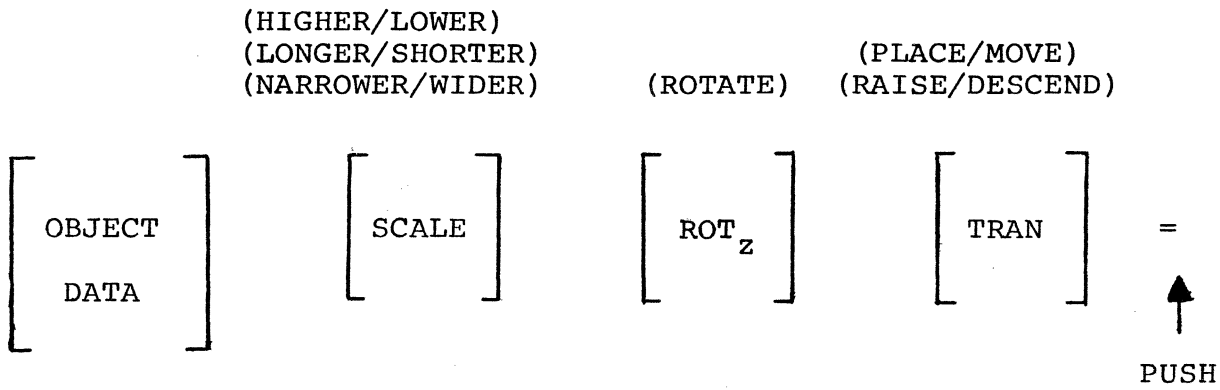
ing the exact size of the linear display lists.

5.5.2 Matrix Transformations

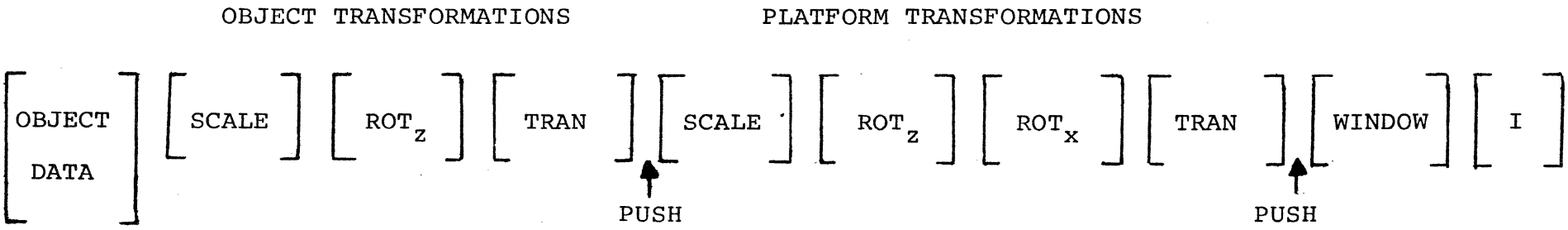
The matrix transformations that are applied to the platform may be diagrammed as follows:



The matrix transformations that are applied to any of the objects may be diagrammed as follows:



Thus the transformations applied to any object are:



5.5.3 The Architecture FORTRAN Program

Following are the source listings for the Architecture FORTRAN main program, subroutine MENU and subroutine OBJCTS:

```
C     EVANS & SUTHERLAND COMPUTER CORPORATION     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C THE ARCHITECTURE PROGRAM
C
C PROGRAMMER:  M. MANTLE
C
C
      INTEGER OMATRX(16,30)
      INTEGER OBJECT(8,30),NOBS
      INTEGER IDENT(16)
      INTEGER PLATBF(110),PLATFM(2,5),ARROW(2,6)
      INTEGER CURSX(8)
      INTEGER TY(5)
      INTEGER SPIN,TILT,ZOOM,LEFT,UP,IN,VALUE
      INTEGER SPINPT,TILTPT,LEFTPT,ZOOMPT
      INTEGER CURRNT,MENUF,MOVER,HITYET
      INTEGER ITEM,ITEMX,ITEMY
      INTEGER X,Y,ISIZE
      INTEGER INC
      INTEGER LASTOB
      INTEGER PENUP
      INTEGER ISPEN
C
      COMMON /PSCOM/ICLOCK,IFMCNT,IX,IY,IPEN
C
      DATA NOBS/30/
      DATA OBJECT/240*0/
      DATA SPIN/0/
      DATA TILT/16384/
      DATA ZOOM/4000/
      DATA LEFT/0/
      DATA UP/-2000/
      DATA IN/2500/
      DATA VALUE/100/
      DATA IDENT/O40000,0,0,0, 0,O40000,0,0,
     1           0,0,O40000,0, 0,0,0,O40000/
      DATA PLATFM/16384,16384,16384,-16384,
     1           -16384,-16384,-16384,16384,
     2            16384,16384/
      DATA ARROW/15000,16000, 15000,11000,
     1           15000,16000, 14000,15000,
     2           15000,16000, 16000,15000/
      DATA CURSX/0,0,-1000,-1000,1000,0,-1000,1000/
      DATA TY/-30034,-27304,-24574,-21844,-19114/
      DATA CURRNT/0/
      DATA MOVER/0/
      DATA HITYET/0/
      DATA ITEM/6/
      DATA INC/100/
      DATA LASTOB/0/
      DATA PENUP/0/
C
```

```
C  OBJECT(1,N) => OBJECT TYPE (1-8) (0=NO OBJECT)
C  OBJECT(2,N) => X PLACEMENT
C  OBJECT(3,N) => Y PLACEMENT
C  OBJECT(4,N) => ORIENTATION
C  OBJECT(5,N) => WIDER/NARROWER
C  OBJECT(6,N) => LONGER/SHORTER
C  OBJECT(7,N) => HIGHER/LOWER
C  OBJECT(8,N) => RAISE/DESCEND
C
C INITIALIZE PICTURE SYSTEM 2
C
      CALL PSINIT(2,0)
      CALL TABLET(1)
C
C CREATE THE PLATFORM AND TRANSFORMATIONS
C AS A LINEAR DISPLAY LIST
C
      CALL MAKEOB(PLATBF,110,LEN)
      LEFTPT=LEN
      CALL TRAN(LEFT,UP,IN)
      TILTPT=LEN
      CALL ROT(TILT,1)
      SPINPT=LEN
      CALL ROT(SPIN,3)
      ZOOMPT=LEN
      CALL SCALE(ZOOM,ZOOM,ZOOM)
      CALL DRAW2D(PLATFM,5,2,2,0)
      CALL DRAW2D(ARROW,6,0,2,0)
      CALL STOPOB
C     WRITE(7,9) LEN
C 9   FORMAT(' PLATFORM LENGTH=',I4)
C
C SET THE PERSPECTIVE WINDOW
C
      CALL WINDOW(-2000,2000,-2000,2000,
     1            -2000,-10000,-10000)
C
C DISPLAY THE MENU AND DETERMINE THE MENU REGION SELECTED
C
100   CALL MENU(MENUF)
C
C READY PEN STATUS FOR THIS UPDATE
C
      ISPEN=ISPCHD(IPEN)
C
C BRANCH ON MENU ITEM SELECTED
C  MENUF=0   => PEN NOT IN MENU AREA (DOWN OR OTHERWISE)
C  MENUF=1-8 => OBJECT   SELECTED
C  MENUF=9   => PEN DOWN IN MENU AREA
C  MENUF=10  => PEN NOT DOWN BUT IN MENU AREA
C
C
      II=MENUF+1
```

```
      GO TO(200,110,110,110,110,110,110,110,110,120,400) II
C
C OBJECT BEING POINTED AT IN MENU
C UPDATE THE CURRENT OBJECT IF THE PEN IS DOWN
C
110   IF(ISPEN.NE.0) CURRNT=MENUF
      GO TO 400
C
C MENU ITEM BEING POINTED AT...DETERMINE WHICH ONE
C
120   CURRNT=0
      HITYET=0
      ISIZE=1000
      ITEMY=1
      DO 121 J=1,5
      IF(IY.GT.TY(J)) ITEMY=J+1
121   CONTINUE
      GO TO(130,140,140,150,150,150) ITEMY
C
C VALUE UPDATE
C
130   VALUE=IX/50
131   IF(ITEM.GT.6) GO TO 400
      GO TO(132,133,134,135,136,137) ITEM
C
C UPDATE SELECTED FOR PLATFORM FUNCTION
C
132   LEFT=LEFT+VALUE
      GO TO 400
133   UP=UP+VALUE
      GO TO 400
134   IN=IN+VALUE
      GO TO 400
135   SPIN=SPIN+VALUE
      GO TO 400
136   TILT=TILT+VALUE
      GO TO 400
137   IF(ZOOM+VALUE.GT.200) ZOOM=ZOOM+VALUE
      GO TO 400
C
C DETERMINE WHICH MENU FUNCTION SELECTED
C
140   ITEMX=1
      IF(IX.GT.-8192) ITEMX=2
      IF(IX.GT. 8192) ITEMX=3
      ITEM=(ITEMY-2)*3+ITEMX
      VALUE=0
      GO TO 131
C
150   ITEMX=1
      IF(IX.GT.-12288) ITEMX=2
      IF(IX.GT.0)      ITEMX=3
      IF(IX.GT. 12288) ITEMX=4
```

```
        ITEM=(ITEMY-2)*4 - 2 + ITEMX
        VALUE=1000
        GO TO 400
C
C PEN NOT IN MENU AREA...DOWN OR OTHERWISE
C
C SCALE THE TABLET VALUES FOR THE PLATFORM
C
200     I=IY-10924
        X=IX*.75
        Y=I*.75
C
C SET "CURSOR" COORDINATES
C
        CURSX(1)=X+500
        CURSX(2)=Y+500
C
C IS A MOVE IN PROGRESS?  (IF SO GO TO 325)
C
        IF(MOVER.NE.0) GO TO 325
C
C BRANCH IF NO HIT TESTING FUNCTION TO PERFORM
C
        IF(ITEM.LT.7) GO TO 400
        IF(ITEM.EQ.15) GO TO 300
C
C IS THE PEN DOWN?
C (IF NOT RESET THE HIT FLAGS AND GO TO 400)
C
        IF(ISPEN.NE.0) GO TO 205
        HITYET=0
        ISIZE=1000
        GO TO 400
C
C OBJECT SELECTED ALREADY?
C (IF SO GO TO 215 AND PERFORM THE FUNCTION)
C
205     IF(HITYET.NE.0) GO TO 215
C
C PERFORM HIT TESTING
C START BY SAVING THE CURRENT TRANSFORMATION
C
        CALL PUSH
C
C AND LOAD AN IDENTITY MATRIX TO "FOOL" HITWIN
C
        CALL BLDCON(1,IDENT)
C
C SET THE HIT WINDOW
C
        CALL HITWIN(X,Y,ISIZE)
C
C AND HIT TEST FOR EACH OBJECT
```

```
C
      J=0
C
      IF(LASTOB.EQ.0) GO TO 211
C
      DO 210 I=1,LASTOB
C
C IF THERE'S NO OBJECT...JUST CONTINUE
C
      IF(OBJECT(1,I).EQ.0) GO TO 210
C
C SAVE THE CURRENT TRANSFORMATION
C AND PERFORM THE OBJECT TRANSFORMATIONS
C
      CALL PUSH
      CALL TRAN(0,0,10-OBJECT(8,I))
      CALL BLDCON(2,OMATRX(1,I))
      CALL OBJCTS(OBJECT(1,I))
      CALL HITEST(J,0)
      CALL POP
      IF(J.NE.0) GO TO 211
210   CONTINUE
      ISIZE=ISIZE+INC
211   CALL HITEST(K,1)
      CALL POP
      IF(J.EQ.0) GO TO 400
C
C OBJECT HIT...PERFORM THE SELECTED FUNCTION
C
      HITYET=I
215   I=HITYET
      J=IABS(VALUE)
      II=ITEM-6
      GO TO(220,230,240,250,260,270,280,290,
           300,310,320,330) II
C
C LOWER
C
220   IF(OBJECT(7,I)-J.GT.0) OBJECT(7,I)=OBJECT(7,I)-J
      GO TO 390
C
C
C SHORTER
C
230   IF(OBJECT(6,I)-J.GT.0) OBJECT(6,I)=OBJECT(6,I)-J
      GO TO 390
C
C NARROWER
C
240   IF(OBJECT(5,I)-J.GT.0) OBJECT(5,I)=OBJECT(5,I)-J
      GO TO 390
C
C DESCEND
```

```
C
250     OBJECT(8,I)=OBJECT(8,I)+J
        GO TO 390
C
C HIGHER
C
260     IF(OBJECT(7,I)+J.GT.0) OBJECT(7,I)=OBJECT(7,I)+J
        GO TO 390
C
C LONGER
C
270     IF(OBJECT(6,I)+J.GT.0) OBJECT(6,I)=OBJECT(6,I)+J
        GO TO 390
C
C WIDER
C
280     IF(OBJECT(5,I)+J.GT.0) OBJECT(5,I)=OBJECT(5,I)+J
        GO TO 390
C
C RAISE
C
290     OBJECT(8,I)=OBJECT(8,I)-J
        GO TO 390
C
C PLACE
C
300     IF(CURRNT.EQ.0) GO TO 400
        IF(LASTOB.EQ.NOBS) GO TO 400
        LASTOB=LASTOB+1
        I=LASTOB
C
C OBJECT AVAILABLE...PLACE THE OBJECT
C
302     OBJECT(1,I)=CURRNT
        OBJECT(2,I)=X
        OBJECT(3,I)=Y
        OBJECT(4,I)=0
        OBJECT(5,I)=3000
        OBJECT(6,I)=3000
        OBJECT(7,I)=3000
        OBJECT(8,I)=0
        CURRNT=0
        MOVER=I
        GO TO 390
C
C ROTATE
C
310     OBJECT(4,I)=OBJECT(4,I)+VALUE
        GO TO 390
C
C MOVE
C
320     MOVER=I
```

```
        PENUP=1
        GO TO 390
325     IF(PENUP.NE.0) GO TO 326
        IF(ISPEN.NE.3) GO TO 326
        MOVER=0
        HITYET=0
        ISIZE=1000
        GO TO 400
326     IF(ISPEN.NE.3) GO TO 327
        PENUP=0
327     OBJECT(2,MOVER)=X
        OBJECT(3,MOVER)=Y
        I=MOVER
        GO TO 390
C
C DELETE
C
330     IF(I.EQ.LASTOB) GO TO 333
        DO 331 J=1,8
        OMATRX(J,I)=OMATRX(J,LASTOB)
331     OBJECT(J,I)=OBJECT(J,LASTOB)
        DO 332 J=8,16
332     OMATRX(J,I)=OMATRX(J,LASTOB)
333     OBJECT(1,LASTOB)=0
        LASTOB=LASTOB-1
        ITEM=15
        GO TO 400
C
C SAVE THE OBJECT'S COMPOUND TRANSFORMATION
C
390     CALL PUSH
        CALL BLDCON(1,IDENT)
        CALL TRAN(OBJECT(2,I),OBJECT(3,I),OBJECT(8,I))
        CALL ROT(OBJECT(4,I),3)
        CALL SCALE(OBJECT(5,I),OBJECT(6,I),OBJECT(7,I))
        CALL BLDCON(3,OMATRX(1,I))
        CALL POP
C
C
C
C DISPLAY THE PLATFORM, OBJECTS AND (MAYBE) THE CURSOR
C
C RESET THE PEN FLAG AND SAVE THE CURRENT TRANSFORMATION
C
400     IPEN=0
        CALL PUSH
C
C INSERT THE PLATFORM TRANSFORMATIONS
C
        CALL GETTRN(PLATBF(LEFTPT+2),LEFT,UP,IN)
        CALL GETROT(PLATBF(TILTPT+2),TILT,1)
        CALL GETROT(PLATBF(SPINPT+2),SPIN,3)
        CALL GETSCL(PLATBF(ZOOMPT+2),ZOOM,ZOOM,ZOOM)
```

```
C
C DISPLAY THE PLATFORM AND "3D" CURSOR (IF NECESSARY)
C
      CALL DRAWOB(PLATBF)
      IF(MENUF.EQ.0) CALL DRAW2D(CURSX,4,0,0,0)
C
C DISPLAY THE OBJECTS
C
      IF(LASTOB.EQ.0) GO TO 420
C
      DO 410 I=1,LASTOB
C
C SAVE THE PLATFORM TRANSFORMATION AND
C PERFORM THE OBJECT TRANSFORMATIONS
C
      IF(OBJECT(1,I).EQ.0) GO TO 410
      CALL PUSH
      CALL BLDCON(2,OMATRX(1,I))
      CALL OBJCTS(OBJECT(1,I))
      CALL POP
410   CONTINUE
C
C SET TO DISPLAY THE NEW FRAME
C
420   CALL NUFRAM
C
C RESTORE THE ORIGINAL TRANSFORMATION AND LOOP AGAIN
C
      CALL POP
      GO TO 100
C
      END
C
C
C
```

```
      SUBROUTINE MENU(NUMBER)
C
      LOGICAL DONE
      INTEGER IARRAY(1660)
      INTEGER LINES(2,42),IDENT(16),BOUNDS(4)
      INTEGER VL(8),VR(8),VB(8),VT(8)
      INTEGER COUNT,ANGLE,BLDPT,MBLDPT
C
      COMMON /PSCOM/ICLOCK,IFMCNT,IX,IY,IPEN
      COMMON COUNT,ANGLE,BLDPT
C
      DATA DONE/.FALSE./
      DATA LINES/-32767,32767,32767,32767,
     1           -32767,16384,-24576,16384,
     2           -32767,0,-24576,0,
     3           -32767,-16384,32767,-16384,
     4           -32760,-32760,32760,-32760,
     5            24576,16384,32767,16384,
     6            24576,0,32767,0,
     7           -24576,-19114,24576,-19114,
     8           -24576,-21844,24576,-21844,
     9           -24576,-24576,24576,-24576,
     0           -24576,-27304,24576,-27304,
     1           -24576,-30034,24576,-30034,
     2           -24576,32767,-24576,-32767,
     3            24576,32767,24576,-32767,
     4           -12288,-16384,-12288,-24576,
     4            0,-16384,0,-24576,
     5            12288,-16384,12288,-24576,
     6           -8192,-24576,-8192,-30034,
     7            8192,-24576,8192,-30034,
     8           -32760,32760,-32760,-32760,
     9            32767,32767,32767,-32767/
      DATA IDENT/O40000,0,0,0, 0,O40000,0,0,
     1    0,0,O40000,0, 0,0,0,O40000/
      DATA BOUNDS/32767,16384,0,-16384/
      DATA VL/-2048,-2048,-2048,-2048,1536,1536,1536,1536/
      DATA VR/-1536,-1536,-1536,-1536,2047,2047,2047,2047/
      DATA VB/1024,0,-1024,-2048,1024,0,-1024,-2048/
      DATA VT/2047,1024,0,-1024,2047,1024,0,-1024/
C
C HAS THE MENU ALREADY BEEN CREATED?  (IF IT HAS GO TO 100)
C
      IF(DONE) GO TO 100
C
C CREATE THE OBJECTS FIRST
C
      DO 9 I=1,8
9     CALL OBJCTS(I)
C
C CREATE THE MENU AS A LINEAR DISPLAY LIST
C
      CALL MAKEOB(IARRAY,1660,LEN)
```

```
C
C SAVE THE CURRENT TRANSFORMATION
C
      CALL PUSH
C
C AND LOAD AN IDENTITY MATRIX
C
      CALL BLDCON(1,IDENT)
C
C SET THE VIEWPORT FOR THE ENTIRE SCREEN
C
      CALL VWPORT(-2048,2047,-2048,2047,255,255)
C
C DISPLAY THE MENU BOX OUTLINES
C
      CALL DRAW2D(LINES,42,0,2,0)
      CALL CHARSZ(5,0)
C
C AND DISPLAY THE MENU TEXT
C
      CALL MOVETO(-23500,-32000)
      CALL TEXT(35,'- <<<<<<<<<<<<<< 0 >>>>>>>>>>>>>> +')
      CALL MOVETO(-24000,-29600)
      CALL TEXT(10,'RIGHT/LEFT')
      CALL MOVETO(-7500,-29600)
      CALL TEXT(7,'UP/DOWN')
      CALL MOVETO(9000,-29600)
      CALL TEXT(6,'IN/OUT')
      CALL MOVETO(-24000,-27100)
      CALL TEXT(4,'SPIN')
      CALL MOVETO(-7500,-27100)
      CALL TEXT(4,'TILT')
      CALL MOVETO(9000,-27100)
      CALL TEXT(4,'ZOOM')
      CALL MOVETO(-24000,-24000)
      CALL TEXT(5,'LOWER')
      CALL MOVETO(-11500,-24000)
      CALL TEXT(7,'SHORTER')
      CALL MOVETO(700,-24000)
      CALL TEXT(8,'NARROWER')
      CALL MOVETO(13000,-24000)
      CALL TEXT(7,'DESCEND')
      CALL MOVETO(-24000,-21400)
      CALL TEXT(6,'HIGHER')
      CALL MOVETO(-11500,-21400)
      CALL TEXT(6,'LONGER')
      CALL MOVETO(700,-21400)
      CALL TEXT(5,'WIDER')
      CALL MOVETO(13000,-21400)
      CALL TEXT(5,'RAISE')
      CALL MOVETO(-24000,-18500)
      CALL TEXT(5,'PLACE')
      CALL MOVETO(-11500,-18500)
```

```
        CALL TEXT(6,'ROTATE')
        CALL MOVETO(700,-18500)
        CALL TEXT(4,'MOVE')
        CALL MOVETO(13000,-18500)
        CALL TEXT(6,'DELETE')
        CALL WINDOW(-3000,3000,-2500,9500,
       1            -6000,16000,-32000,16384)
        CALL ROT(16384,1)
        CALL SCALE(10000,10000,10000)
C
        DO 10 I=1,8
        CALL VWPORT(VL(I),VR(I),VB(I),VT(I),255,50)
        MBLDPT=LEN
        CALL OBJCTS(I)
10      CONTINUE
C
        CALL POP
        CALL VWPORT(-1536,1536,-1024,2047,255,0)
C
        CALL STOPOB
C       WRITE(7,11) LEN
C  11   FORMAT(' MENU LENGTH=',I4)
C
        DONE=.TRUE.
C
C SET THE POINTER TO THE MENU WINDMILL BLADES
C
        MBLDPT=MBLDPT+BLDPT+1
        ANGLE=0
C
C UPDATE THE ROTATION FOR THE MENU WINDMILL BLADES
C
100     ANGLE=ANGLE+300
        COUNT=COUNT+1
        CALL GETROT(IARRAY(MBLDPT),ANGLE,2)
C
C DRAW THE MENU
C
        CALL DRAWOB(IARRAY)
C
C SET THE DEFAULT MENU RETURN VALUE
C
        NUMBER=0
C
C DETERMINE WHERE THE PEN IS
C
        IF(IX.GT.-24576.AND.IX.LT.24576) GO TO 115
        DO 110 I=1,4
110     IF(IY.LE.BOUNDS(I)) NUMBER=NUMBER+1
        IF(IX.GE.24576) NUMBER=NUMBER+4
        GO TO 120
115     IF(IY.GT.-16384) GO TO 130
        NUMBER=10
```

```
      IF(ISPCHD(IPEN).EQ.1 .OR. ISPCHD(IPEN).EQ.2) NUMBER=9
120   CALL CURSOR(0)
130   RETURN
      END
C
C
C
```

```
      SUBROUTINE OBJCTS(NUMBER)
C
      LOGICAL DONE
      INTEGER OB1(280),OB2(60),OB3(150),OB4(280),
      INTEGER OB5(100),OB6(40),OB7(20),OB8(170)
      INTEGER BARS(3,18),BOTTOM(2,5),TOP(3,8),OVAL(2,9)
      INTEGER ROAD(2,4),DLINE(2,2),WALK(2,4)
      INTEGER SIDES1(3,5),SIDES2(3,5),BEAM(3,2)
      INTEGER MILL(2,5),MSIDE1(3,5),MSIDE2(3,5)
      INTEGER CENTER(3,3),BLADE(3,11)
      INTEGER COUNT,OLDCNT,ANGLE,BLDPT
C
      COMMON COUNT,ANGLE,BLDPT
C
      DATA DONE/.FALSE./
      DATA BARS/-16384,16384,-32767, -16384,16384,0,
     1          -12288,16384,-32767, -12288,16384,0,
     2          -8192,16384,-32767,  -8192,16384,0,
     3          -4096,16384,-32767,  -4096,16384,0,
     4           0,16384,-32767,      0,16384,0,
     5           4096,16384,-32767,   4096,16384,0,
     6           8192,16384,-32767,   8192,16384,0,
     7           12288,16384,-32767,  12288,16384,0,
     8           16384,16384,-32767,  16384,16384,0/
      DATA BOTTOM/16384,16384, 16384,-16384,
     1            -16384,-16384, -16384,16384,
     2            16384,16384/
      DATA TOP/0,0,-32767, -16384,-16384,0,
     1         0,0,-32767, -16384,16384,0,
     2         0,0,-32767, 16384,16384,0,
     3         0,0,-32767, 16384,-16384,0/
      DATA OVAL/-8192,32767, 8192,32767,
     1          16384,16384, 16384,-16384,
     2          8192,-32767, -8192,-32767,
     3          -16384,-16384, -16384,16384,
     4          -8192,32767/
      DATA ROAD/-4096,32767, -4096,-32767,
     1          4096,32767, 4096,-32767/
      DATA DLINE/0,32767, 0,-32767/
      DATA WALK/-2048,32767, -2048,-32767,
     1          2048,32767, 2048,-32767/
      DATA SIDES1/-16384,16384,0, -16384,16384,-16384,
     1            0,16384,-20480, 16384,16384,-16384,
     2            16384,16384,0/
      DATA SIDES2/-16384,-16384,0, -16384,-16384,-16384,
     1             0,-16384,-20480, 16384,-16384,-16384,
     2             16384,-16384,0/
      DATA BEAM/0,16384,-20480, 0,-16384,-20480/
      DATA MILL/12288,12288, 12288,-12288, -12288,-12288,
     1          -12288,12288, 12288,12288/
      DATA MSIDE1/-16384,16384,0, -12288,12288,-24576,
     1            0,0,-32767, 12288,12288,-24576,
     2            16384,16384,0/
```

```
      DATA MSIDE2/-16384,-16384,0, -12288,-12288, -24576,
     1            0,0,-32767, 12288,-12288,-24576,
     2            16384,-16384,0/
      DATA CENTER/0,0,0, 0,0,-24576, 0,-16384,-24576/
      DATA BLADE/5000,0,-12288, -5000,0,12288,
     1           5000,0,12288, 0,0,0,
     2           12288,0,5000, 12288,0,-5000,
     3           -12288,0,5000, -12288,0,-5000,
     4           0,0,0, -5000,0,-12288, 5000,0,-12288/
C
C BRANCH ON OBJECT NUMBER
C
      GO TO(100,200,300,400,500,600,700,800) NUMBER
C
C OBJECT 1
C
100   IF(DONE) GO TO 150
      CALL MAKEOB(OB1,280,LEN)
      CALL DRAW3D(BARS,18,0,2)
      DO 110 I=1,18
110   BARS(2,I)=-BARS(2,I)
      CALL DRAW3D(BARS,18,0,2)
      DO 120 I=1,18
      BARS(2,I)=BARS(1,I)
120   BARS(1,I)=16384
      CALL DRAW3D(BARS,18,0,2)
      DO 130 I=1,18
130   BARS(1,I)=-BARS(1,I)
      CALL DRAW3D(BARS,18,0,2)
      CALL DRAW2D(BOTTOM,5,2,2,-32767)
      CALL DRAW2D(BOTTOM,5,2,2,0)
      CALL STOPOB
C     WRITE(7,140) LEN
C 140 FORMAT(' OBJECT 1 LENGTH=',I4)
      RETURN
150   CALL DRAWOB(OB1)
      RETURN
C
C OBJECT 2
C
200   IF(DONE) GO TO 250
      CALL MAKEOB(OB2,60,LEN)
      CALL DRAW2D(BOTTOM,5,2,2,0)
      CALL DRAW3D(TOP,8,0,2)
      CALL STOPOB
C     WRITE(7,240) LEN
C 240 FORMAT(' OBJECT 2 LENGTH=',I4)
      RETURN
250   CALL DRAWOB(OB2)
      RETURN
C
C OBJECT 3
C
```

```
300     IF(DONE) GO TO 350
        CALL MAKEOB(OB3,150,LEN)
        J=0
        DO 310 I=1,6
        CALL DRAW2D(OVAL,9,2,2,J)
310     J=J-2048
        CALL STOPOB
C       WRITE(7,340) LEN
C 340   FORMAT(' OBJECT 3 LENGTH=',I4)
        RETURN
350     CALL DRAWOB(OB3)
        RETURN
C
C OBJECT 4
C
400     IF(DONE) GO TO 450
        CALL MAKEOB(OB4,280,LEN)
        DO 410 I=1,32000,2000
410     CALL DRAW2D(BOTTOM,5,2,2,1-I)
        CALL STOPOB
C       WRITE(7,440) LEN
C 440   FORMAT(' OBJECT 4 LENGTH=',I4)
        RETURN
450     CALL DRAWOB(OB4)
        RETURN
C
C OBJECT 5
C
500     IF(DONE) GO TO 550
        CALL MAKEOB(OB5,100,LEN)
        CALL DRAW2D(BOTTOM,5,2,2,0)
        CALL DRAW2D(BOTTOM,5,2,2,-16384)
        CALL DRAW3D(SIDES1,5,2,2)
        CALL DRAW3D(SIDES2,5,2,2)
        CALL DRAW3D(BEAM,2,2,2)
        CALL STOPOB
C       WRITE(7,540) LEN
C 540   FORMAT(' OBJECT 5 LENGTH=',I4)
        RETURN
550     CALL DRAWOB(OB5)
        RETURN
C
C OBJECT 6
C
600     IF(DONE) GO TO 650
        CALL MAKEOB(OB6,50,LEN)
        CALL DRAW2D(ROAD,4,0,2,0)
        CALL TXTURE(1)
        CALL DRAW2D(DLINE,2,0,2,0)
        CALL TXTURE(0)
        CALL STOPOB
C       WRITE(7,640) LEN
C 640   FORMAT(' OBJECT 6 LENGTH=',I4)
```

```
        RETURN
650     CALL DRAWOB(OB6)
        RETURN
C
C OBJECT 7
C
700     IF(DONE) GO TO 750
        CALL MAKEOB(OB7,20,LEN)
        CALL DRAW2D(WALK,4,0,2,0)
        CALL STOPOB
C       WRITE(7,740) LEN
C 740   FORMAT(' OBJECT 7 LENGTH=',I4)
        RETURN
750     CALL DRAWOB(OB7)
        RETURN
C
C OBJECT 8
C
800     IF(DONE) GO TO 850
        CALL MAKEOB(OB8,170,LEN)
        CALL DRAW2D(BOTTOM,5,2,2,0)
        CALL DRAW2D(MILL,5,2,2,-24576)
        CALL DRAW3D(MSIDE1,5,2,2)
        CALL DRAW3D(MSIDE2,5,2,2)
        CALL DRAW3D(CENTER,3,2,2)
        CALL TRAN(0,-16384,-24576)
        BLDPT=LEN
        CALL ROT(ANGLE,2)
        CALL DRAW3D(BLADE,11,2,2)
        CALL STOPOB
C       WRITE(7,840) LEN
C 840   FORMAT(' OBJECT 8 LENGTH=',I4)
        DONE=.TRUE.
        OLDCNT=COUNT
        RETURN
850     IF(COUNT.EQ.OLDCNT) GO TO 860
        OLDCNT=COUNT
        CALL GETROT(OB8(BLDPT+2),ANGLE,2)
860     CALL DRAWOB(OB8)
        RETURN
C
        END
```

CHAPTER SIX

6. THE PICTURE SYSTEM 2 GRAPHICS SOFTWARE PACKAGE

The Graphics Software Package furnished with PICTURE SYSTEM 2 consists of a set of FORTRAN-callable subroutines. These subroutines are written with the intent of providing a user with the full capabilities of the PICTURE SYSTEM 2 without the necessity of the user to interface, on a system level, with the PICTURE SYSTEM 2 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 2 Processor. Instead, the user merely "calls" a subroutine to perform a required graphical function (i.e., TRANslate, ROTate, SCALE, read TABLET information, display a CURSOR, display TEXT, etc.). This chapter is designed to provide an easy reference to each of the subroutines described in Chapter 4. The calling sequence specifications are detailed in alphabetical order, with a summary of the error codes and error conditions contained in Section 6.2.

6.1 THE GRAPHICS SUBROUTINES

The following describes in detail the subroutines which comprise the PICTURE SYSTEM 2 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 defined 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 homogenous 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 homogenous coordinate [IW] refer to Section 4.2.

ANALOG

The ANALOG subroutine is called to read the current value of the specified analog channel and return the relative amount that the channel has changed since the last time ANALOG was called to read that channel. This allows the values returned from a call to the subroutine for a given channel to be accumulated in a variable to be used for absolute positioning.

FORTRAN Calling Sequence:

```
CALL ANALOG(ICHANL,IVALUE)
```

where:

ICHANL is an integer which specifies the device channel number that is to be read. This value may be in the range 0-31, providing support for up to 4 sets of Control Dials.

IVALUE is an integer variable in which the value read from the specified channel is to be returned. This value is in the range of approximately +32700 and is the relative value that the channel has changed since this channel was last polled. IVALUE will be returned with a value of zero the first time ANALOG is called.

ERRORS:

42,0: Invalid number of arguments in the parameter list.

42,1: Invalid parameter value (ICHANL ≠ 0-31).

BLANKS

The BLANKS (BLANK Segment) subroutine is called to cause the specified segment to be blanked/unblanked for subsequent refreshes. Blanking a segment allows it to remain within the refresh buffer but not displayed.

FORTRAN Calling Sequence:

```
CALL BLANKS(N,STATE)
```

where:

N is a nonzero integer which specifies the segment "name" which is to be blanked/unblanked. If N is zero, then the default segment blank/unblank status is modified to that specified by the STATE parameter.

STATE is a logical value which specifies whether the segment is to be blanked or unblanked.

```
STATE = .TRUE. for blanked.
STATE = .FALSE. for unblanked.
```

ERRORS:

41,0: Invalid number of arguments in the parameter list.

41,1: Invalid segment "name". The specified segment does not currently exist.

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 reset the 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:

29,0: Invalid number of arguments in the parameter list.

29,1: Invalid parameter value.

8,0: PUSH error (Matrix Stack overflow). This indicates that the Matrix Stack requirements have exceeded the amount allocated by the user during the call to PSINIT.

9,0: POP error (Matrix Stack underflow). This indicates that the user has attempted to retrieve a matrix which had not been previously PUSHed onto the Matrix Stack.

BLINK

The BLINK subroutine is called to set the Picture Generator status such that all subsequent lines drawn will blink or will not blink, dependent upon 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:

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

CHARSZ

The CHARSZ subroutine is called to update the character size selection parameters used by the Character Generator when characters are displayed by the Picture Generator.

FORTRAN Calling Sequence:

```
CALL CHARSZ(ISIZE,ITILT)
```

where:

ISIZE is an integer which specifies the character size to be selected. Valid values for ISIZE are:

0 = .36 cm (.14 inches)
1 = .08 cm (.03 inches)
2 = .15 cm (.06 inches)
3 = .25 cm (.10 inches)
4 = .40 cm (.16 inches)
5 = .68 cm (.27 inches)
6 = 1.14 cm (.45 inches)
7 = 1.88 cm (.74 inches)

These sizes give the approximate height of a capital letter (A-Z,0-9) based upon a 28.6 x 28.6 cm (11.2 x 11.2 inch) screen viewing area. It should be noted that subscript and superscript characters are only available for ISIZE=2-7. Subscript or superscript character codes (30-33 octal) used when ISIZE=0 or 1 will result in an invalid character size selection.

ITILT is an integer which specifies the character orientation that is desired. Valid values for ITILT are:

ITILT = 0 for horizontal character orientation.
ITILT ≠ 0 for 90 degree counterclockwise character orientation.

ERRORS:

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

17,1: Invalid parameter value (ISIZE ≠ 0-7).

CLEARS

The CLEARS subroutine is called to initialize or re-initialize the display hardware and the segmentation software. Segmentation allows a user to divide his picture into segments which can be independently generated and updated. Each segment is referred to by an integer number (or "name"). The CLEARS subroutine initializes a user specified array which is used to maintain data pertaining to each segment.

FORTRAN Calling Sequence:

```
CALL CLEARS(IARRAY,ISIZE)
```

where:

IARRAY is an integer array ISIZE words in length that is used to contain the data pertaining to each segment.

ISIZE is an integer which specifies the size, in words, of IARRAY. ISIZE should be at least 3*(m+n)+6 where m is the maximum number of segments and n is the maximum number of segments which may be active during a refresh cycle. ISIZE must be less than or equal to 4064 (4096-32).

ERRORS:

38,0: Invalid number of arguments in the parameter list.

38,1: Invalid parameter value (ISIZE > 4064).

38,2: Invalid call to a segmentation routine. This error may be caused by a call to OPENS, CLOSES, BLANKS or DELETS without previously calling CLEARS.

38,3: Segmentation error. This error is generated when memory compaction attempts to write data past the write-back limit.

38,4: Segmentation error. This error is generated when a "name" is not found when searched for in the refresh buffer.

38,5: Refresh buffer overflow. This error occurs when there is no room for further memory compaction but more data pendin for output.

CLOSES

The CLOSES (CLOSES Segment) subroutine is called to cause all currently active segments to be closed and included for display during the next refresh cycle.

FORTRAN Calling Sequence:

```
CALL CLOSES
```

ERRORS:

None

COLOR

The COLOR subroutine is called to specify the color that all subsequent data drawn (or until the next COLOR specification) should be displayed. Color changes should be specified in the following order: red, orange, yellow-orange, yellow, green to ensure maximum display effectiveness. If the user attempts to change colors too often during a frame such that the color monitor is being overdriven, subsequent color changes will not occur.

Note: While this routine is provided for use specifically with the beam penetration color monitors, it may be used with a monochrome display to provide an increase in the intensity (and a decrease in the speed) with which lines and characters are drawn.

FORTRAN Calling Sequence:

```
CALL COLOR(IHUE)
```

where:

IHUE is an integer which specifies the color that all subsequent data are to be displayed.

0 = monochrome (deselects color writing rate)
1 = Red
2 = Orange
3 = Yellow-Orange
4 = Yellow
5 = Green

Colors may be specified in any combination or any order, but to achieve maximum effectiveness, specification should be from red to green.

ERRORS:

21,0: Invalid number of arguments in the parameter list.

21,1: Invalid parameter value (IHUE ≠ 0-5).

COSSIN

The COSSIN subroutine is called to compute a cosine and sine for the angle specified and return the values to the calling routine as a binary fraction (the form expected by the Picture Processor). COSSIN is useful for forming one's own rotation matrices for use in updating a Linear Display List.

FORTRAN Calling Sequence:

```
CALL COSSIN(IANGLE,ICOS,ISIN)
```

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.

ICOS is an integer variable in which the computed cosine will be returned.

ISIN is an integer variable in which the computed sine will be returned.

ERRORS:

30,0: Invalid number of arguments in the parameter list.

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 regardless 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:

24,0: Invalid number of arguments in the parameter list.

NOTE: If "IX", "IY" and "IPEN" are not specified in the parameter list, the default variables IX, IY and IPEN in the named COMMON block PSCOM will be used to specify the position of the cursor. With exception of the automatic cursor mode, the cursor is at all times displayed in the viewport that is specified by the values in the default variables, IVL, IVR, IVB, IVT, IHI, IYI in the PICTURE SYSTEM COMMON block PSCOM. If the cursor is to be displayed in other than the default viewport set by PSINIT, these variables must then be modified by the user to reflect the boundaries of the viewport in which the cursor is to be displayed. In automatic cursor mode, the cursor is always displayed within a full-screen viewport.

DELETS

The DELETS (DELETe Segment) subroutine is called to cause the segment specified to be deleted from the refresh buffer. Deleting a segment not only causes it not to be displayed but also frees up the area of refresh memory it occupied. It should be noted that the opening (OPENS) of an existing segment is an implied deletion when the new segment is closed.

FORTRAN Calling Sequence:

```
CALL DELETS(N)
```

where:

N is a nonzero integer which specifies the segment "name" to be deleted.

ERRORS:

40,0: Invalid number of arguments in the parameter list.

40,1: Invalid segment "name". The specified segment does not currently exist.

DOT

The DOT subroutine is called to draw a dot at the specified 2D relative X,Y coordinates or the 3D relative X,Y,Z coordinates from the current position.

FORTRAN Calling Sequence:

```
CALL DOT(IDX,IDY[,IDZ])
```

where:

IDX is an integer which specifies the delta X relative coordinate.

IDY is an integer which specifies the delta Y relative coordinate.

IDZ is an integer which, if specified, is the delta Z relative coordinate. If IDZ is not specified, the 3-space relative coordinate (IDX,IDY,0) is positioned to.

ERRORS:

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

DOTAT

The DOTAT subroutine is called to draw a dot at the 2D absolute X,Y coordinates or the 3D absolute X,Y,Z coordinates specified.

FORTRAN Calling Sequence:

```
CALL DOTAT(IX,IY[,IZ])
```

where:

IX is an integer which specifies the X absolute coordinate.

IY is an integer which specifies the Y absolute coordinate.

IZ is an integer which, if specified, is the Z absolute coordinate. If IZ is not specified, the 3-space point (IX,IY,0) is positioned to.

NOTE: DOTAT positions for the dot with the homogeneous coordinate (IW)=32767.

ERRORS:

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

DRAWOB

The DRAWOB subroutine is called to output a linear display list, previously prepared by the MAKEOB subroutine, to the Picture Processor. When this routine is called, the PICTURE SYSTEM is placed in a mode such that the entire command/data list is processed in a single DMA operation.

FORTRAN Calling Sequence:

```
CALL DRAWOB(IARRAY)
```

where:

IARRAY is a user-supplied array in which the linear display list previously accumulated by MAKEOB is located.

NOTE: If desired, DRAWOB may be called while MAKEOB mode is active. This will cause the linear display list stored in IARRAY to be incorporated into the object currently being constructed. In this case, the command and data array supplied with the call will be incorporated into the object currently being constructed.

ERRORS:

35,0: Invalid number of arguments in the parameter list.

35,1: Invalid array content.

35,2: Attempt to exceed matrix stack limits. All pushes and pops must be able to be performed in hardware, thereby limiting the maximum level of pushes or pops within a linear display list to 8.

DRAW2D

The DRAW2D subroutine is called to draw two-dimensional data coordinate points using the drawing mode specified in the parameter list. The points to be drawn are arranged in x,y pairs and displayed at the intensity dependent upon the IZ parameter and the intensity values specified for the hither and yon planes.

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.
3 = set base-offset-offset-offset, etc.
4 = offset 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*.

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:

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

14,1: Invalid parameter value. This error may be caused by:

INUM < 0.
IF1 ≠ 0-4.
IF2 ≠ 0-4.

---

*The maximum intensity is specified using the VWPORT subroutine.

DRAW3D

The DRAW3D subroutine is called to draw three-dimensional data coordinate points using the drawing mode specified in the parameter list. The points to be drawn are arranged in x,y,z triplets and displayed at the intensity dependent upon the z coordinates and the intensity 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 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.
3 = set base-offset-offset-offset, etc.
4 = offset 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:

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

15,1: Invalid parameter value. This error may be caused by:

INUM < 0.
IF1 ≠ 0-4.
IF2 ≠ 0-4.

DRAW4D

The DRAW4D subroutine is called to draw homogeneous coordinate data using the drawing mode specified in the parameter list. The points to be drawn are arranged as sets of x,y,z,w coordinates and displayed at the intensity dependent upon the scaled Z coordinates and the values specified for the intensity values specified for the hither and yon planes.

FORTRAN Calling Sequence:

```
CALL DRAW4D(IDATA,INUM,IF1,IF2)
```

where:

IDATA is an integer array (4*INUM words in length) which contains the x,y,z,w coordinate data 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 sets 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.
- 3= set base-offset-offset-offset-etc.
- 4= offset always.

ERRORS:

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

16,1: Invalid parameter value. This error may be caused by:

INUM < 0.
IF1 ≠ 0-4.
IF2 ≠ 0-4.

GETCHR

The GETCHR subroutine is called to return all characters that have been input since the last line terminator (i.e., Carriage Return, Line Feed or Form Feed). Multiple lines may be buffered internal to GETCHR and the user may continue to input characters until 80 characters have been input. If more than 80 characters are entered before a line terminator, all characters after the 80th will be ignored.

FORTRAN Calling Sequence:

```
CALL GETCHR(ICOUNT,IBUFF,ISTAT[,MAX])
```

where:

ICOUNT is an integer variable where the count of the number of characters in the user supplied buffer (IBUFF) will be returned. If ICOUNT = 0 then no characters have been input since the previously entered line terminator.

IBUFF is an integer array into which all characters which have been input, since initialization or the last line terminator, will be placed. This buffer should be 80 bytes in length (or MAX bytes in length, if the MAX parameter is specified). Note that the line terminator is not returned to the user buffer.

ISTAT is an integer variable which is set by GETCHR to indicate the status of the character string returned in the user supplied buffer.

```
ISTAT = 0  character string incomplete.
ISTAT = 1  character string complete.
```

MAX is an integer which, if specified, is the maximum number of characters to return in the user supplied buffer. If MAX is not specified, then up to 80 characters may be returned. MAX must be $\leq$ 80.

ERRORS:

43,0: Invalid number of arguments in the parameter list.

GETROT

The GETROT subroutine is called to build a rotation transformation based on the angle and axis of rotation specified in the parameter list. The transformation is then returned in the user-supplied 16-word matrix buffer, IARRAY.

FORTRAN Calling Sequence:

```
CALL GETROT(IARRAY,IANGLE,IAXIS)
```

where:

IARRAY is a 16-word array in which the 4x4 rotation transformation is to be returned.

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 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:

30,0: Invalid number of arguments in the parameter list.

30,1: Invalid argument specified for the axis of rotation.

GETSCL

The GETSCL 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 returned in the user-supplied 16-word matrix buffer, IARRAY.

FORTRAN Calling Sequence:

```
CALL GETSCL(IARRAY,ISX,ISY,ISZ[,IW])
```

where:

IARRAY	is a 16-word array in which the 4x4 scaling transformation is to be returned.
ISX	is an integer which specifies the X scale value.
ISY	is an integer which specifies the Y scale value.
ISZ	is an integer which specifies the Z scale value.
IW	is an integer which specifies the factor used to scale the scaling definition values. If the scale factor is omitted or given as zero, it is treated as 32767.

ERRORS:

32,0:	Invalid number of arguments in the parameter list.

GETTRN

The GETTRN 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 returned in the user-supplied 16-word matrix buffer, IARRAY.

FORTRAN Calling Sequence:

```
CALL GETTRN(IARRAY,ITX,ITY,ITZ[,IW])
```

where:

IARRAY	is a 16-word array in which the 4x4 translation transformation is to be returned.
ITX	is an integer which specifies the scaled X translation value.
ITY	is an integer ehich 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:

31,0:	Invalid number of arguments in the parameter list.

HITEST

The HITEST subroutine is called to determine if any data has passed within a prespecified 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, DRAW3D, etc.) 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 nonzero 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.

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 Controller.

ERRORS:

26,0: Invalid number of arguments in the parameter list.

HITWIN

The HITWIN subroutine is called to specify a window through which data will be drawn. 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:

25,0: 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 (POPed) 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:

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

ISPCHD

ISPCHD (Is Pen CHangeD?) is a FORTRAN-callable integer function subroutine which may be used to determine whether the status of the pen in relation to the last time this routine was called has changed. This facilitates the testing for pen transitions (i.e., up to down, down to up), a function often required in tablet interaction.

Typical FORTRAN Calling Sequence:

```
C
C WAIT FOR PEN TO BE LIFTED UP
C
100     IF(ISPCHD(IPEN).NE.3) GO TO 100
          .
          .
          .
  or
C
C GET CURRENT PEN STATUS
C
        IPSTAT=ISPCHD(IPEN)
          .
          .
          .
```

where:

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

ISPCHD = 0 if pen is up and was up last call.
ISPCHD = 1 if pen is down and was up last call.
ISPCHD = 2 if pen is down and was down last call.
ISPCHD = 3 if pen is up and was down last call.

ERRORS:

None

ISWSET

ISWSET (Is SWitch SET) is a FORTRAN-callable integer function subroutine which may be used to determaine whether a particular switch of the PICTURE SYSTEM 2 Function Switches is set. This function routine allows FORTRAN application programs which do not have the ability to perform bit testing to test the status of a switch.

Typical FORTRAN Calling Sequences:

```
C
C COLLECT ALL SWITCH SETTINGS
C
           DO 10 I=1,16
           IVALUE(I) = ISWSET(I-1)
10         CONTINUE
  or
C
C IS SWITCH "N" SET?
C
           IF(ISWSET(N).EQ.1) GO TO 100
C
C NO...IT'S NOT SET, JUST CONTINUE
C
C50        CONTINUE
```

where:

N is an integer which specifies the switch number that is to be tested.

N=0-15 for 1 set of Function Switches & Lights
N=0-31 for 2 sets of Function Switches & Lights
N=0-47 for 3 sets of Function Switches & Lights

ISWSET(N)=0 if the switch is not set.
ISWSET(N)=1 if the switch is set.

ERRORS:

37,0: Invalid number of arguments in the parameter list.
37,1: Invalid switch number (N ≠ 0-63).

LIGHTS

The LIGHTS subroutine is called to set a 16-bit value into a particular set of lights of the PICTURE SYSTEM 2 Function Switches & Lights.

FORTRAN Calling Sequence:

```
CALL LIGHTS(IVALUE[,ISET])
```

where:

IVALUE is an integer which specifies the 16-bit value to be placed into the lights.

ISET is an integer which specifies which set of PICTURE SYSTEM Function Switches & Lights is to be used. If the set number is omitted, Funcion Switch & Light set 1 is assumed.

ISET=1 for Function Switch & Light Set 1.
ISET=2 for Function Switch & Light Set 2.
ISET=3 for Function Switch & Light Set 3.
ISET=4 for Function Switch & Light Set 4.

ERRORS:

37,0: Invalid number of arguments in the parameter list.

37,1: Invalid parameter value (ISET ≠ 1-4).

LINE

The LINE subroutine is called to draw a line in the present line mode, specified during initialization or by a previous call to TXTURE or BLINK, from the current position to the 2D relative X,Y coordinates or the 3D relative X,Y,Z coordinates specified.

FORTRAN Calling Sequence:

```
CALL LINE(IDX,IDY[,IDZ])
```

where:

IDX is an integer which specifies the delta X relative coordinate.

IDY is an integer which specifies the delta Y relative coordinate.

IDZ is an integer which, if specified, is the delta Z relative coordinate. If IDZ is not specified, a line is drawn to the 3-space coordinate (IDX,IDY,0).

ERRORS:

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

LINETO

The LINETO subroutine is called to draw a line in the present line texture from the current position to the 2D absolute X,Y coordinates or the 3D absolute X,Y,Z coordinates specified.

FORTRAN Calling Sequence:

```
CALL LINETO(IX,IY[,IZ])
```

where:

IX is an integer which specifies the X absolute coordinate.

IY is an integer which specifies the Y absolute coordinate.

IZ is an integer which, if specified, is the Z absolute coordinate. If IZ is not specified, a line is drawn to the 3-space point (IX,IY,0).

NOTE: LINETO draws the line with the homogeneous coordinate (IW)=32767.

ERRORS:

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

MAKEOB

The MAKEOB subroutine is called to initiate a mode in which all commands and data directed to the Picture Processor are intercepted and accumulated in a user-supplied array in the form of a linear display list. The command and data accumulated in this array may later be output to the PICTURE SYSTEM as a unit (or object), thus saving preparation time and other overhead. Any of the Graphics Software distributed as part of the PICTURE SYSTEM 2 Graphics Software Package may be used in creating a linear display list.

FORTRAN Calling Sequence:

```
[EXTERNAL FULSUB]

CALL MAKEOB(IARRAY,MAX,LEN[,FULSUB])
```

where:

IARRAY is a user-supplied array (MAX words in length) into which the accumulated commands and data are buffered.

MAX is an integer which specifies the maximum length of the user-supplied buffer (IARRAY).

LEN is an integer variable where the number of buffer words actually used will be maintained. This variable is set to 1 when MAKEOB is called and again by each call to FULSUB (see below). It may be modified by the user (carefully).

FULSUB is a subroutine which, if specified, is to be called when IARRAY becomes full. If supplied, the calling sequence must be:

```
CALL FULSUB(IARRAY,LEN)
```

NOTE: As commands and data are collected in IARRAY, the maximum nesting of matrix pushes and pops is maintained in counters, and is included as part of the structure. In this way, it may be determined in advance if display of this object will cause a matrix stack overflow or underflow, since software extension of the 8-deep matrix stack is not possible during the processing of a linear display list.

The "LEN" parameter specified above may serve as an extremely valuable tool where the "IARRAY" buffer is large enough to contain the entire array in one piece. In this case, if the value of "LEN" is saved immediately preceding the call to any PICTURE SYSTEM 2 graphics subroutine, and the saved value incremented by one, it will serve as a FORTRAN subscript for "IARRAY" pointing to the generated command word. If it is again incremented, it then points to the object data.

ERRORS:

33,0: Invalid number of arguments in the parameter list.

33,1: Invalid parameter value (MAX $\leq$ 0).

33,2: Re-entrant call to MAKEOB.

33,3: Overflow of user-supplied buffer (IARRAY). This will occur only if a FULSUB subroutine was not specified.

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 values 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:

7,0: Invalid number of arguments in the parameter list.

MOVE

The MOVE subroutine is called to position to the specified 2D relative X,Y coordinates or the 3D relative X,Y,Z coordinates from the current position.

FORTRAN Calling Sequence:

```
CALL MOVE(IDX,IDY[,IDZ])
```

where:

IDX is an integer which specifies the delta X relative coordinate.

IDY is an integer which specifies the delta Y relative coordinate.

IDZ is an integer which, if specified, is the delta Z relative coordinate. If IDZ is not specified, the 3-space relative coordinate (IDX,IDY,0) is positioned to.

ERRORS:

11,0: Invalid number of arguments in the parameter list.

MOVETO

The MOVETO subroutine is called to position to the 2D absolute X,Y coordinates or the 3D absolute X,Y,Z coordinates specified.

FORTRAN Calling Sequence:

```
CALL MOVETO(IX,IY[,IZ])
```

where:

IX is an integer which specifies the X absolute coordinate.

IY is an integer which specifies the Y absolute coordinate.

IZ is an integerw which, if specified, is the Z absolute coordinate. If IZ is not specified, the 3-space point (IX,IY,0) is positioned to.

NOTE: MOVETO positions with the homogeneous coordinate (IW)=32767.

ERRORS:

11,0: 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 data switch does not occur until the appropriate refresh interval).

FORTRAN Calling Sequence:

```
CALL NUFRAM
```

ERRORS:

None

OPENS

The OPENS (OPEN Segment) subroutine is called to open an area of the refresh buffer for the creation of a new segment. All subsequent data output to Picture Memory will be placed in this segment until OPENS, CLOSES or DELETS is called. If a segment by the same name currently exists then that segment will be automatically deleted when this new segment is closed and included for display during the next refresh cycle.

FORTRAN Calling Sequence:

```
CALL OPENS(N)
```

where:

N is a nonzero integer which is to serve as the name for this segment.

ERRORS:

39,0: Invalid number of arguments in the parameter list.

39,1: Invalid segment "name". This may be caused by N = 0.

39,2: This segment has been opened again without ever having been explicitly closed.

39,3: Too many segments active for the size of the array specifed in the call to CLEARS.

POP

The POP subroutine is called to pop the top element of the Matrix Stack into the Picture Processor Transformation Matrix.

FORTRAN Calling Sequence:

```
CALL POP
```

ERRORS:

10,0: POP error (Matrix Stack underflow). This indicates that the user has attempted to retrieve a matrix which had not been previously PUSHed onto the Matrix Stack.

PSINIT

The PSINIT subroutine is called to initialize the PICTURE SYSTEM 2 hardware and software. The initialization process includes the following:

The PICTURE SYSTEM 2 interrupt handlers are connected and set to provide 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 Controller is set to double-buffer mode with an initial null frame. The Picture Generator status is initialized to solid line texture and to display characters of .68 cm (.27") character size, and horizontal character mode.

All Picture Displays (scopes 0-5) are selected for output.

FORTRAN Calling Sequence:

```
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:

N=Error Condition Code*256 + PICTURE SYSTEM Subroutine Identifier.

See Section 4.3 for a list of PICTURE SYSTEM subroutine identifiers and condition codes.

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 parameter 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 to be stacked exceeds eight--the number utilized 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, this argument will not be utilized.

IFMCNT is an integer variable which, if 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.

It should be noted that a named COMMON block is defined by the Graphics Software for default variable usage. The named COMMON block is defined as follows:

```
COMMON/PSCOM/ICLOCK,IFMCNT,IX,IY,IPEN,IVL,IVR,IVB
,IVT,IHI,IYI
```

where:

ICLOCK: is the default variable which is incremented if the user does not specify an 'ICLOCK' parameter in the call to PSINIT.

IFMCNT: is the default variable which is incremented each 1/120 second if the user does not specify an "IFMCNT" parameter in the call to PSINIT.

IX,IY,IPEN:
are the default parameters which are used by the TABLET and CURSOR routines if no parameters are specified by the user to those subroutines.

IVL,IVR,IVB,IVT,IHI,IYI:
specify the viewpoint boundaries within which the cursor is always displayed. Upon initialization by PSINIT, the variables are set to the default boundaries for the cursor, -2048 to +2047. These variables may be altered dynamically to cause a cursor to be displayed on any portion of the screen for nonautomatic cursor mode.

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.

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 ensure 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

PUSH

The PUSH subroutine is called to push the current Picture Processor Transformation Matrix onto the Matrix Stack.

FORTRAN Calling Sequence:

```
CALL PUSH
```

ERRORS:

9,0: PUSH error (Matrix Stack overflow). This indicates that the Matrix Stack requirement has exceeded the amount allocated.

ROT

The ROT subroutine is called to build a rotation transformation based on the angle and axis of 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* 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:

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

4,1: Invalid argument specified for the axis of rotation.

---

*PICTURE SYSTEM 2 software is designed for a left-handed coordinate system.

SCALE

The SCALE subroutine is called to build a scaling transformation based on the X, Y and Z scaling values 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 scale value.

ISY is an integer which specifies the Y scale value.

ISZ is an integer which specifies the Z scale 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:

6,0: Invalid number of arguments in the parameter list.

SCOPES

The SCOPES subroutine is called to select/deselect the Picture Displays to which output will be directed.

FORTRAN Calling Sequence:

```
CALL SCOPES(IVALUE)
```

where:

IVALUE is an integer which specifies which Picture Displays are to be selected/deselected. IVALUE is interpreted as a 6-bit binary value where each bit that is set will select the corresponding scope and each bit that is not set will deselect the corresponding scope. Thus, the value 1 will select scope 0; 2, scope 1; 4, scope 2; 8, scope 3; 16, scope 4; 32, scope 5. The values are additive so that 1+2+4+8+16+32=63 will select all scopes for display.

ERRORS:

22,0: Invalid number of arguments in the parameter list.

22,1: Invalid parameter value (IVALUE ≠ 0-63).

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 either mode, it may be reset at any time to the opposite mode. The user need only call this subroutine if the refresh buffer is to be 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 Controller is to be set to. Valid values for ISTAT are:

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

ERRORS:

28,0: Invalid number of arguments in the parameter list.

28,1: Invalid parameter value (ISTAT ≠ 1 or 2).

SETLIT

The SETLIT subroutine is called to set or clear an individual light of the PICTURE SYSTEM 2 Function Switches & Lights, dependent upon the parameters specified to the subroutine.

FORTRAN calling Sequence:

```
CALL SETLIT(N,ISTAT)
```

where:

N is an integer which specifies the light number that is to be set or cleared.

N=0-15 for 1 set of Function Switches & Lights.
N=0-31 for 2 sets of Function Switches & Lights.
N=0-47 for 3 sets of Function Switches & Lights.
N=0-63 for 4 sets of Function Switches & Lights.

ISTAT is an integer which specifies whether the light is to be set or cleared.

ISTAT=0 to clear an individual light.
ISTAT≠0 to set a light on.

ERRORS:

37,0: Invalid number of arguments in the parameter list.

37,1: Invalid light number (N ≠ 0-63).

STOPOB

The STOPOB subroutine is called to terminate the creation of a linear display list, initiated by a previous call to the MAKEOB subroutine. The termination of the MAKEOB mode causes the PICTURE SYSTEM 2 software to revert to the normal mode of operation so that all subsequent data "drawn" will be output to the Picture Processor. A call to the FULSUB subroutine will also be invoked if one was specified by the user in the call to MAKEOB.

FORTRAN Calling Sequence:

```
CALL STOPOB
```

ERRORS:

34,0: STOPOB called when MAKEOB not in process.

STOPWB

The STOPWB subroutine is called to terminate the write-back to memory mode of operation, initiated by a previous call to the WBTMEM subroutine. The termination of the WBTMEM mode causes the PICTURE SYSTEM 2 software to revert to the normal mode of operation so that all subsequent data output to the Picture Processor will be transformed and output to Picture Memory for display. A call to the FULSUB subroutine will also be invoked if one was specified by the user in the call to WBTMEM.

FORTRAN Calling Sequence:

```
CALL STOPWB
```

ERRORS:

36,4: STOPWB called when WBTMEM not in process.

SWITCH

The SWITCH subroutine is called to read a 16-bit value from a particular set of PICTURE SYSTEM 2 Function Switches and return that value in the parameter passed to the routine.

FORTRAN Calling Sequence:

```
CALL SWITCH(IVALUE[,ISET])
```

where:

IVALUE is an integer variable in which the 16-bit value read from the Function Switches is returned.

ISET is an integer which specifies which set of Function Switches is to be read. If the set number is omitted, Function Switch set 1 is assumed.

ISET=1 for Function Switch & Light Set 1.
ISET=2 for Function Switch & Light Set 2.
ISET=3 for Function Switch & Light Set 3.
ISET=4 for Function Switch & Light Set 4.

ERRORS:

37,0: Invalid number of arguments in the parameter list.

37,1: Invalid parameter value (ISET ≠ 1-4).

SYNCS

The SYNCS (SYNC Segments) subroutine is called to allow the synchronization of the update cycle with the refresh cycle when using segmented-buffer mode. SYNCS will not return to the user until all active segments have been included in the current refresh cycle and all deleted segment areas have been compacted.

FORTRAN Calling Sequence:

```
CALL SYNCS
```

ERRORS:

None

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 regardless of new frame update rate. It should be noted that once the pen information is updated with the pen down bit set (bit 4), 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.

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 4 will be set if the pen is down and bits 2-0 will be zero if the pen is within proximity of the tablet surface. If bit 4 of IPEN is set, IX and IY will not be updated when the pen is up.

NOTE: If "IX", "IY" and "IPEN" are not specified in the parameter list, the default variables IX, IY and

IPEN in the named COMMON block PSCOM will receive the tablet information.

ERRORS:

23,0: 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. PSINIT initializes the character status or it may be updated by calling the CHARSZ subroutine.

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 as in a FORTRAN DATA statement.

ERRORS:

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

18,1: Invalid parameter value (NCHARS < 0).

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:

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

TXTURE

The TXTURE subroutine is called to set the Picture Generator status such that all subsequent lines drawn will be in the mode selected by the value of the argument.

FORTRAN Calling Sequence:

```
CALL TXTURE(ISTAT)
```

where:

ISTAT is an integer which specifies the line mode to be selected:

ISTAT = 0 for solid lines.
ISTAT = 1 for lines consisting of short dashes.
ISTAT = 2 for lines consisting of medium-short dashes.
ISTAT = 3 for lines consisting of medium-long dashes.
ISTAT = 4 for lines consisting of long dashes.
ISTAT = 5 for lines consisting of long-short dashes (centerline).
ISTAT = 6 for lines consisting of long-short-short dashes.

ERRORS:

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

20,1: Invalid parameter value (ISTAT ≠ 0-6).

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:

2,0: Invalid number of arguments in the parameter list.

NOTE: The values of the IHI and IYI parameters are normally of the range 255 to 0. This is for software compatibility with THE PICTURE SYSTEM. These values are divided by 4 to yield the final values used in specifying the viewport intensities for PICTURE SYSTEM 2 (63-0).

WBTMEM

The WBTMEM (Write-Back To MEMory) subroutine is called to initiate a mode of operation whereby all data written out to PICTURE SYSTEM 2 are returned to the calling program in a user-supplied buffer, rather than being output to the refresh buffer. The calling program is then returned to and write-back continues, as long as there is space in the write-back buffer, until either the STOPWB or NUFRAM subroutine is called. When this occurs the write-back subsystem is restored to its status prior to the WBTMEM call.

FORTRAN Calling Sequence:

```
[EXTERNAL FULSUB]
CALL WBTMEM(IARRAY,MAX,LEN,ITYPE[,FULSUB])
```

where:

IARRAY is the user-supplied write-back buffer (MAX words in length) in which data written back to memory are stored.

MAX is an integer which specifies the maximum buffer size, in words.

LEN is an integer variable where the number of buffer words actually used is maintained. This variable is zeroed when WBTMEM is called, and must be zeroed again by FULSUB (see below). It may be modified dynamically by the user (carefully).

ITYPE is an integer which specifies the type of data to be written back:

1 = data transformed only.
2 = data transformed and clipped.
3 = data transformed, clipped and viewport mapped.

FULSUB is the name of an optional user-supplied subroutine which is to be called when the writeback buffer, IARRAY, becomes full. If supplied, the FORTRAN calling sequence will be:

```
CALL FULSUB(IARRAY,LEN)
```

When called, the user's FULSUB routine must empty the writeback buffer (IARRAY) and zero the LEN

parameter before returning from the subroutine.

The data written back to memory and stored into the user buffer has one of three formats, depending upon the value of ITYPE. The formats are:

FORMAT 1 (ITYPE = 1):

```
X-coordinate (1 word)
Y-coordinate (1 word)
Z-coordinate (1 word)
W-coordinate (1 word)
(repeat)
```

FORMAT 2 (ITYPE = 2):

```
Command Code (1 word):
  0 = MOVETO
  1 = DRAWTO
  2 = TEXT
  3 = STATUS DATA
  4 = DOT
 -1 = EOF (end of frame)

For Command Code = 0,1,4:
  X-coordinate (1 word)
  Y-coordinate (1 word)
  Z-coordinate (1 word)
  W-coordinate (1 word)
  (repeat)

For Command Code = 2:
  character count (1 byte)
  character 1     (1 byte)
  character 2     (1 byte)
    .
    .
    .
  character n     (1 byte)


Note:  If n is even, a null character
       (0) is appended to create an
       even total byte count.

For Command Code = 3:
  Picture Generator Status  (2 words)
  Not Used                  (1 word)
```

Note: See the PICTURE SYSTEM 2 Reference Manual, Section 2.4.3, for specific Status Word formats.

FORMAT 3 (ITYPE = 3):

```
Command Code (1 word):
  0 = MOVETO
  1 = DRAWTO
  2 = TEXT
  3 = STATUS DATA
  4 = DOT
 -1 = EOF (end of frame)

For Command Code = 0,1,4:
  X-coordinate (1 word)
  Y-coordinate (1 word)
  Z-coordinate (1 word)

For Command Code = 2:
  character count    (1 byte)
  character 1        (1 byte)
  character 2        (1 byte)
    .
    .
    .
  character n        (1 byte)
```

Note: If n is even, a null character (0) is appended to create an even total byte count.

```
For Command Code =3:
  Picture Generator Status  (2 words)
  Not Used                  (1 word)
```

Note: See the PICTURE SYSTEM 2 Reference Manual, Section 2.4.3, for specific Status Word formats.

No data follows the EOF code, as it always signifies the end of the frame, and hence, the end of this buffer.

For ITYPE = 1, TEXT and status operations will be ignored.

ERRORS:

36,0: Invalid number of arguments in the parameter list.

36,1: Invalid parameter value.

36,2: Re-entrant WBTMEM call.

36,3: User array overflow in writing back to memory.

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. For three-dimensional windowing, if this parameter equals IWH, the yon boundary is positioned at infinity on the side of 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 minus infinity, 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, or is given as zero, it is treated as 32767.

ERRORS:

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

6.2 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 the following four types of programming errors that will be detected by the Graphics Software Package:

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:

IERR*256 + 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.

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 y.

where:

- x is the error type that was encounterd.
- y indicates the subroutine in which the error was detected.

The subroutine numbers, error codes and causes of the error condition are summarized in Table 6.2-1.

NOTE: Unless the user's error subroutine is named PSERRS, the resultant executable program created will include the graphics error subroutine PSERRS.

TABLE 6.2-1
PICTURE SYSTEM 2 Error Codes

Subroutine	Error Codes and Meaning
0. System Error	0: PICTURE SYSTEM 2 not ready. This is an indication that the Direct I/O Interface has not become ready and has timed out. Check to make sure that the PICTURE SYSTEM is properly powered up. If the problem persists, it may be indicative of a system hardware failure.
1. PSINIT	0: Invalid number of parameters. 1: Invalid parameter value.
2. VWPORT	0: Invalid number of parameters.
3. WINDOW	0: Invalid number of parameters.
4. ROT	0: Invalid number of parameters. 1: Invalid parameter value.
5. TRAN	0: Invalid number of parameters.
6. SCALE	0: Invalid number of parameters.
7. MASTER	0: Invalid number of parameters.
8. INST	0: Invalid number of parameters.
9. PUSH	0: PUSH Error. Attempt to push more than 8 matrices onto the matrix stack or more than 8 plus the number allocated as matrix stack area in the call to PSINIT.
10. POP	0: POP Error. Attempt to retrieve a matrix which had not been previously PUSHed onto the matrix stack.
11. MOVETO,MOVE	0: Invalid number of parameters.
12. LINETO,LINE	0: Invalid number of parameters.
13. DOTAT,DOT	0: Invalid number of parameters.
14. DRAW2D	0: Invalid number of parameters. 1: Invalid parameter value.

15. DRAW3D	0: Invalid number of parameters. 1: Invalid parameter value.
16. DRAW4D	0: Invalid number of parameters. 1: Invalid parameter value.
17. CHARSZ	0: Invalid number of parameters. 1: Invalid parameter value.
18. TEXT	0: Invalid number of parameters. 1: Invalid parameter value.
19. BLINK	0: Invalid number of parameters.
20. TXTURE	0: Invalid number of parameters. 1: Invalid parameter value.
21. COLOR	0: Invalid number of parameters. 1: Invalid parameter value.
22. SCOPES	0: Invalid number of parameters. 1: Invalid parameter value.
23. TABLET	0: Invalid number of parameters.
24. CURSOR	0: Invalid number of parameters.
25. HITWIN	0: Invalid number of parameters.
26. HITEST	0: Invalid number of parameters.
27. NUFRAM	0: Directive Failure.
28. SETBUF	0: Invalid number of parameters. 1: Invalid parameter value.
29. BLDCON	0: Invalid number of parameters. 1: Invalid parameter value.
30. GETROT,COSSIN	0: Invalid number of parameters. 1: Invalid parameter value.
31. GETTRN	0: Invalid number of parameters.
32. GETSCL	0: Invalid number of parameters.
33. MAKEOB	0: Invalid number of parameters. 1: Invalid parameter value. 2: Re-entrant call to MAKEOB. 3: User array overflow in creation of object. 4: Illegal RSR value for P$DMA system subroutine.

34. STOPOB	0: Call to STOPOB without MAKEOB in progress.
35. DRAWOB	0: Invalid number of parameters. 1: Invalid parameter value. 2: Matrix stack overflow/underflow error.
36. WBTMEM	0: Invalid number of parameters. 1: Invalid parameter value. 2: Re-entrant call to WBTMEM. STOPWB or NUFRAM should be called between succesive calls to WBTMEM. 3: User array overflow in writing back to memory. 4: Call to STOPWB without WBTMEM in progress.
37. ISWSET,SWITCH, SETLIT,CLRLIT, LIGHTS	0: Invalid number of parameters. 1: Invalid parameter value.
38. CLEARS	0: Invalid number of parameters. 1: Invalid parameter value. 2: CLEARS has not been called. 3: Segmentation error. 4: Segmentation error. 5: Refresh buffer overflow.
39. OPENS	0: Invalid number of parameters. 1: Invalid segment name. 2: Segment reopened without being explictly closed. 3: Too many segments active.
40. DELETS	0: Invalid number of parameters. 1: Invalid segment name.
41. BLANKS	0: Invalid number of parameters. 1: Invalid segment name.
42. ANALOG	0: Invalid number of parameters. 1: Invalid argument specified in the parameter list.
43. GETCHR	0: Invalid number of parameters.

REFERENCES

1. "PICTURE SYSTEM 2/PDP-11 Reference Manual", 901130-100 NC
   Evans & Sutherland Computer Corporation, 1976

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

APPENDIX A

PICTURE SYSTEM 2 SPECIFICATIONS

PHYSICAL CHARACTERISTICS

Electronics Cabinet	-29.5" (75 cm) height -27.5" (70 cm) width -36.5" (93 cm) depth
Work Table (may be separated from the electronic cabinet)	-29.5" (75 cm) height -44.5" (113 cm) width -30" (76 cm) depth
Picture Display	-58" (147 cm) height -30" (76 cm) base diameter
Tablet Surface	-.75" (2 cm) height -15.5" (40 cm) width -15.5" (40 cm) depth
Tablet Electronics Cabinet	-7.25" (18 cm) height -9.5" (24 cm) width -11.25" (29 cm) depth
Combined Weight	-700 lbs. (318 kgm)

POWER REQUIREMENTS

Primary Power	-115 volts, 60 Hz, Single Phase -30 Amp -Two Wire + ground -Ground common with Picture Controller
Receptacle	-Hubbel 2610 or equivalent
Power Consumption	-2160 Watts
Heat Dissipation	-7400 BTU/Hr.

ENVIRONMENTAL REQUIREMENTS

Operating Temperature	-65°F to 80°F (18°C to 27°C)
Relative Humidity	-20% to 80%
Recommended Clearance	-3' all sides

PICTURE CONTROLLER INTERFACE	-Connection between PICTURE SYSTEM 2 and PDP-11. Requires one Hex Small Peripheral Slot in PDP-11 wired for non-processor request (NPR) and non-processor grant (NPG) operation.
Data Paths	-Direct Input/Output (DIO): Single-word transfer -Direct Memory Access (DMA): Block transfers -Concurrent Operation
Data Transmission	-Up to 500 feet between interface and PICTURE SYSTEM 2. -Differential Driver/Receivers.
Transfer Rate	-DMA to PICTURE SYSTEM 2 (not including CPU, UNIBUS, or Memory Access Overhead): (1.0 + .004D) usec/word, where "D" is distance in feet.
Interrupt Types	-DMA -System -Clock -Interactive Devices
Standard Register Assignments (UNIBUS Address) [Picture Data Bus Address]	-PICTURE SYSTEM Data (767660) -DIO PICTURE SYSTEM Address (767662) -DMA Word Count (767664) -DMA Bus Address (767666) -I/O Status (767670) -DMA PICTURE SYSTEM Address [177747] -DMA Passive Input Port [177770]
PICTURE DATA BUS	-Connecting link for all PICTURE SYSTEMS 2 components and devices.
Word Transfer Rate	-150 nsec (6.66 M wps)
Bus Arbitration	-Synchronous -Round Robin -8 Channels
Bus Word Size	-16 Bits

PICTURE PROCESSOR

-High-speed, micro-programmed, digital arithmetic processor, may be either an Active or Passive device.

Input/Output Registers

-MAP Active Output Limit [177750]
-MAP Active Output Address [177751]
-MAP Active Input Address [177752]
-MAP Status [177753]
-MAP Passive Output Port [177776]
-MAP Passive Input Port [177777]

Micro-Code

-88 Instruction Bits
-256 Instruction words
-150 nsec Cycle Time

Dimension Modes

-2-dimensional data requires two words of data to store x and y coordinate values.
-3-dimensional data requires three words of data to store x, y and z coordinate values.
-4-dimensional data (homogeneous) requires four words of data to store x, y, z and w coordinate values. W serves as a scale factor by which x, y, and z are scaled in order to provide a much larger effective dynamic range.

Data Modes

-ABSOLUTE: specifies coordinates as a displacement from the origin of the data space.
-RELATIVE: specifies coordinates as a displacement from the previous coordinate.
-SET BASE: specifies a coordinate to be loaded into the Base Register.
-OFFSET: specifies coordinates as a displacement from the contents of the Base Register.
-PASS FORMATTED: specifies data to be passed though the MAP unprocessed, but formatted on output for processing by the Line Generator.
-PASS CONDITIONAL: specifies data to be passed through the MAP unprocessed, but output only if the last point processed by the MAP was not clipped. Output data are identical to input data (unformatted).
-PASS: specifies data to be passed through the MAP unprocessed and output unformatted.

Drawing Modes	-MOVE (M): positions the vector point to the specified location. -DRAW (D): defines a line from the last specified coordinate to the current coordinate. -The repeat drawing sequences are: * M,D,M,D,...(unconnected lines) * D,M,D,M,...(unconnected lines) * M,D,D,D,...(connected lines) * D,D,D,D,...(connected lines) * M,M,M,M,...(used to process dots)
Output Format Modes	-DISPLAY: transformed, normalized, clipped, perspective divided, and viewport mapped data formatted into two 16-bit words for processing by the Picture Generator. -SIXTEEN BIT PRECISION: output data formatted into four, 16-bit words for writing fully transformed data back into memory for further processing by the Picture Controller. -TRANSFORM/NORMALIZE: output data formatted into four 16-bit words which contain the transformed and normalized x, y, z and w coordinates just processed. No windowing, clipping or viewport mapping takes place. -TRANSFORM ONLY: output data formatted into four 16-bit words which contain the transformed x, y, z and w coordinates just processed. No normalization, windowing, etc. takes place.
Word Length	-16-bit input. -24-bit internal. -12 or 16-bit x and y output. -6 or 16-bit z output.
Processor Registers	-256 words of which 224 may be used as a processor matrix stack containing up to 14 4x4 matrices.
Transformations	-TRANSLATION in any direction. -ROTATION about any axis. -SCALE with respect to any dimension. -MIRROR IMAGES about a plane. -TRANSFORMATIONS expressed as a 4x4 matrix.

Compound Transformations	-Hardware matrix concatenation. -Transformation matrix may be loaded from or stored into the data base residing in Picture Controller memory. -Push-down stack for storing 14 full transformation matrices with provision for continuing the stack in the Picture Controller memory.
Clipping	-Eliminates the portions of objects that are outside the program specified field of view. -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.
Instancing	-A method of defining structures once in the data base and replicating it several times in different positions, sizes and orientations. -Instancing performed to any level.
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 have been transformed, clipped, and put in perspective are 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 to give simultaneous use of several areas of the screen. -Specification of viewport front and back provides the intensity bounds for depth-cueing.

Zooming

-Allows for smooth and rapid motion into (or out of) a data structure in order to obtain a more detailed (or wide angle) view of a chosen region in the drawing space.

Hit Test

-Can detect whether any part of a given picture element is within a program-specified region in the data space. Used for implementing the pointing function.

Processing Speeds (From Picture Memory)

-Including complete MAP processing:

* Matrix Concatenation	31 usec
* Characters (4)	2.4 usec
* Status Word (32 bits)	2.4 usec
* Move or Dot	
Clipped (nonvisible)	8.7 usec
Not Clipped (visible)	11 usec
* Absolute Line Endpoint	
Nonvisible ——→	9.6 usec
Completely Visible ●——→	12 usec
Visible, 1 plane	
1 coordinate ——●——→	16.5 usec
Visible, 1 plane	
2 coordinates ——●——→	20.6 usec
Visible, 2 planes	
2 coordinates ——●——●→	24.8 usec

User Commands

-DRAW: 120 Draw mode combinations.
-HALT: stops the input operation.
-JUMP: transfer of control.
-PUSHJ: transfer of control with current address saved on stack.
-POPJ: return to saved address.
-LOAD: loads the 256 registers.
-STORE: stores the 256 registers.
-LOADSTK: loads data into the stack area of the processor registers.
-STORESTK: stores data from the stack area of the processor registers.
-XFER: transfers contents of the processor registers from one location to another or swaps them.
-PUSH: contents of processor registers loaded onto the stack.
-POP: top of stack loaded into processor registers.
-MATCON: multiplies input matrix by the current transformation matrix.

PICTURE MEMORY	-General purpose storage medium for processed and unprocessed data. Its primary purpose is to allow complete separation of display refresh and dynamic update requirements. A passive only device.
Memory Type	-Dual Port. -16 pin, 4Kx1 dynamic MOS IC's.
Word Size	-16 bits.
Memory Size	-16K, 32K, 48K, or 64K words.
Maximum Throughput	-1.33 M wps; 1 Active device. -2.22 M wps; 2 Active devices (1.11 M wps each). -2.50 M wps; 3 Active devices (.83 M wps each).
Memory Uses	-User Storage. -Untransformed data file. -Transformed display file. -Transformed data file.
Display File Content	-Dots and line endpoint data for use by the Picture Generator. One dot or line endpoint entry requires 2 words of memory: 12 bits for x, 12 bits for y, 6 bits for intensity, and 2 bits for command. -Packed character codes for use by the Character Generator. One entry requires 2 words of memory and contains 4 character codes. -Control information used to direct the operation of the Picture Generator. Each entry requires 2 words of memory.
Display File Size	-Up to 8K entries for each 16K memory module.

PICTURE GENERATOR

REFRESH CONTROLLER

-An Active device responsible for reading processed data and channelling it to the Line Generator for display on the Picture Display.

Refresh Rate

-First 16 multiples of 1/120 of a second (1/120, 1/60, 1/40, etc.).

Buffer Modes

-Single Buffer
-Double Buffer
-Segmented Buffer

Registers

-Current Segment Name	[177730]
-Segment Name	[177731]
-Active Writeback Address	[177732]
-Active Writeback Limit	[177733]
-Active Input Address	[177734]
-Active Start Address	[177735]
-Active Input Limit	[177736]
-Status Register	[177737]

Refresh Modes and Timing (per 2 16-bit words)

-Normal: 1.5 usec.
-Automatic: 1.5 usec.
-Memory Compaction: 1.8 usec.
-Name Search: 1.8 - 2.1 usec.
-Segment Skip: 1.5 usec.
-Jump: 3.15 usec.

Search Modes

-Next Name
-Left Byte Match
-Right Byte Match
-Word Match

CHARACTER GENERATOR

-Interprets character codes, producing strokes which are sent to the Line Generator for display.

Character Font

-128 ASCII Codes.
-95 displayable ASCII code subsets.
-Fast font allows up to 7000 characters per frame.
-Slow font has more detail for clarity.
-Positioning commands interpreted (i.e., Tab, Carrage Return, Line Feed, Backspace, etc.).
-Programmable left and top margins.
-256 user programmable codes.
-Programmable Inter-character and Inter-line spacing.

Character Memory	-1024 x 12 bit PROM (ASCII Font). -1024 x 12 bit RAM (User Programmable Font).
Character Sizes	-8 sizes standard from .03" to .74". -64 user programmable sizes and orientations. -Italics for all standard character sizes.
Character Orientation	-User Programmable. -Controlled by 2x2 matrix. -Superscripts for 6 of the standard character sizes. -Subscripts for 6 of the standard character sizes. -ROM storage for 16 sets of character coefficients. -RAM storage for 32 sets of character coefficients. -14-level Font Parameter Stack.
Character Integrity	-No character wrap-around. -Characters beyond any edge of the screen are not displayed.
Display Capacity (fast font, average character frequency)	-At 30 frames per second refresh rate on 11.23" x 11.23" display area.

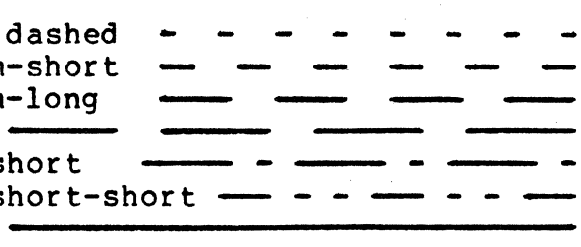
Point Size	Height	Characters Per Line	Number of Characters
4	.03"	436	7000
6	.06"	218	6200
10	.10"	131	6700
14	.14"	93	5600
17	.16"	81	5300
28	.27"	48	4300
46	.45"	29	4300
76	.74"	17	4300

LINE GENERATOR

-Converts digital end-point coordinates to analog signals for drawing on the Picture Display.

Line Modes

-Blink
-Short dashed
-Medium-short
-Medium-long
-Long
-Long-short
-Long-short-short
-Solid



-Endpoints drawn as dots.
-Continuous texture mode for dashed lines.

Color
(Optional Color Display)

-Green
-Yellow
-Orange
-Red-Orange
-Red

Display Select

-Up to 6 Picture Displays.

Intensity

-64 levels of constant intensity.
-Depth-cueing.

Drawing Speeds
(14"x14" viewing area)

-Line length (L) $\leq$ .155" = 1.55 usec.
-.155"$<$L$\leq$.218"= ($\overline{1}$.15 + 2.586L) usec.
-L $>$ .218" = (1.6 + 1.293L) usec.

Move Speed
(14"x14" viewing area)

-Move length (M) $\leq$ .696" = 1.55 usec.
-M $>$ .696" = ($\overline{1}$.1 + .647M) usec.

Dot Speed
(14"x14" viewing area)

-Dot spacing (D) $\leq$ .386" = 1.55 usec.
-D $>$ .386" = ($\overline{1}$.3 + .647D) usec.

Display Capacity

-At 30 frames per second refresh rate.

Length	Moves	Dots	Connected Lines
.1"	21,500	21,500	21,500
.2"	21,500	21,500	20,000
.5"	21,500	20,530	14,830
1"	19,080	17,120	11,520
2"	13,920	12,850	7,960
5"	7,690	7,350	4,130
6"	6,690	6,430	3,560
10"	4,400	4,290	2,290
12"	3,760	3,675	1,940
14"	3,280	3,215	1,690

PICTURE DISPLAY	-Cathode Ray Tube (CRT).
Display Type	-Calligraphic (random stroke).
Deflection Type	-Electromagnetic with phosphor and overdrive protection.
Addressable Locations	-4096 x 4096 x 64
Display Screen Size	-21" rectangular -16" x 13" opening. -10" x 10" quality viewing area.
Phosphor	-P4 standard. -Others available.
Controls	-On/Off -Focus -X and Y Gain -X and Y Position -Intensity -Brightness -Contrast (5 selectable values)
Spot Size and Endpoint Match	-Not exceeding 0.020" in quality viewing area.
Brightness	-10 foot-lamberts with 10" x 10" raster of 512 lines at 40 Hz frame with P4 phospher within specified spot size.
Jitter	-.005" maximum.
Contrast	-4:1 or greater.
Deflection Input Impedance	-Greater than 1K ohm.
Deflection Sensitivity	-1 inch per volt.
Video Input Impedance	-75 ohms.
Video Sensitivity	-0 to 1 Volts (1 volt = maximum intensity).
Unblank Input Impedance	-75 ohms.
Unblank Sensitivity	-0 to +2.5 Volts. (2.5 volts unblanked)
Cable Length	-7' standard. -15' optional.

DATA TABLET	-General purpose interactive input device.
Output	-11 bits of X. -11 bits of Y. -Pen proximity and up/down status.
Resolution	-100 lines per inch.
Proximity Range	-3/16"
Repetition Rate	-Continuously variable up to 200 coordinate pairs per second.
Size	-11" x 11" active area. -15" x 15" outside dimensions. -Others available.

APPENDIX B

B. USE OF THE PICTURE SYSTEM 2

The physical configuration of a stand-alone PICTURE SYSTEM 2 consists of the following parts:

1. The Picture Controller (Host Computer System)
2. PICTURE SYSTEM 2 Console Work Station
3. The Data Tablet
4. The Control Dials
5. The Function Switches & Lights
6. The Alphanumeric Keyboard

The following sections describe the use of each of these parts.

B.1 THE PICTURE CONTROLLER (Host Computer System)

The Digital Picture Equipment Corporation PDP-11 computer serves as the standard Controller for PICTURE SYSTEM 2. The PDP-11 is a stand-alone mini-computer system which is connected to PICTURE SYSTEM 2 by means of a Picture Controller Interface. This interface requires a Hex Small Peripheral Control mounting slot in the PDP-11 computer system that has been wired for non-processor request/non-processor grant (NPR/NPG) operation. For use and preventative maintenance procedures of the PDP-11, the user is referred to the Digital Equipment Corporation reference manuals for his particular PDP-11 installation.

B.2 PICTURE SYSTEM 2 CONSOLE WORK STATION

PICTURE SYSTEM 2 is enclosed in an attractive, functional work station consisting of a work table, and electronics cabinet and a Picture Display as shown in Figure B-1. The work table is 44.5 inches (113 cm) wide, 30 inches (76 cm) deep, and 29.5 inches (75 cm) high, which allows adequate room for interactive devices and user documentation. It is black in color and has been designed for the comfort of the user having no sharp edges or corners. The table legs have adjustable glides used for leveling the surface. The table weighs 90 lbs. (41 kg).

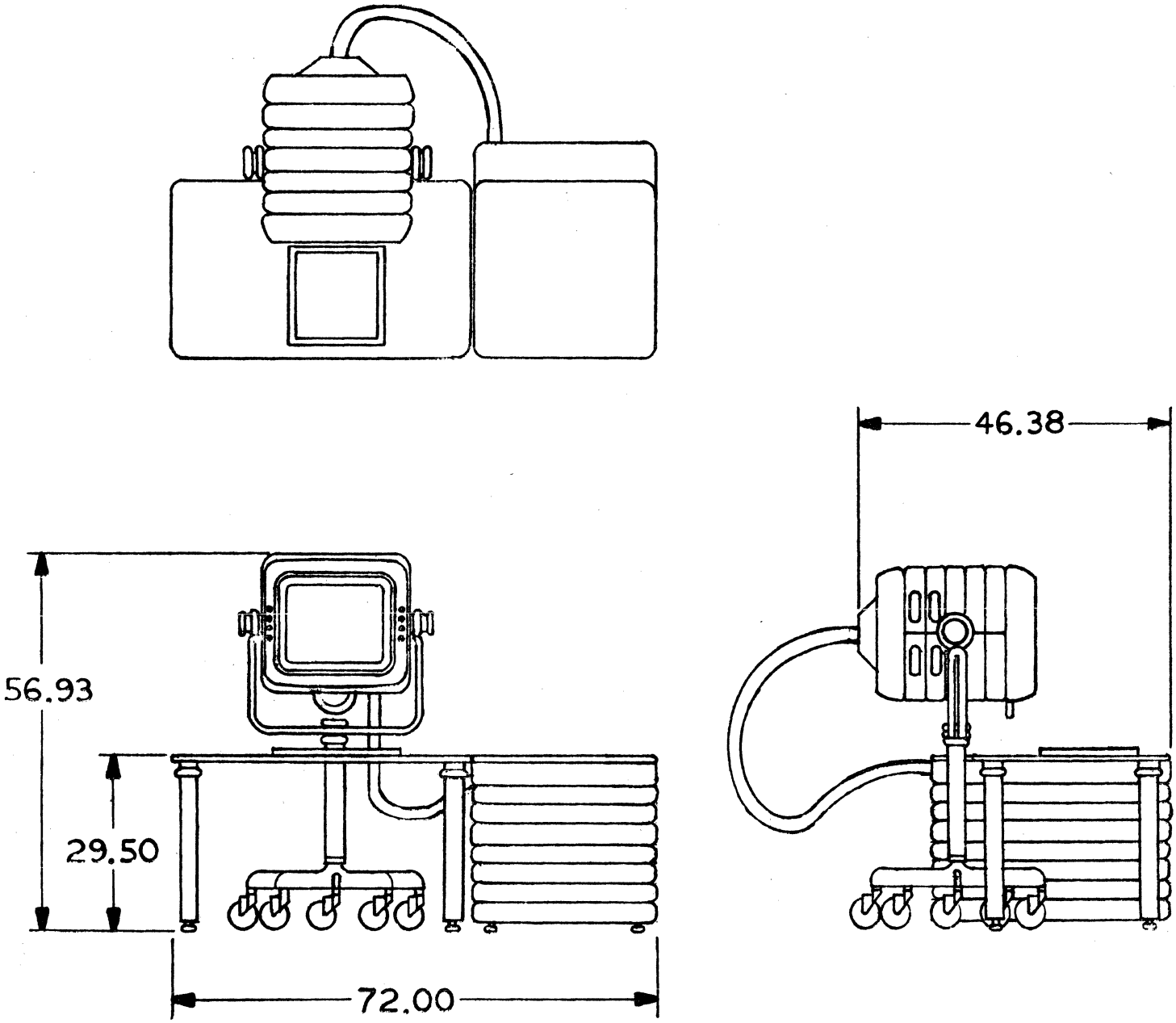


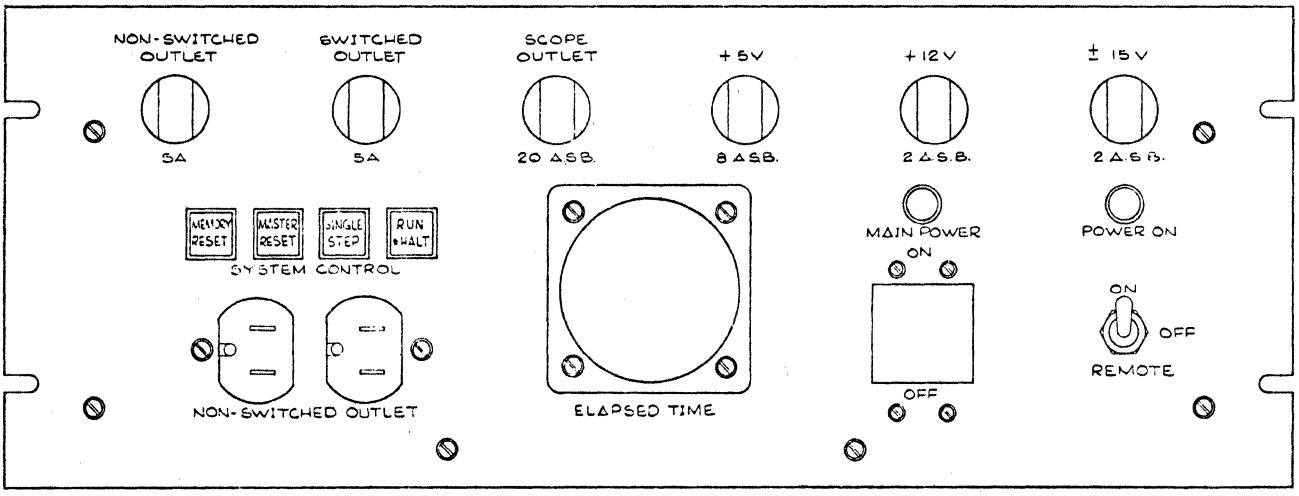
Figure B-1

PICUTRE SYSTEM 2 Work Station

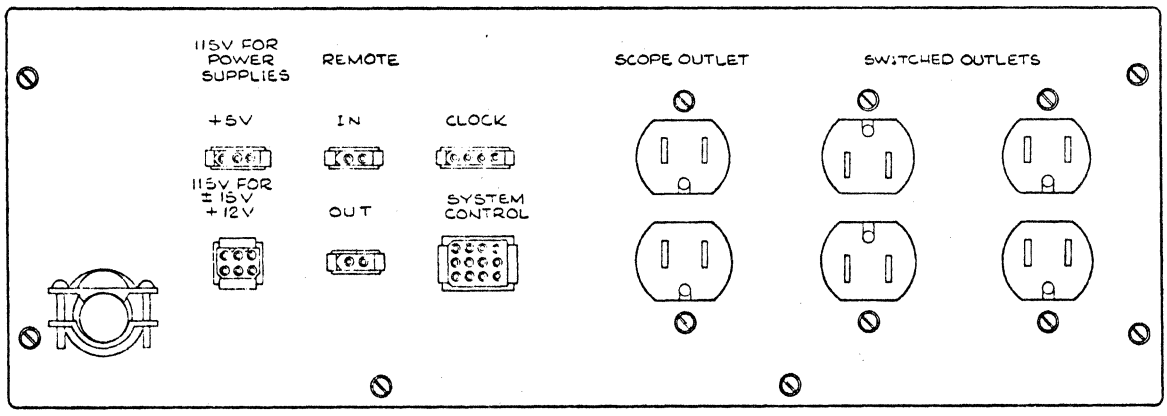
The PICTURE SYSTEM 2 electronics cabinet contains the electronic components, power supplies and power controller. This enclosure is 27.5 inches (70 cm) wide, 36.5 inches (93 cm) deep, and 29.5 inches (75 cm) high and includes a 27.5 inch (70 cm) wide by 30 inch (76 cm) black table top that matches the work table surface and provides additional space for interactive devices and documentation. The cabinet is on casters for mobility and has leveling glides for use in aligning its table top surface with the work table. The four sides of the cabinet are enclosed with removable black fiberglass panels and the table top is easily removed, allowing maximum exposure of the equipment for maintenance purposes. The panels are removed by pulling them straight out (away from the metal chassis) and then lifting them up and off the bottom supports. To reinstall the side panels, set the side panels down on the bottom supports, align the ball studs at the top of the fiberglass panel with the clips in the chassis, and push the ball studs into the clips. The table top is removed by unscrewing the four captive screws from the table top that are attached to the top rails of the chassis. The side panels must be removed to obtain access to these captive screws. The table top may be replaced by aligning the four spacers on the bottom of the table top with the four captive screws attached to the top rails of the chassis and tightening the screws into the spacers. The cabinet with its electronic components weights 245 lbs. (111 kg). Cooling air enters the electronic cabinet through the bottom of the left side panel, flows up through the components and exits through a .5 inch opening around the entire cabinet between the top of the side panels and the bottom of the table top. The air intake panel of this cabinet should not be blocked.

The PICTURE SYSTEM 2 electronic components consist of a 2 row backpanel and the various standard Evans & Sutherland component boards which plug directly into this backpanel. There are two 25 foot cables which link PICTURE SYSTEM 2 to the Picture Controller. Longer length cables are available.

Power is supplied to PICTURE SYSTEM 2 by a single phase, 60 Hz, 115 volt, 30 amp, 2 wire + ground electrical circuit with a Hubbell 2610 twist-lock receptacle (or equivalent). This power is distributed to the electronic components by means of three power supplies within the electronic cabinet. The power supplies located there are +5 volt, +12 volt and ±15 volt regulated power supplies. These voltages are distributed to the electronic components of PICTURE SYSTEM 2 through a power control panel. This power control panel, shown in Figure B-2, provides fused protection for the electronic components.



FRONT VIEW



REAR VIEW

Figure B-2

PICTURE SYSTEM 2 Power Control Panel

The front of the power control panel is equipped with several lighted indicators and switches, each of which are detailed below.

MAIN POWER — This switch is the primary power breaker switch for the PICTURE SYSTEM 2. This switch is normally set to the ON position during system installation and left in that position thereafter. The red light above the switch is lit to indicate that power is being supplied to the system. This breaker switch should not be used to power the system off and on.

POWER ON — This is a three position switch which is used to select power for the system. The three switch positions are:

ON-when set to this position, PICTURE SYSTEM 2 will be on whenever there is power from the external input.

OFF-when set to this position, no power is distributed from the power control panel (i.e., the system is off).

REMOTE-when set to this position, PICTURE SYSTEM 2 will be on whenever there is power from the external input and there is power supplied to the remote sensor plug. This plug is typically inserted into a 115 volt A.C. switched outlet on the host Picture Controller computer system, thus allowing power to be turned off to the entire system from a single switch.

The red POWER ON light associated with this switch is lit whenever the system is on.

FUSES — Six fuses are located along the top of the power control panel, providing visual status of the individual power supply subsystems. Each fuse resides in a cap which will be lit if the fuse has been blown. Fuses may be replaced by turning the system off and then pushing the blown fuse cap in and twisting one-quarter turn counterclockwise. The

fuse may then be removed by pulling out on the cap. To replace a fuse, remove the blown fuse and insert a new fuse of the proper amperage as marked on the Power Control panel (e.g., 2A S.B.).

SYSTEM CONTROL There are four switches provided for controlling the status of PICTURE SYSTEM 2:

MEMRY RESET- his switch is depressed to reset the memory control system. Depressing this switch leaves the contents of the Picture Memory in an undefined state.

MASTER RESET-This switch is depressed to reset PICTURE SYSTEM 2 to its initial power-up state. The contents of the Picture Memory are not modified by this action.

SNGL STEP-This switch is depressed to cause a single system clock pulse to be issued when the PICTURE SYSTEM is in the HALT state (see below).

RUN/*HALT-This switch is a back lighted two position switch that selects the operating mode the system is to run in. This switch is normally depressed (and lit, RUN) selecting the system to free run mode. This switch is set to the out position (*HALT) only for specific system maintenance operations.

ELAPSED TIME This meter is used to record the number of hours (to the 1/10 hour) that the system has been powered on.

There are also several outlets available on the front and back of the power control panel:

NON-SWITCHED OUTLET
This outlet on the front of the power control panel provides power whenever MAIN POWER is on. This provides 115 volt A.C. power to be used for miscellaneous non-switched purposes such as maintenance, etc.

SWITCHED OUTLETS

These outlets on the back of the power control panel provide 115 volt A.C. power for devices to be used in conjunction with the PICTURE SYSTEM 2. Units such as the Data Tablet Controller are plugged into these outlets.

SCOPE OUTLET

This switched outlet on the back of the power control panel is provided specifically for the Picture Display. All Picture Displays should be plugged into the SCOPE OUTLET (or equivalent).

The normal status of the power control panel is:

MAIN POWER - Light on.
POWER ON - Light on.
RUN/*HALT - Light on.
FUSES - All lights off.

If the visual status of the power control panel is not that listed above or if the system is not ready (as indicated by an error message on the user's terminal), the following check list may be referenced in an attempt to isolate a basic system failure.

1. No Lights on.
   a. PICTURE SYSTEM 2 is not properly plugged in or external power is not being delivered to the system. Take appropiate action.
   b. MAIN POWER switch OFF. Push circut breaker switch up to ON position.

2. MAIN POWER light on, all other lights off.
   a. POWER ON switch is off. Set the switch to ON or REMOTE.
   b. POWER ON switch is set to REMOTE. Turn on remote host computer power or set POWER ON switch to ON.

3. MAIN POWER light on, POWER ON on, all other lights off.
   a. RUN/*HALT switch is in HALT position. Depress switch to RUN position.
   b. RUN/*HALT light is burned out. Replace the light.

4. MAIN POWER light off, all other lights on.
   a. MAIN POWER light is burned out. Replace the light.

5. POWER ON light off, all other lights on.
   a. POWER ON light is burned out. Replace the light.

6. Fuse light(s) on.
   a. Fuse is blown. Rectify problem causing the fuse to blow and replace fuse.

It should be noted that PICTURE SYSTEM 2 may be turned on and off without affecting the status of the host Picture Controller. This allows maintenance to be performed upon PICTURE SYSTEM 2 without interferring with the normal usage of the Picture Controller (other than that required for diagnostic support).

The Picture Display is supported by a pedestal so that it can be positioned about the work station at the users convenience. The cable between the Picture Display and the electronics cabinet comes standard as seven usable feet. The base of the pedestal requires a 30 inch (76 cm) diameter circle of floor space and has five, 4 inch (10 cm) locking casters. These casters provide stability when locked and easy movement when not locked. When mounted on the pedestal, the top of the Picture Display is 58 inches (148 cm) above the floor. The display rests in the yoke of the pedestal and is able to rotate within the yoke and be held in any desired position by locking knobs on both sides of the display. The Picture Display is enclosed with black fiberglass panels, matching the electronics cabinet, that may be easily removed for maintenance purposes. These panels are removed in the following order:

CAUTION

The Picture Display contains an 18,000 volt power supply. The display should be disconnected from the 115 volt source before any panels are removed.

1. Top Panel:
   Remove the four screws, two on each side at the front and rear, and lift the panel staight up keeping the sides of the top panel spread apart until it clears the top of the display chassis.

2. Bottom Panel:
   Remove the four screws, two on each side at the front and rear, and move the panel down and to the rear of the display keeping the sides of the bottom panel spread apart until it clears the display chassis.

3. Cable Receptacle:
   Loosen the two captive screws on the cable receptacle, slide the cable receptacle and cable hose

carefully away from the rear panel. Disconnect the four BNC connectors, the Molex connector, and the power cord.

4. Rear Panel:
Loosen the captive screw at the bottom of the rear panel. Pull the bottom of the rear panel towards the rear of the display until it clears the chassis and then lift the panel straight up and off the bracket attached to the top of the chassis.

5. Front Panel:
Remove the large part of the knobs on the front of the display that control "ON/OFF" and "CONTRAST". Remove the four hex-head bolts, two at the top and two at the bottom, from the inside of the front panel. Pull the front panel straight out from the chassis until it clears the control knobs.

To install the display panels, follow the reverse procedure for removing them.

Cooling air enters the left side of the Picture Display and exits through the right side. The Picture Display with its pedestal weighs 240 lbs. (110 kg). The Picture Display has a 16 inch (40.6 cm) wide by 13 inch (33 cm) high opening for the face of the CRT. Figure B-3 shows the face of the Picture Display and the control knobs used for adjusting the displayed image. The controls provided are:

X GAIN	Controls the horizontal width of the area scanned by the electron beam. This is normally adjusted so that the maximum width is approximately 10 inches.
X POSITION	Controls the horizontal position of the display area. This is normally adjusted so that the display area is centered in the front opening of the Picture Display.
Y GAIN	Same as of X GAIN except for the vertical axis.
Y POSITION	Same as for X POSITION except for the vertical axis.

BRIGHTNESS	Controls the brightness at which lines and characters are displayed. BRIGHTNESS should be used to adjust the intrinsic brightness of the image since it affects all lines equally regardless of their length or the speed with which they are drawn.
INTENSITY	Controls the maximum intensity range of all images displayed. This control should be set to the full clockwise position and the BRIGHTNESS control adjusted to obtain desired brightness. This control is used to decrease the intensity if an image which may burn the CRT is displayed.
FOCUS	Controls the clarity of the images displayed. This is normally adjusted to minimize the width of lines over the entire viewing area.
CONTRAST	Controls the ratio of light between the darkest and brightest part of an image. This control is a five position knob designed to provide correct contrast for varying room lighting conditions. The knob is set at the full counterclockwise position when no intensity contrast (i.e., depth-cueing) is to be observed. The remaining four positions of the knob are for progressively dimmer background lighting conditions. If depth-cueing is to be observed, then this knob should be adjusted so that the dimmest lines can barely be observed.
ON/OFF	Controls the power to the Picture Display. This switch is normally in the clockwise (ON) position. The Picture Display is usually powered from the SCOPE OUTLET, which provides power to the display only when the PICTURE SYSTEM is on.

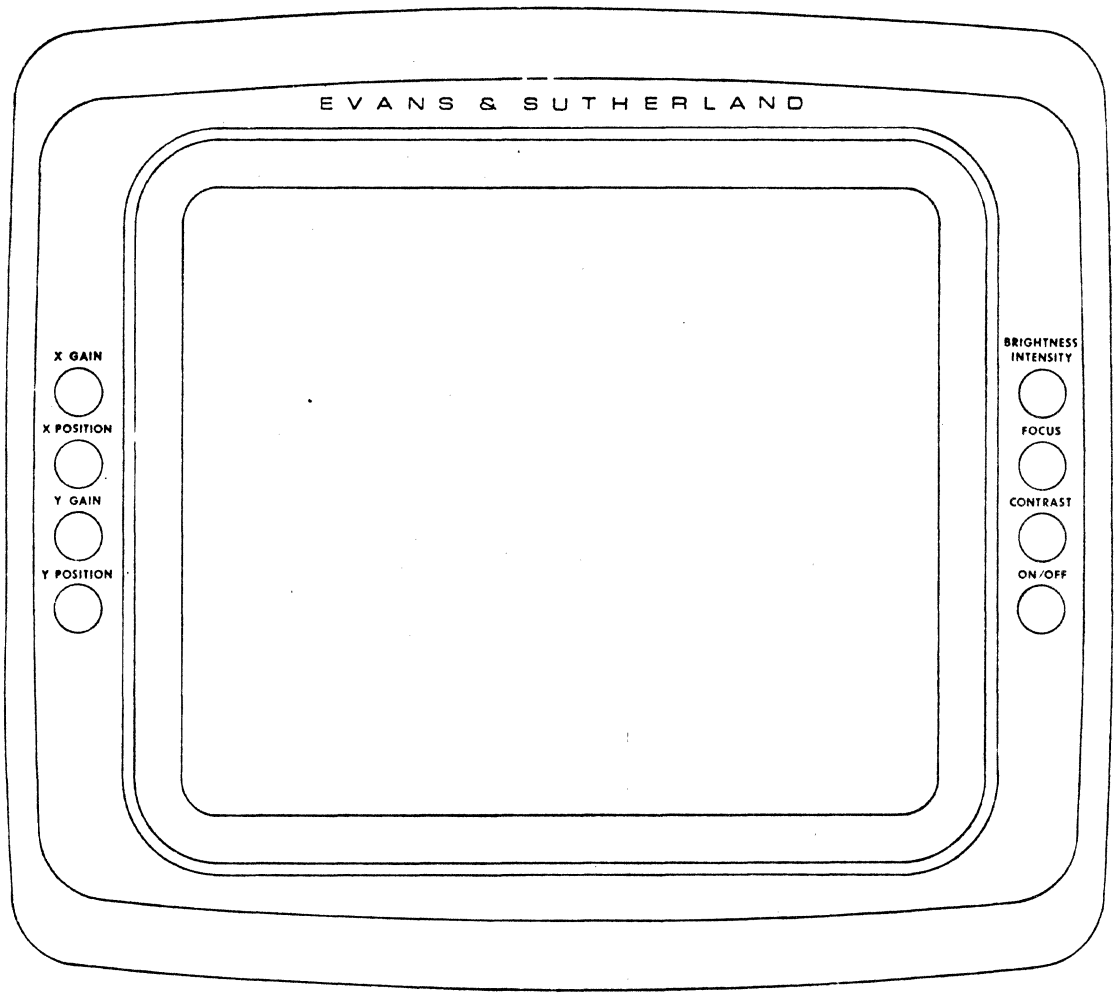


Figure B-3

Picture Display Control Knobs

Since the viewing of computer generated graphics images is the primary use of PICTURE SYSTEM 2, the display should be used very carefully to ensure quality pictures for the life of the system. In particular, the user should avoid displaying data which appears as a bright spot on the display. If a bright spot is allowed to remain on the screen for a period of time the phosphor on the inside face of the CRT may actually be "burnt" away by the electron beam leaving a burned spot on the CRT. The following procedure should be followed by all those who use PICTURE SYSTEM 2 to minimize the chance of burning the Picture Display.

1. Before turning the computer system off, turn down the intensity of the display and then turn off the display.

2. Before turning on the display, make sure the intensity is turned down (full counterclockwise).

3. After turning the computer system on, turn on the display and then turn up the intensity of the display.

4. Never leave a "bright" image on the display for a prolonged period of time.

5. If a very bright image ever appears on the display, immediately turn down the intensity and turn off the display.

6. Never leave the display unattended for very long. If you're going to be away for a while, turn the intensity down.

7. Log any problems observed while using the system and any burns noticed on the display.

B.3 THE DATA TABLET

The standard Data Tablet for PICTURE SYSTEM 2 is a Summagraphics tablet with an 11" x 11" active area, stylus, controller and interface to the Picture Data Bus. These components are described in the following sections:

B.3.1 The Tablet

The tablet serves as the digitizing surface on which graphic material may be placed or on which the stylus may be moved to input data to PICTURE SYSTEM 2. The outside dimensions for the tablet are 15.5 inches (40 cm) wide by 15.5 inches (40 cm) deep by 0.75 inches (2 cm) thick and it weighs 5.25 lbs. (2.39 kg). For clarity, digitization is described here with the system in point mode; that is, one pair of X and Y coordinates is generated each time the stylus ball point is pressed against a desired position on the tablet.

Upon stylus press-down, a strain-wave is injected from an X-send wire along one edge of the tablet into a set of magnetostrictive delay lines in the X-direction toward the stylus. At the time of injection, a crystal oscillator (clock) is gated into a counter. When the strain-wave passes under the receive coil stylus a pulse is generated wich stops the counting. The number of counts represents the X-coordinate. Directly thereafter, similar events are sequenced in the Y-direction. Suitable intervals are provided in this sequence of digitization to provide for margins on the tablet and accurate, repeatable coordinate sets.

The speed of the strain wave in the magnetostrictive delay line is nominally 5000 m/sec, or inversely 200 nsec/mm, or 50 nsec/.25 mm = 0.010". Therefore, the system 20 MHz clocking frequency provides a resolution of 100 lines/inch.

The tablet has been carefully aligned to assure the two axes are perpendicular. It is unlikely realignment will ever be necessary. Should you want to do so, however, it is a simple matter. Remove the top frame by removing the four screws in the bottom of the tablet. The vinyl cover will now lift off.

The perpendicular alignment of the two axes requires only that the two "send" wires along the edges of the tablet be straight and perpendicular. Inspect the wires for good ten-

sion. Note that a slight turn of the alignment screw on the "X" send wire will give the necessary adjustment. If the wires need to be tightened, this can be done by loosening the teflon screws, pulling on the wire and retightening the screw.

Before the alignment adjustment is made, the vinyl and the metal frames should be replaced. This is necessary because part of the magnetic bias is derived from the magnets attached to the frame. It is not necessary to reinsert the screws in the bottom for this step. Access to the adjustment screw is through the slot in the side of the metal frame.

One simple way to proceed is to use an accurate right, 45 degree triangle on the tablet surface. Simply digitize the three corners, record the X and Y coordinates, and determine that $\Delta X = -\Delta Y$, as shown in Figure B-4. This should be repeated several times to average out the human error and the $\pm 1/2$ count uncertainty.

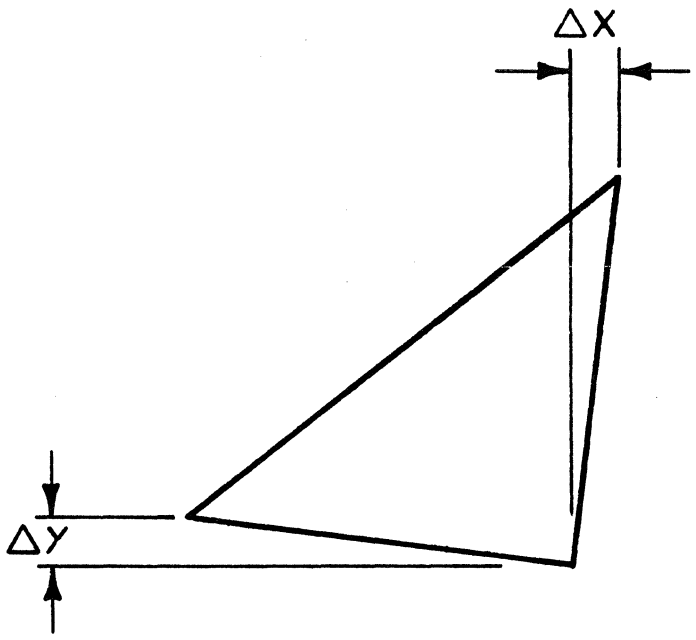


Figure B-4

Tablet Adjustment Measurement

NOTE: If a magnet or magnetized article is placed on the tablet the remnant flux in the magnetostrictive substrate could be altered. This would affect the accuracy of the digitizing process. If this should happen, the tablet is re-magnetized by following a tablet wipe procedure using the flexible magnet supplied with the system. Holding the magnet out straight with one hand at each end and with the arrowhead facing down and toward yourself, sweep the magnet smoothly and diagonally from the upper left hand corner to the lower right hand corner, as shown in Figure B-5. Under normal environmental and usage conditions it will not be necessary to refresh the magnetic bias. However, if any inaccuracies occur, it should be suspected that some magnetized material was left on the tablet and the procedure described here initiated.

Store the magnet well away from the tablet and magnetic disks and tapes.

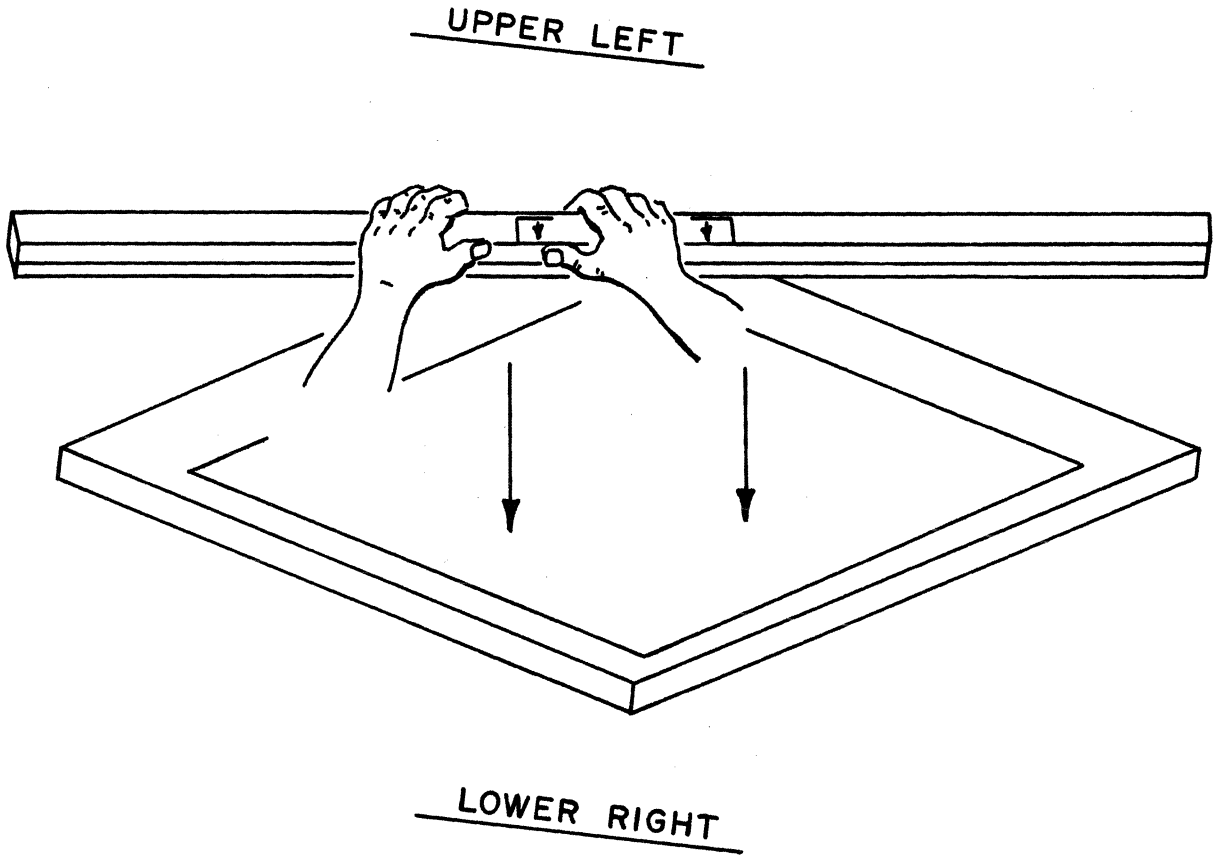


Figure B-5

TABLET WIPING INSTRUCTIONS: Hold magnet as shown in drawing. (Arrows on label must point forward with steel side of assembly up.) Hold magnet firmly against the surface of tablet. Wipe diagonally from upper left to lower right. Wiping must be done slowly in one continuous motion. DO NOT STORE MAGNET NEAR TABLET.

B.3.2 The Stylus

The stylus is a pen shaped device measuring 6 inches (15 cm) long and 0.5 inches (1.3 cm) in diameter. The stylus with its ball point is positioned on the point to be digitized. The stylus has a built in switch actuated by pressing the ball point against the tablet to start digitizing in point or stream switch mode. This switch is bypassed in other modes. The stylus, by its connection to the controller, starts and stops the X and Y count during the digitization process. The stylus may be noninking, or inking with a choice of colors available. Both noninking and blue cartridges are provided with the tablet.

The stylus has been designed to allow the user to replace the ball point refill (when it becomes dry or to change color). To change or replace the refill simply:

1. Unscrew front section of stylus.
2. Grasp internal front section and carefully pull it straight off.
3. Pull out refill and replace with a new one.
4. Replace front inner section
5. Replace outer cover.

B.3.3 The Tablet Controller

The Tablet Controller supplies all operating voltages and currents, provides the controls for choice of operating modes and choice of rate of coordinate pair production, provides a clock, X and Y counters and timing sequences. It is enclosed in a separate table top enclosure measuring 9.5 inches (24 cm) wide by 11.25 inches (29 cm) deep by 7.25 inches (18 cm) high which weights 13.5 lbs. (6 kg). Upon demand from the stylus the controller will forward an X and Y sense pulse to the tablet, start the counting and upon command from the stylus, will stop the counting, and transmit in binary the X and Y coordinates of the stylus position.

Figure B-6 shows the front view of the tablet controller. As this figure shows, these are several operating modes and status indicators. These are described as follows:

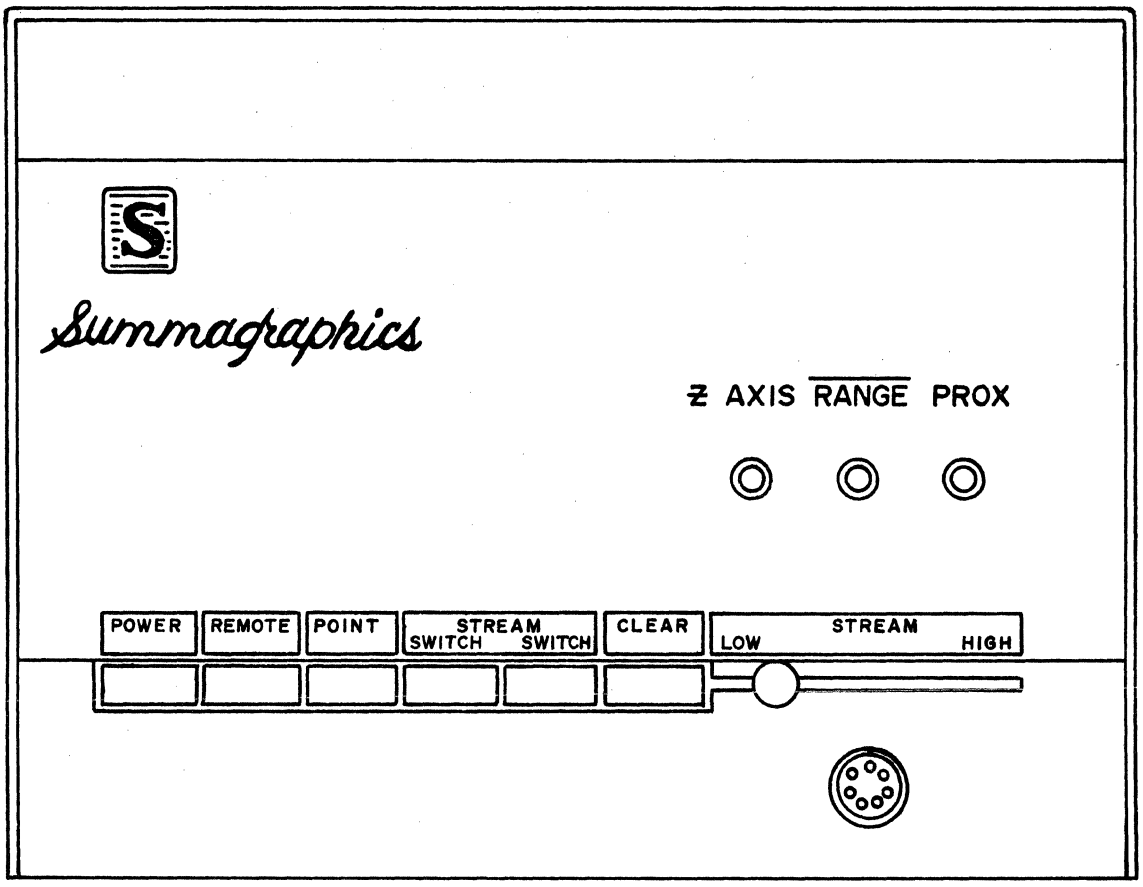


Figure B-6

Tablet Controller - Front View

CONTROLS

Push button selection of Point, Stream Switches Stream, or Remote modes. Potentiometer selection of repetition rate.

Point-Single Coordinate pair available by pen touch-down.

Stream-Coordinate pairs updated continuously (normal PICTURE SYSTEM operating mode).

Switch Stream-Coordinate pairs updated continuously while pen is down. Pen-switch activating pressure nominally 60 grams; other pressures available.

Remote-External signal is used to update coordinates.

INDICATORS

Range-Lights when stylus off active area.

Z Axis-Lights when stylus depressed.

Proximity-Lights when stylus is within 3/16" of tablet surface.

A seven usable foot cable comes standard connecting the Data Tablet with its interface located in the electronics cabinet which allows the tablet to be placed anywhere on the PICTURE SYSTEM work surface. See Section 4.13.1 for details on the Graphics Software Package support for the Data Tablet. See Reference 1, Section 2.6.2 for details of the Data Tablet Interface registers.

B.4 THE CONTROL DIALS

The analog Control Dials are available as interactive data input devices which are interfaced to the Picture Data Bus by the Evans & Sutherland analog-to-digital converter module. This module has sixteen channels and can support one or two Controls Dial units, each containing eight control dials. The Control Dial unit is contained in a black table top enclosure measuring 14.0 inches (36 cm) wide by 6.0 inches (15 cm) deep by 3.5 inches (9 cm) high. It has two rows of four continuous turn knobs and weighs 5.5 lbs. (2.5 kg). The cable between the Control Dial unit and the electronics cabinet comes standard as seven usable feet allowing the Control Dials to be placed anywhere on the PICTURE SYSTEM work surface. See Section 4.13.2 for details on the Graphics Software Package support for the Control Dials option. See Reference 1, Section 2.6.5 for details of the analog-to-digital converter interface registers.

B.5 THE FUNCTION SWITCHES AND LIGHTS

The Function Switches and Lights are available as interactive data input and data display devices for PICTURE SYSTEM 2. Each set of Function Switches and Lights has sixteen white toggle switches and sixteen white lights and is interfaced to the Picture Data Bus. Up to four sets may be attached to a PICTURE SYSTEM.

The Function Switches and Lights option is contained in a black table top enclosure measuring 14.0 inches (36 cm) wide by 6.0 inches (15 cm) deep by 3.5 inches (9 cm) high and weighs 5.5 lbs. (2.5 kg). Paper overlays are included with this option so that the user may define switch and light assignments for specific program functions. A clear plastic cover keeps the overlays in place. A seven usable foot cable comes standard connecting the Function Switch and Light unit with its interface in the electronics cabinet. This allows this unit to be located anywhere on the PICTURE SYSTEM work surface. See Section 4.13.3 for details on the Graphics Software Package support for this option. See Reference 1, Section 2.6.4 for details of the Function Switch and Lights interface registers.

B.6 THE ALPHANUMERIC KEYBOARD

The Alphanumeric Keyboard is contained in a table top enclosure measuring 18 inches (43 cm) wide by 9 inches (23 cm) deep by 4.5 inches (11 cm) high which weighs 9.5 lbs. (4.3 kg). It has 61 keys with unshift/shift and control keys, allowing a total of 128 character codes to be entered.

The keyboard generates an interrupt request to the Picture Controller whenever a key is struck, whereupon the user program may read and act upon the code for the key struck.

The cable between the Alphanumeric Keyboard and its interface which resides in the electronics cabinet, comes standard as seven usable feet allowing the keyboard to be placed anywhere on the PICTURE SYSTEM work surface. See Section 4.13.4 for details on the Graphics Software Package support for the Alphanumeric Keyboard option. See Reference 1, Section 2.6.3 for details of the Keyboard interface registers.

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z