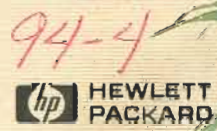


HP 2700 Family

High Performance Color Graphics Terminal



color graphics reference manual



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Color Graphics Reference Manual



HEWLETT-PACKARD COMPANY
974 EAST ARQUES AVENUE, SUNNYVALE, CALIFORNIA 94086

Table of Contents

How To Use This Manual	i	Errors	1-18
Manual Sections And Appendices	i	Major Sequence Errors	1-18
HP 2700 Series Terminal Documenta- tion List	ii	Illegal Parameter Errors	1-18
Features And Accessories	iii	Non-Parameter Errors	1-18
Graphics Keypad	iv	Execution Errors	1-20
Compatibility With Other Hewlett-Packard Graphics Terminals	iv	Exceptions	1-20
		Collecting and Reporting Errors	1-20
		Summary	1-21
Section 1: Graphics Concepts			
Introduction	1-1	Section 2: Drawing Fundamentals	
Picture Organization	1-1	Introduction	2-1
Vector List	1-2	Display Control	2-3
Description	1-2	Graphics Video On/Off	2-3
Vector Space	1-2	Alphanumeric Video On/Off	2-3
Vector List Elements	1-2	Graphics Zoom	2-3
How to Create a Vector List	1-3	Cursor Control	2-4
Object Attributes	1-4	Alphanumeric Cursor	2-4
Description	1-4	Graphics Cursor	2-4
Virtual Space	1-4	Define Display/Virtural Graphics Cursor	2-4
Object Attributes	1-5	Set Box Cursor Size	2-5
Views	1-5	Move Graphics Cursor	2-5
Description	1-5	Error Conditions	2-5
How to Define a View	1-5	Plotting Commands	2-6
View Attributes	1-6	Lift Pen	2-7
The "Active View"	1-7	Lower Pen	2-7
Zooming and Panning	1-7	Use Cursor as Next Data Point	2-8
User Defined Characters and Fonts	1-7	Rubberband Line Mode	2-8
Description	1-7	Draw a Point at the Current Pen Position	2-8
How to Create a Character	1-7	Vector Data Formats	2-8
Uses of Characters and Fonts	1-7	ASCII Formats	2-9
Color Palettes	1-8	ASCII Absolute Format	2-9
Picture File Commands	1-9	ASCII Incremental Format	2-10
Raster Color Graphics	1-9	ASCII Relocatable Format	2-10
Raster Memory	1-9	Binary Formats	2-11
Loading Raster Memory	1-12	Binary Medium Absolute Format	2-13
Specifying Output Devices and Write Masks	1-12	Binary Long Absolute Format	2-15
Specifying Display Modes and Read Masks	1-13	Binary Short Incremental Format	2-16
Double Buffer Mode	1-13	Binary Long Incremental Format	2-16
Graphics Escape Sequences	1-14	Binary Long Relocatable Format	2-17
Conventions	1-14	Mixing Data Formats	2-17
Miscellaneous Characters	1-15	Graphics Relocatable Origin	2-18
Bit Masks	1-17	Set Relocatable Origin in Absolute Coordinates	2-18
Negative Numbers	1-17		

Views	4-25	Writing to Raster	6-4
Definition	4-25	Specifying Output Devices	6-5
How to define a View	4-27	System Write Mask	6-7
Creating/Activating a View	4-27	Object Write Mask	6-10
Setting Virtual Window/Screen		Vector Drawing Modes	6-11
Viewport Limits	4-28	Raster Visibility	6-16
Viewport Highlighting and Back-		Turning On Graphics Video	6-17
ground Color	4-28	System Read Mask	6-17
View Default Attributes	4-29	Display Modes	6-18
Redrawing the Active View	4-30	Applications	6-21
Redraw Modes	4-30	Raster Data Transfers	6-21
Multiple views	4-31	Raster Dumps	6-22
View Library Commands and Status		Raster Dump Dimensions	6-23
Requests	4-31	Printer Specifications	6-23
User Defined Characters and Fonts	4-31	Data Type and Format	6-23
Definition	4-31	Start Record	6-23
How to Design a Character	4-31	Data Transfer	6-24
Specify the Character Code and		Termination Record	6-26
Font Number	4-32	Raster Loads	6-27
Define the Character Cell	4-32	Dimensioning the Source File	6-27
Plot the Character Data	4-33	Partitioning the Source Frame	
Considerations for Character Definitions ..	4-34	(Window Frame)	6-28
Data Points Outside the Character Cell ..	4-34	Partitioning the Raster	6-30
Text Origin Considerations	4-35	Raster Load Formats	6-30
Memory Considerations	4-35	Start Record	6-33
How to Use characters and Fonts	4-35	Data Transfer	6-33
Defining Fonts for Extended Roman		Termination Record	6-37
Characters	4-36	Raster Load from Datacomm	6-37
User Defined Font Library Commands	4-37	How Raster Loads are Affected by	
Section 5: Picture File Commands		Drawing Modes	6-37
Introduction	5-1	Raster Reads	6-38
Transparent Mode	5-2	Raster Defaults	6-41
Object Visibility Fence	5-3	Raster Status	6-41
Library Commands	5-3	Section 7: Graphics Input/Output	
Vector List Library Commands	5-3	Operations	
Object Library Commands	5-7	Introduction	7-1
View Library Commands	5-8	Graphics Hardcopy Devices	7-1
User Defined Character and Font		Raster Dumps	7-2
Library Commands	5-8	Raster Dump to Printer	7-2
Color Palette Library Commands	5-9	Raster Dumps to Disc or Datacomm Files ..	7-5
Protecting the Picture File	5-9	Plotting from the Picture File	7-5
Copying/Examining the Picture File	5-9	Color Raster Dumps	7-6
Choosing Selected Elements	5-9	Graphics Input Devices	7-7
How to Interpret the Picture File	5-10	Thumbwheels and Gid Key	7-7
Reading the Picture File from a		Graphics Tablet	7-7
Host Computer	5-11	Description	7-7
Picture File Defaults	5-12	Installation	7-8
Picture File Inquiry Commands	5-14	Operation	7-8
Section 6: Raster Manipulations		Controls	7-8
Introduction	6-1	Absolute/Relative Mode	7-9
Raster Memory	6-1	Lefthand/Righthand Operation	7-9
Raster Memory Operations	6-2	Using the Tablet in Graphics Operations ..	7-9
Setting Raster Memory	6-3	Tablet Escape Sequences	7-10

External Monitors and Cameras	7-10	Read Graphics Cursor Position	
Graphics I/O Device Configuration Menus ..	7-12	with Wait	8-12
Keyboard Operation of Configuration		Read Display Size	8-12
Menus	7-13	Read Device Capabilities	8-13
How to Display the Menus	7-13	Read Graphics Text Status	8-14
How to Make Menu Selections	7-13	Read Zoom Status	8-14
Activating and Saving Configuration		Read Relocatable Origin	8-15
Menu Fields	7-14	Read Reset Status	8-15
Notes on Configuration Values Stored		Read Area Shading Capability	8-15
in Non-Volatile Memory	7-14	Read Graphics Modification Capabilities ..	8-15
Printer and Plotter Configuration Menus ..	7-15	Read Secondary Cursor Status	8-16
Keyboard Configuration Menu	7-15	Status Requests (Parameters 14-30)	8-16
Raster Configuration Menu	7-15	Read Graphics Freespace	8-16
Terminal Configuration Menu	7-15	Any Other Parameter	8-16
Escape Sequence Operation	7-17	Sample Program	8-16
Specify Device Class	7-18		
Specify Device Index	7-18	Appendix A: Application Notes	
Set Working Parameters	7-18	Introduction	A-1
Select Menu Field and New Value	7-18	Animation	A-1
Configuration Menu Fields	7-19	The Layered Look	A-7
Graphics Device Configuration Inquiry	7-26	Reading from the Picture File	A-8
		HP 2647/48 and HP 2700 Graphics	
Section 8: Inquiry Commands and Status		Differences	A-11
Requests		Display Size	A-11
Graphics Inquiry Commands	8-1	Escape Sequences	A-11
Inquire View	8-2		
Inquire Implied Object	8-2	Appendix B: Transformation Equations	
Inquire Vector List	8-2	Object Attribute Transformations	B-1
Inquire Picked Object	8-3	Viewing Transformations	B-3
Inquire Graphics Device Configuration	8-3	Color Transformations	B-6
Inquire Raster Hardcopy Format	8-3	HSL to RGB	B-6
Inquire Graphics Color	8-3	RGB to HSL	B-6
Inquire Picture File	8-3		
Sample Program	8-6	Appendix C: Color Key	
Graphics Status	8-6		
Read Device Error Code	8-11	Appendix D: Escape Sequence Directory	
Read Device I.D.	8-11		
Read Current Pen Position	8-11	Appendix E: Glossary	
Read Graphics Cursor Position	8-12	Index	

Preface

HOW TO USE THIS MANUAL

This manual will explain how to incorporate the terminal's color graphics capabilities in application programs. We suggest that the first time you read this manual, you progress through the sections in the order they are presented. After you gain familiarity with the system, you can use the Index or Appendix D. GRAPHICS ESCAPE SEQUENCE DIRECTORY to locate answers to specific questions you may have.

NOTE: To see how the graphics system fits into the overall terminal design, read Section 1. INTRODUCTION of your *HP 2700 Alphanumeric Reference Manual*.

MANUAL SECTIONS AND APPENDICES

The manual consists of the following sections and appendices:

Section I. *Graphics Concepts* — provides you with an overview of all the graphics concepts discussed in the manual, and gives you important instructions for using graphics escape sequences.

Section II. *Drawing Fundamentals* — explains how to use the basic plotting elements: vectors, area fills, and graphics text.

Section III. *Color* — explains how to use the terminal's color capability as an effective communications tool.

Section IV. *Creating A Picture* — shows you how to create an image, store it in vector memory, and manipulate it on the screen using graphics escape sequences.

Section V. *Picture File Commands* — tells you how to manipulate and examine the contents of vector memory.

Section VI. *Raster Manipulations* — explains how you can directly control information written from vector memory to raster memory, and from raster memory to the screen (or other output device).

Section VII. *Graphics Input/Output Operations* — shows you how to incorporate printers, plotters, discs, video monitors, video cameras, and the HP 13273T Graphics Tablet into your graphics system.

Preface

Section VIII. *Inquiry Commands and Status Requests* — shows you how to obtain information about the picture file from an application program running on a host computer.

Appendix A. *Application Notes* — discusses differences between this terminal and other Hewlett Packard graphics terminals. It also provides examples and helpful programming hints.

Appendix B. *Transformation Equations* — provides you with the equations used internally by the terminal to perform object transformations and picture clipping.

Appendix C. *Color Key* — provides sample colors and their RGB values.

Appendix D. *Escape Sequence Directory* — lists all the graphics escape sequences and the sections where they are discussed.

Appendix E. *Glossary* — provides a useful list of graphics terms used in the manual and their definitions.

Index

HP 2700 SERIES TERMINAL DOCUMENTATION LIST

Other manuals which have been prepared to assist you in learning about your terminal are listed below.

The *HP 2700 User's Manual* (HP part no. 02703-90002) will acquaint you with the various features of the terminal, especially its keyboard operation. (NOTE: This manual contains a complete description of the graphics keypad.)

The *HP 2700 Alphanumeric Reference Manual* (HP part no. 02703-90003) provides the systems programmer with the information needed to use the terminal's alphanumeric capabilities in a variety of applications. Extensive coverage of installation procedures, configuration menus, and datacomm operation is also included.

You may also order as options any of the following manuals:

The *HP 2700 Service Manual* (HP part no. 02703-90004) provides information regarding trouble-shooting and repair.

The *Autoplot/2700 User Manual* (HP part no. 13273-90001) shows you how to produce a variety of charts and graphs using *Autoplot/2700* software.

The *Paintbrush/2700 User Manual* (HP part no. 13273-90003) shows you how to edit a picture locally and produce freehand art on the screen using *Paintbrush/2700* software and, optionally, the HP 13273T Graphics Tablet.

FEATURES AND ACCESSORIES

The HP 2700 Series is a high-performance, color graphics terminal which lets you interactively create and manipulate pictures. By utilizing its local processing capabilities you can substantially ease the burden on your host computer. In addition, your terminal has these features:

- **Local Vector Storage With a Raster Display** A picture can be redrawn from the information stored in vector memory, even after the raster has been cleared.
- **Segmented Picture File** A picture is created from objects which can be manipulated and copied.
- **Large Plotting Space** Two-dimensional pictures are stored to a high resolution in a local memory addressing space of 32K × 32K "points".
- **True Zoom and Pan** Local vector storage lets you redefine your view of the picture, and thereby zoom and pan over the 32K × 32K plotting space.
- **Multiple Views** All or part of your picture can be selectively viewed on different portions of the screen. Up to 256 different views can be stored in memory and displayed simultaneously.
- **High-Speed Vector Generation** Local processing capability yields high-speed graphics interaction and saves CPU time. The terminal uses a 2901 bit slice processor for vector to raster conversion and a 68000 processor to perform clipping and transformations.
- **Color** 16 different colors (selected from a possible 4096) can be displayed simultaneously on the screen. Two methods of color selection are available: hue-saturation-lightness and red-green-blue. Mapping from the terminal to black and white hardcopy devices can be achieved through automatic halftoning.
- **User Defined Characters and Fonts** You can create and store your own graphics alphabet or symbol library.
- **Graphics Text** You can add graphics text to your picture.
- **Graphics File Handling** Integral and peripheral mass storage devices are available for picture storage.
- **Graphics Input Devices** The terminal has an integral graphics input device consisting of cursor-control thumbwheels and a GID key. It also supports the optional HP 13273T graphics tablet.
- **Output Devices** The terminal supports the following interfaces to peripheral output devices:
 - RS-422 (RS-232C compatible) (standard)
 - Shared Peripheral Interface (optional); provides interface capability to HP-IB compatible products
 - External Video Interface (optional); for R-G-B compatible monitors and cameras
- **Expandable Memory** The terminal has a standard raster memory. By purchasing a second raster memory you can achieve double buffered picture redraws. You may also add vector memory and application memory.

GRAPHICS KEYPAD

Some of the graphics functions performed by escape sequences are also controlled by graphics keys. Each of these keys is explained in detail in Section 6. THE GRAPHICS KEYS, of the *HP 2700 User's Manual*.

COMPATIBILITY WITH OTHER HEWLETT-PACKARD GRAPHICS TERMINALS

Most of the drawing commands used to create picture on other Hewlett Packard graphics terminals are recognized by this terminal. For a complete list of differences, refer to Appendix A. APPLICATION NOTES.

Graphics Concepts _____ 1

INTRODUCTION

This section will familiarize you with the fundamental graphics concepts of your terminal. The material is divided into three parts:

- *Picture Organization*: Explains how pictures are created on the terminal and how picture data is stored in local terminal memory. You should read this section before proceeding to the explanation of escape sequence programming in Section 4. **CREATING A PICTURE**.
- *Raster Color Graphics*: Describes how color graphics are represented in raster memory. This discussion prepares you for Section 3. **COLOR** and Section 6. **RASTER MANIPULATIONS**.
- *Graphics Escape Sequences*: Provides a detailed explanation of graphics escape sequence conventions and errors.

PICTURE ORGANIZATION

The terminal not only displays a picture on the screen, it also has the capability of storing the data used to create the picture in local terminal memory. The stored data (vectors, pens, graphics text, et cetera) is referred to as the *picture file*.

The process of storing image data in the picture file and displaying it on the screen is accomplished in three steps:

1. The generation of drawing commands to form a graphical entity: *VECTOR LIST*
2. The application of transformation parameters to the vector list to cause the graphics entity to be scaled, rotated, and positioned on an imaginary plotting space: *OBJECT ATTRIBUTES*
3. The mapping of all or a portion of the imaginary plotting space to a section of the screen: *VIEW*

Vector Lists, Object Attributes, and Views are stored as memory blocks in the picture file. Two other members of the picture file are:

4. *COLOR PALETTES*: A set of any 16 of the available 4096 colors may be saved in the picture file as a palette. The 16 colors are referred to as "pens". Up to 255 palettes can be stored in the picture file.
5. *USER DEFINED CHARACTERS AND FONTS*: You may design your own graphic type face or symbol library.

Each of the five members of the picture file, introduced above, is discussed in the following paragraphs.

Vector List: A Group of Drawing Elements Which Form A Graphical Entity

DESCRIPTION. The first step toward creating a picture on the terminal is the construction of a *vector list*. A vector list contains drawing elements called *primitives* (vectors, area fills, and labels) and *primitive attributes* (color pen numbers, line types, area fill patterns, et cetera) which describe a graphical entity. Since this entity can later be scaled, rotated, repositioned, and copied, a vector list is usually a building block of a larger picture.

VECTOR SPACE. Valid data points for images in vector lists are integers in the range (-16383, . . . , +16383). Therefore, you can think of the drawing as being plotted on a grid with limits of -16383, -16383 and +16383, +16383. This grid is referred to as *vector space*. The concepts of vector lists and vector space are illustrated in figure 1-1.

VECTOR LIST ELEMENTS. A vector list can contain any of the following drawing elements:

- *Plotting Commands and Data Points*: moves, draws, area fills
- *Vector Drawing Modes*: line types, area fill patterns, drawing modes
- *Graphics Text and Attributes*: labels which may be slanted, scaled, rotated, et cetera
- *Symbols*: characters plotted at vector end points
- *Color Pen Numbers*: color pens used to draw vectors

NOTE: Most of the commands for the above categories are compatible with other Hewlett-Packard graphics terminals. They are explained in Section 2. DRAWING FUNDAMENTALS.

A vector list may also include:

- *Pick ID*: a programming aid used when editing the picture file. (See SECTION 8. INQUIRY COMMANDS AND STATUS REQUESTS)

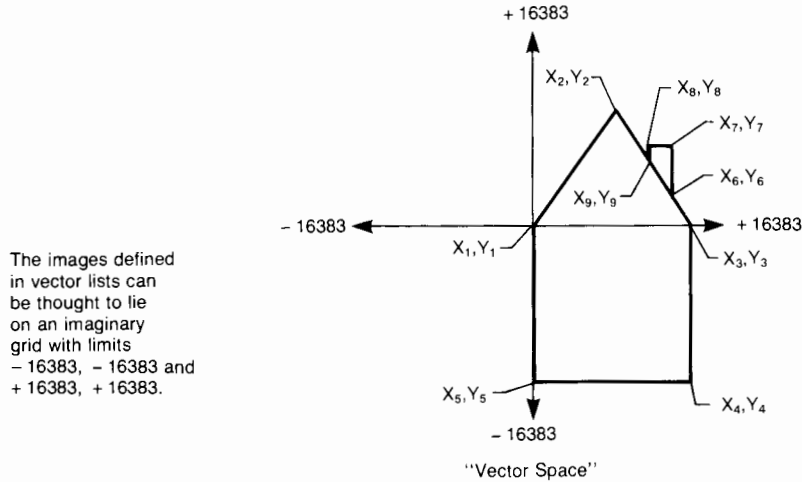
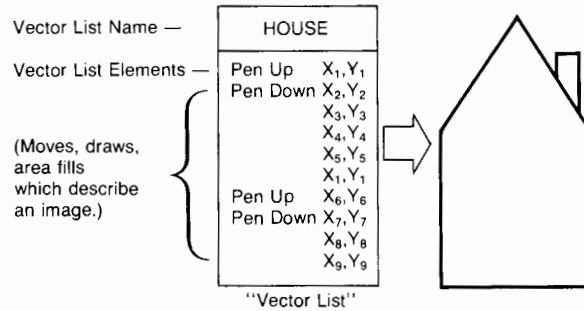


Figure 1-1. Vector Lists, Vector Space

HOW TO CREATE A VECTOR LIST. At any time, the terminal has an “open” vector list; that is, a vector list which can accept and store primitives. Primitives are created with escape sequences (or application software). After the primitives have been entered into the vector list, the vector list must be “closed” in order to save it in the picture file. A vector list is closed by naming it in an escape sequence, and thereafter, it is always referenced by name.

NOTE: Once a vector list is closed, it may not be reopened or edited. If you wish to alter the picture, you must create new vector lists. This restriction is not a disadvantage when you consider the relationship of the vector list to the picture file. A vector list is used as a building block or library element to *create* a picture file. Its function is similar to that of lines or symbols on a simpler terminal.

Example. By following these instructions, you can make a vector list for a square.

Step 1. Press `SHIFT / CLEAR` to reset the graphics defaults.

Step 2. Type `ε*pa 100,100 200,100, 200,200 100,200 100,100Z`
(A square should appear on your screen.)

Step 3. Now type `ε*h<square>c`. (Be sure to type the angle brackets.) Did the square disappear? It should! We'll bring it back to the screen by applying object attributes to it. Read on!

Object Attributes: Transformation Parameters Applied to Vector Lists

DESCRIPTION. A vector list is not visible on the screen until it is linked with another data structure — a set of *object attributes* — which specifies the scaling, rotation, translation, and location of the vector list. The combination of a vector list and a set of object attributes constitutes an *object*. By applying different sets of object attributes to one vector list, you can construct several objects.

VIRTUAL SPACE. When object attributes are applied to a vector list, the graphic representation is translated to another address space of graphics memory which also has limits of $-16383, -16383$ and $+16383, +16383$. The manual refers to this grid as *virtual space*. This is the plotting space for your picture.

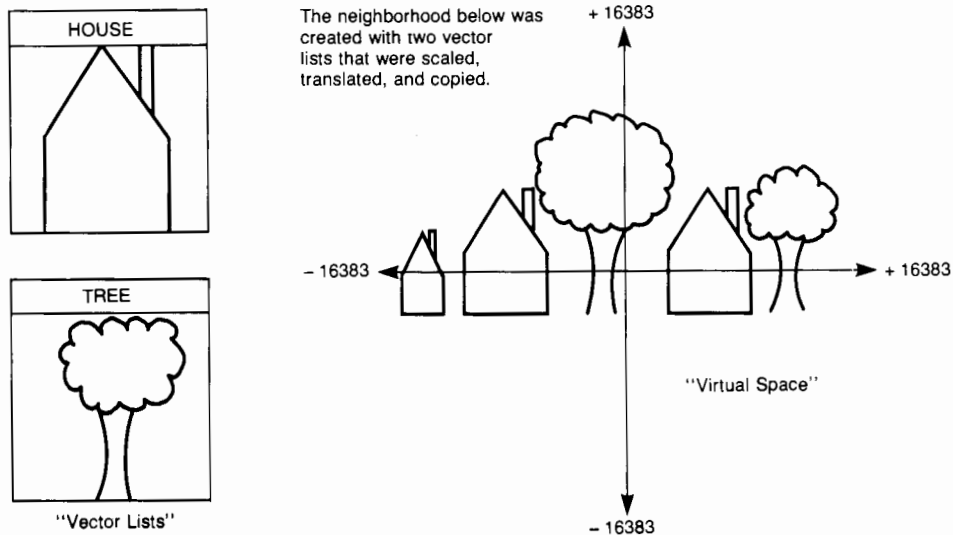


Figure 1-2. Object Attributes Applied to Vector Lists in Virtual Space

OBJECT ATTRIBUTES. The object attributes which may be applied to vector lists are as follows:

- *X and Y Transformation Center:* All transformations take place about this point.
- *Scaling:* The object may be scaled independently in either the X or Y direction.
- *Rotation:* The object may be rotated about its transformation center.
- *Translation:* The object may be translated ΔX and ΔY units in virtual space.

Example. Create an object with object attributes named "box" from your vector list "square". Type `g<box>a<square>c`. If your square does not reappear on the screen, repeat the steps from the previous exercise.

Views: The Mapping of the Picture in Memory to the Screen

DESCRIPTION. Objects, as they are positioned in virtual space, are mapped to the screen. You may display all of the $32K \times 32K$ virtual space at once, and thereby sacrifice detail on the 512×390 screen, or you may view only a small portion of virtual space in exchange for more detail. The sections of virtual space which are ultimately displayed on the screen are called *views*.

HOW TO DEFINE A VIEW. In order to create a view, you must first define the lower left and upper right boundaries of the desired area of virtual space. This rectangle is called the *virtual window*. The virtual window may be the entire virtual space or any subsection thereof (within the limits of $-16383, \dots, +16383$).

Similarly, you may specify a rectangular area of the screen where the contents of the virtual window will be displayed. This rectangle is the *screen viewport*. The screen viewport may be defined to be the entire raster memory (512×512) or a subsection thereof.

NOTE: Raster memory consists of 512×512 pixels, although only 512×390 are displayed on the screen. The remaining portion of memory may be viewed on certain monitors and cameras or accessed via raster dumps. (See Section 7. GRAPHICS INPUT/OUTPUT OPERATIONS.)

Data which falls outside the limits of the virtual window is "clipped", and not displayed on the screen. Equations for the mapping of the virtual window to the screen viewport may be found in Appendix B. TRANSFORMATION EQUATIONS.



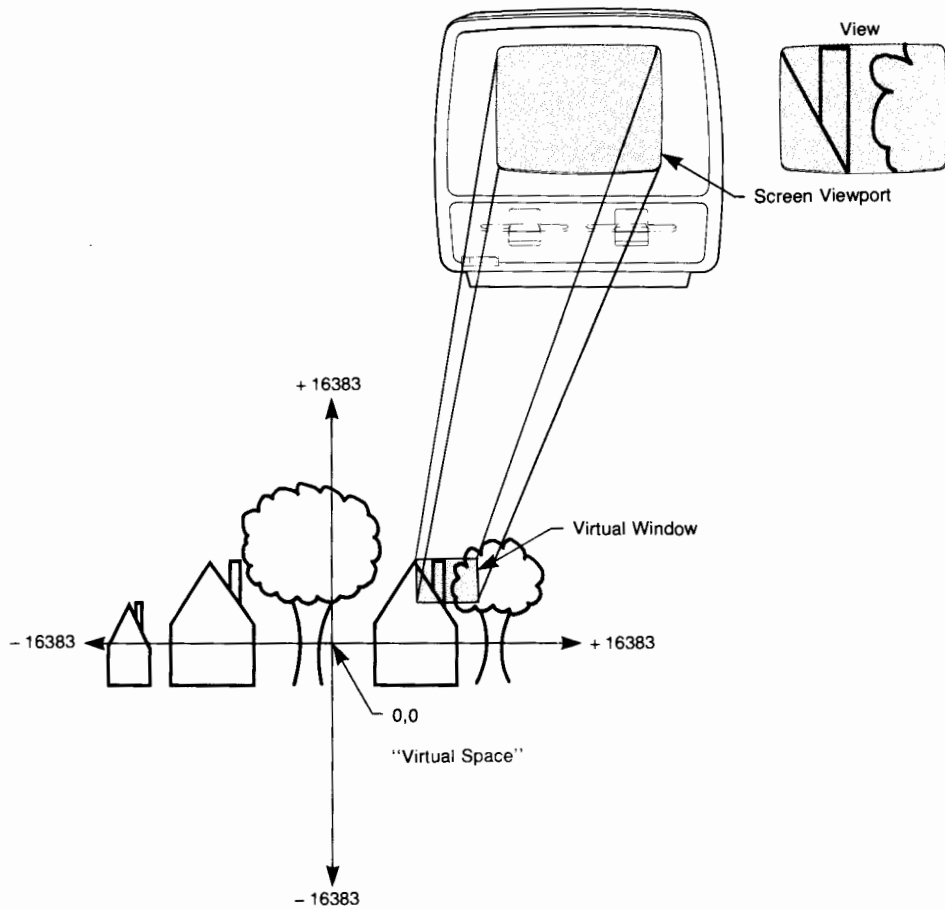


Figure 1-3. Views

VIEW ATTRIBUTES. The view attributes, or qualities of a view which you may define, are:

- *Virtual Window Limits:* $-16383, \dots, +16383$
- *Screen Viewport Limits:* $0, \dots, 511$
- *Highlighting:* You may outline or "highlight" the view with the color pen of your choice.
- *Viewport Background Color:* The area behind the objects is filled in with the color associated with the specified pen.

THE "ACTIVE" VIEW. The terminal allows you to store up to 256 views. Views have reference numbers (just as vector lists and object attribute lists have names) by which they are addressed. The terminal always has one (and only one) active view at any time. Although more than one view may be displayed, only the active view may be redrawn or have its view attributes changed.

NOTE: You can use the graphics keys to modify your current view of the object "box". Read the descriptions entitled "Views" and "Redrawing the Active View" in Section 6. THE GRAPHICS KEYS of the *HP 2700 User's Manual*.

ZOOMING AND PANNING. By redefining the window/viewport relationship you can zoom in and pan over virtual space. Lost detail can be picked up over a limited area by shrinking the size of the virtual window as shown in the following equation:

$$\text{zoom factor} = \frac{\text{size of viewport}}{\text{size of window}}$$

The zoom center is defined to be the center of the virtual window. See also "Zooming and Panning" in Section 6 of the *HP 2700 User's Manual*.

User Defined Characters and Fonts

DESCRIPTION. You can design a character and store it in the picture file. Up to 95 of these characters can be assigned to a "font". The terminal can reference a maximum of 14 user defined fonts, depending upon the amount of available vector memory.

HOW TO CREATE A CHARACTER. A character is created using special move, draw, and area fill commands. Each character is created within a "character definition space" or character cell. A character is assigned an ASCII number (which associates it with a graphics ASCII character (32, . . . , 126) for use by graphics text) and a font number.

USES OF CHARACTERS AND FONTS. Characters/Fonts may be used in the following ways:

- As symbols used to define the end points of vectors.
- As substitutes for the system font for graphics text. When used in this capacity, user defined fonts are subject to the same labels attributes (size, slant, rotation, et cetera) as the system font.

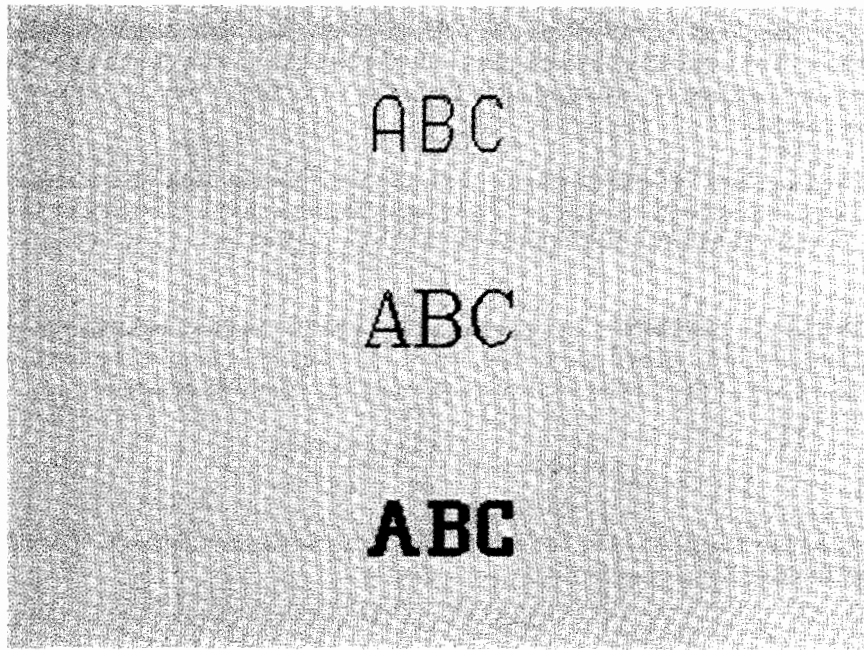


Figure 1-4. Examples of User Defined Characters and Fonts

Color Palettes

Any 16 of 4096 available colors can be displayed at once. A set of 16 colors may be saved within the terminal as a *palette*. 255 palettes can be stored locally. Any saved palette can become active with one quick command.

The 16 colors of a palette are referred to as *pens*. At any time, one of these pens is assigned as the *primary pen*, and another is assigned as the *secondary pen*. The primary pen is used to draw vectors. The secondary pen is used in conjunction with the primary pen to define line and area fill patterns when certain drawing modes are in effect. These pen assignments are entered as primitives into vector lists. Note that the actual color of a pen will depend on the palette in use.

Two models are available for color specification: (*RGB*) — mixing red, green, and blue or (*HSL*) — selecting hue, saturation, and lightness.

To observe the current palette, press the COLOR key on the graphics keypad. This key is explained in detail in Section 6 of the *HP 2700 User's Manual*.

Picture File Commands

The terminal has several commands through which you can manipulate the contents of the picture file:

- Any portion of the picture file (vector lists, object attributes, views, user defined fonts, or color palettes) may be deleted.
- Vector list and object attribute names and references may be changed.
- A special mode may be selected which prevents primitives from being stored in vector lists.
- The current picture file can be protected from being overwritten by like named elements read in from datacomm or disc.
- The picture file can be examined or saved by copying selected elements to the display, printer, disc, or datacomm.

These commands are discussed in Section 5. PICTURE FILE COMMANDS.

RASTER COLOR GRAPHICS

Raster Memory

Raster memory stores a graphics image as an array of binary numbers that represents the screen display. The raster memory of a black and white terminal is composed of a 2-dimensional array of binary numbers, where each bit represents a picture element or *pixel*. If a bit=1, the phosphor dot on the screen is lit. If a bit=0, the dot is not lit.

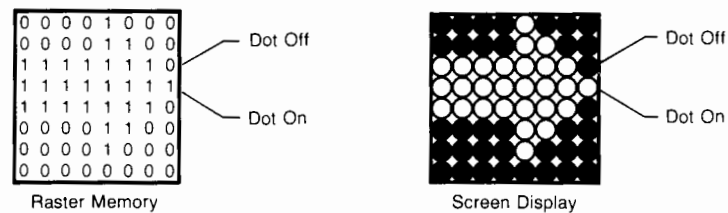
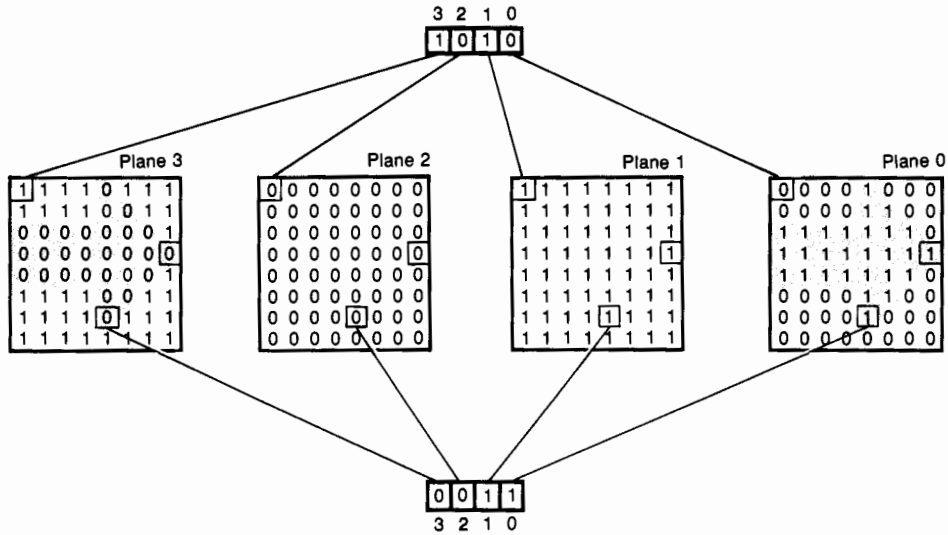


Figure 1-5. Raster Memory and Screen Display of a Black and White Terminal

A pixel of an HP 2700 Series terminal consists of three phosphor dots (red, green, blue). By controlling the intensity of each, these dots can produce one of the 16 colors of the active palette. Thus, an associated raster pixel is represented by 4 bits (16 possible states). It will be convenient for purposes of discussion to picture the color raster as an extension of the black and white raster model, i.e. 4 sets of 2-dimensional arrays, as shown in figure 1-6. Each array is referred to as a *raster plane*. A set of four raster planes is referred to as a *raster buffer*.



The value of the pixels for the arrow is 0011B=3.
The value of the background pixels is 1010B=10.

The combined pixel array looks like:

10101010	3	101010
10101010	3	3 1010
3 3 3 3	3 3 3 10	
3 3 3 3	3 3 3 3	
3 3 3 3	3 3 3 10	
10101010	3 3 1010	
10101010	3 101010	
10101010	101010	

Figure 1-6. Color Raster

The pixel values from raster memory are used to address a look-up table (on the Color-Mapper board) which associates the raster pixel value with a color from the active palette (figure 1-7). The table contains the values used to control the red, green, and blue phosphors. In other words, the pixel value provides the pen number; the color mapper controls the color. (See the *HP 2700 Alphanumeric Reference Manual* for more information on the color mapper.)

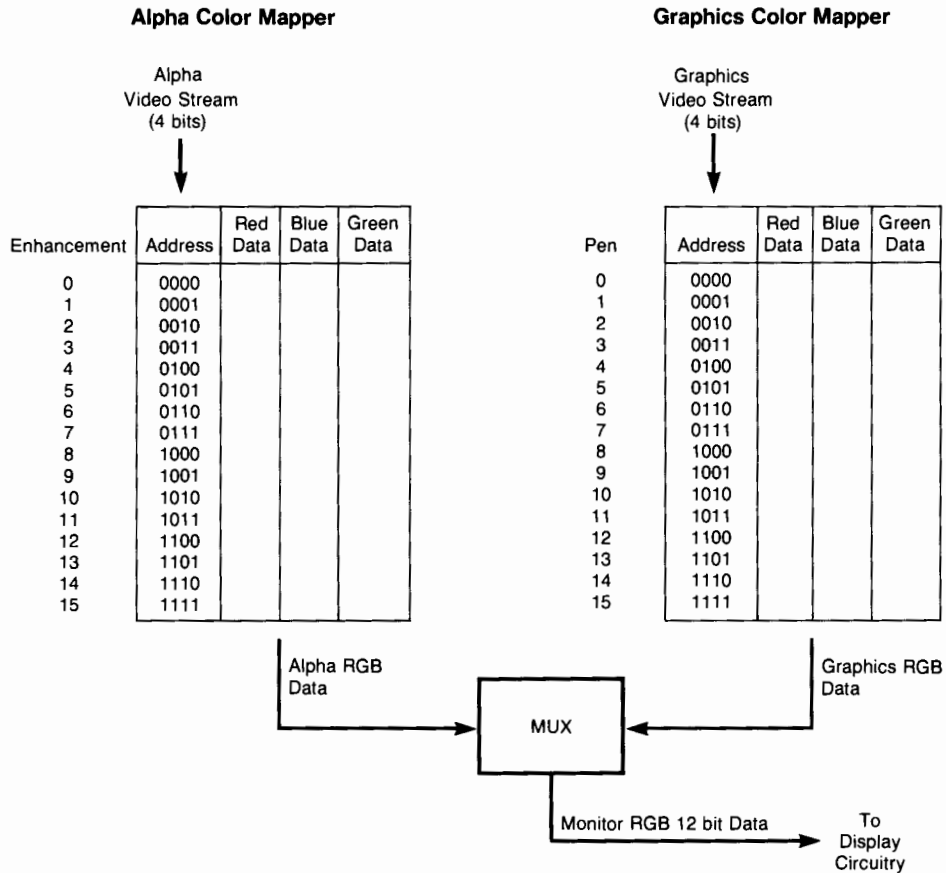


Figure 1-7. Color Mapper

NOTE: An optional accessory of the HP 2700 Series Terminal is an auxiliary raster buffer. This manual will discuss some possible applications of two raster buffers in Section 6.

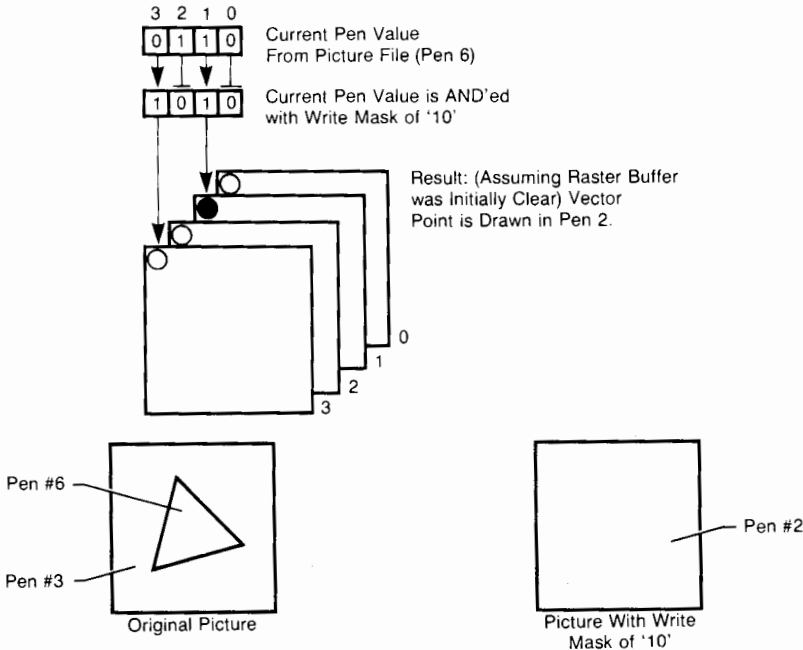
Loading Raster Memory

The graphics processor calculates the image in the active view, and loads the appropriate bits into raster memory. The final display is dependent upon two factors:

- Output Devices/Write Masks
- Display Modes/Read Masks

SPECIFYING OUTPUT DEVICES AND WRITE MASKS. You can specify which raster buffer may receive data by selecting an *output device*. In addition, the terminal lets you select which raster planes may receive data from the graphics processor by setting a *display write mask*. If a mask bit is set to '1', then data may be transferred to the associated plane. If the bit is set to '0', the associated plane may not receive data. You can also set write masks for individual objects.

Write masks may affect the color of a vector. Normally, if a vector is drawn in pen 6, the bits in planes 1 and 2 associated with the vector pixels would be set to '1'. However, if a write mask is set which inhibits writing in planes 0 and 2, the vector will appear to be drawn in pen 2 (0010_B), assuming all the raster bits were initially set to '0'.



The only raster planes which may receive data are planes 3 and 1. There is no data for plane 3, and pens 6 and 3 both use plane 1. The result is a raster filled with pixel values that select pen 2.

Figure 1-8. Write Masks

SPECIFYING DISPLAY MODES AND READ MASKS. Just as write masks allow you to control the flow of data written to a raster, display modes and read masks allow you to control the flow of raster data sent to the display or other output device. A *display mode* selects which raster buffer will be displayed; a *read mask* selects which planes will be displayed. Thus, the pixel value may be altered before it is sent to the color-mapper.

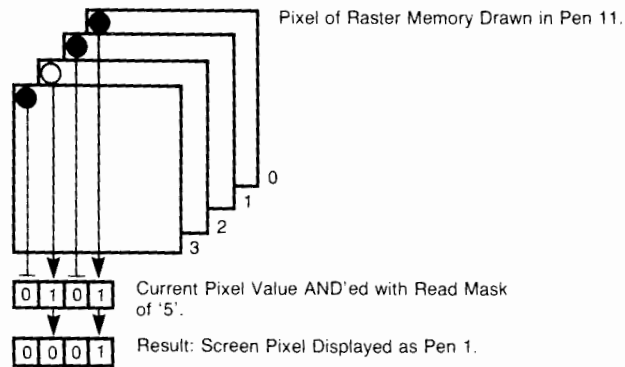


Figure 1-9. Read Masks

DOUBLE BUFFER MODE. If your terminal contains the auxiliary raster buffer, you can set *Double Buffer Mode*. Double buffer mode makes use of both read and write masks. Initially, a read mask is set so that only one buffer is displayed on the screen, and a write mask is set so that vector information can only be written into the non-displayed buffer. While one buffer is displayed on the screen, new information from the picture file is written into the non-displayed buffer. When the writing is finished, new read/write masks are set so that the new information is displayed, and the old information is updated. The apparent result is a smooth redraw of the picture on the screen.

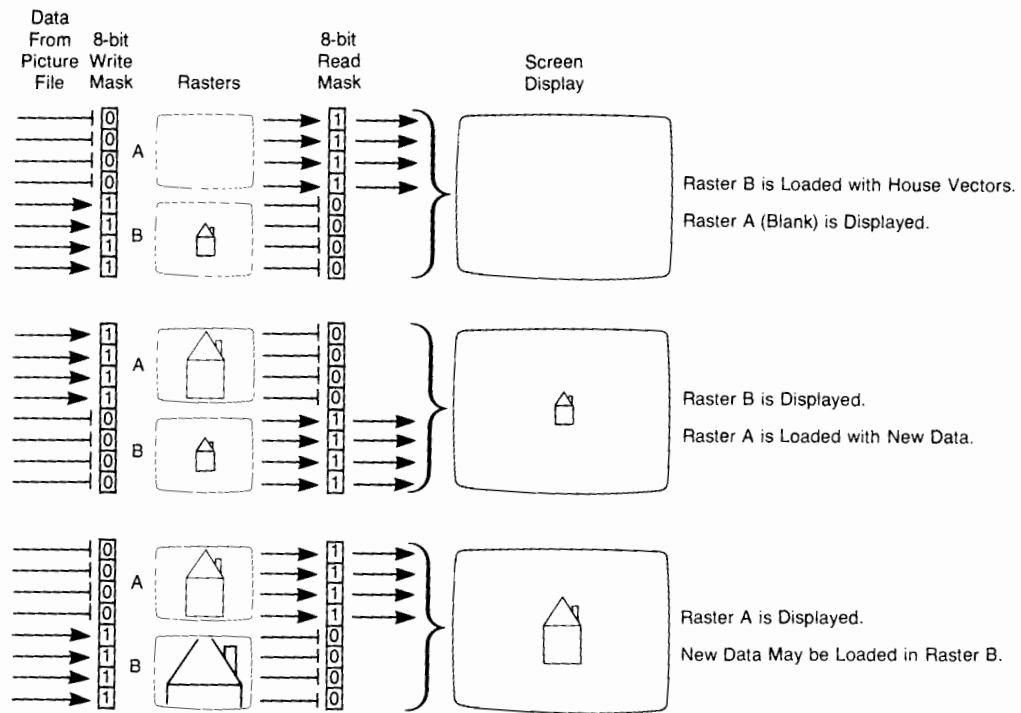



Figure 1-10. Double Buffering

NOTE: You can set double-buffer mode using escape sequences or using the  key on the graphics keypad.

GRAPHICS ESCAPE SEQUENCES

All terminal color graphics functions can be accomplished through escape sequences. An escape sequence is a series of ASCII characters preceded by the escape character. The escape character signals the terminal that the succeeding characters form a command.

Conventions

The general forms of the graphics escape sequences are:

- Esc* <FG> <FL> or <FU> ,
- Esc* <FG> ! <decimal parameter> ! <FL> or <FU> , and
- Esc* <FG> <<character parameter>> <FL> or <FU>

where $\langle FG \rangle$ is the function group or family indicator which must be a lower case character from the set (a,b, . . . , z)

e.g., $\text{\textbackslash} * v$ sequences are color commands.

$\text{\textbackslash} * p$ sequences are plotting commands.

$\text{\textbackslash} :$ contains one or more decimal parameters separated by spaces or commas

e.g., $\text{\textbackslash} * g \langle Sx \rangle, \langle Sy \rangle s$ — Set object scaling.

$\text{\textbackslash} * g 2, 2 s$ or $\text{\textbackslash} * g 2 2 s$ — Scale object by a factor of 2 in the X and Y directions.

$\langle \rangle$ are angle brackets which always enclose character parameters

e.g., $\text{\textbackslash} * g \langle \langle \text{object attributes name} \rangle \rangle A$ — Activate object attributes.

$\text{\textbackslash} * g \langle \text{car} \rangle A$ — Activate object attributes for "car".

$\langle FL \rangle$ is a lower case function terminator from the set (a,b, . . . , z).

e.g., $\text{\textbackslash} * g \langle Tx \rangle, \langle Ty \rangle t$ - Set object translation.

If $\langle FL \rangle$ is used to terminate a function, other function terminators from the same family or function group may be appended to the sequence until an upper case terminator $\langle FU \rangle$ is used.

$\langle FU \rangle$ is an upper case function terminator from the set (A,B, . . . , Z, {, |, }, ^). If $\langle FU \rangle$ is used, it terminates both the function and the escape sequence. (It may also cause an automatic redraw of the picture.)

e.g., $\text{\textbackslash} * g \langle Sx \rangle, \langle Sy \rangle s$ - 'S' ends sequence

$\text{\textbackslash} * g \langle Sx \rangle, \langle Sy \rangle s \langle Tx \rangle, \langle Ty \rangle T$ - 'T' ends sequence

NOTE: $\text{\textbackslash} * 1$ (label) commands are terminated by $\text{\textbackslash} :$, $\text{\textbackslash} :$, or $\text{\textbackslash} :$.

MISCELLANEOUS CHARACTERS. No spaces are allowed among the first three characters of an escape sequence, (e.g. $\text{\textbackslash} * g$), but spaces after these are ignored (except in character parameters). Carriage returns and linefeeds are ignored after the first three characters (except in label commands — $\text{\textbackslash} * 1$).

NON-OPERATIVES. "Undefined" function letters act as non-operatives (NOP's). They perform no action, but may be used to end the sequence if they are upper-case. The letter "Z" is a commonly used NOP.

Example. `␣*pa 0 0 100 100 Z`

NOTE: The use of undefined function letters may cause compatibility problems with future terminals.

VALID DECIMAL CHARACTERS. The terminal accepts ASCII numbers (0, . . . ,9), signs (+,-), and decimal point (.) as valid characters for decimal parameters.

DECIMAL PARAMETER STORAGE PRECISION. The resolution to which a real number is stored is listed beside the parameter range.

Example. Set Text Size: `␣*m<X size><Y size>m`

where <X size>,<Y size> are in the range (-16383.996, . . . , +16383.996)(.004)

"(-16383.996, . . . , +16383.996)" is the range of valid parameter values. "(.004)" is the resolution to which the the parameter is stored.

Examples of formats used to store real numbers in the terminal are:

1. (0.001, . . . , 63.999)(.001)

where 2 bytes are used to store the number; 6 bits for the integer portion and 10 bits for the fraction.



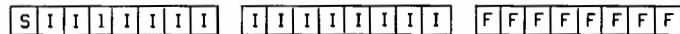
2. (-127.996, . . . , +127.996)(.004)

where 2 bytes are used to store the number; 1 sign bit, 7 bits for the integer portion, and 8 bits for the fraction.



3. (-16383.996, . . . , +16383.996)(.004)

where 3 bytes are used to store the number; 1 sign bit, 15 bits for the integer portion, and 8 bits for the fraction.



DELIMITERS. Decimal parameters may be separated by commas and spaces.

BIT MASKS. Some escape sequences require a value which acts as a bit mask. The decimal parameter entered is transformed into its binary equivalent, and the bits whose values are '1' determine the action to be taken. Bit masks are requested in the commands:

- *Inquiry Commands:* e.g., `^*h<box>s<mask>^` — Inquire information about the vector list "box"

where <mask> is an integer between -32767 and +32768. The two least significant bits of the binary representation of this number determine the information returned according to the table below.

<i>bit</i>	<i>format</i>	<i>meaning</i>
0	n	vector list name exists (0/1)
1	n	object with vector list exists (0/1)

- *Read and Write Masks:* A mask can also be associated with a picture element (or pixel). A pixel is represented by 4 bits in raster memory. A read or write mask determines whether or not information can be read from or into a particular plane. (See "Raster Color Graphics" in this section and Section 6. RASTER MANIPULATIONS.)
- *Pen number selections, user defined lines types and area fill patterns:* See Section 2. DRAWING FUNDAMENTALS.

NEGATIVE NUMBERS. The terminal will accept negative numbers as parameters. In such a case, the two's complement form of the positive number will be used.

Two's complement of a binary number is determined as follows:

- Every '1' is changed to a '0'.
- Every '0' is changed to a '1'.
- '1' is then added to the least significant bit.

Example. Assume a graphics command requires that you supply an 16-bit mask, and you enter -14. The terminal calculates the two's complement of the decimal number:

- $14 = 00000000\ 00001110_B$
- transform $00000000\ 00001110$ to $11111111\ 11110001$
- | | |
|-----|-------------------|
| add | 1 |
| | 11111111 11110010 |

Therefore, if you supply a command with an 16 bit mask of -14, you will get the same result as if you had entered a mask of +65191. (Note, however, that the terminal does not accept positive numbers over 32767, nor will it ever be necessary to set the most significant bit.)

Errors

This section describes how the terminal handles escape sequences which were incorrectly formed or attempted to specify some illegal action. Error conditions are checked in the following order:

- Major Sequence Errors
- Illegal Parameter Errors
- Non-Parameter Errors
- Execution Errors

MAJOR SEQUENCE ERRORS. The following conditions are noted as major sequence errors.

- The occurrence of a second escape character before a terminator. The escape sequence is immediately aborted, and a new sequence is started. Labels are terminated.
- An occurrence of a parameter which is not in the range ($-32768, \dots, 32767$). Characters are swallowed until the next `<FL>` is received. The `<FL>` is swallowed, and the processing of the sequence continues.
- More than 255 characters in a graphics label (`%*1`). The remainder of the escape sequence is printed on the alphanumeric display if the `% Abort` field of the Terminal Configuration Menu is set to "YES". If `% Abort` is set to "NO", the overflow is not displayed, and the entire sequence is aborted.

ILLEGAL PARAMETER ERRORS. When a major sequence error has not occurred, the parameters of the escape sequence are verified as shown in figure 1-11.

NON-PARAMETER ERRORS. When a function has collected and verified its parameters, it evaluates the context for performing the function. During this evaluation, the following errors may be discovered:

- *Undefined character in label.* The 'undefined character' error is reported, and the character is replaced by a blank.
Example. Activating a user defined font, and entering characters into a label which are not defined for that font.
- *Undefined font.* The 'undefined font' error is reported, and the system font (font 1) is used.
Example. Attempting to activate a font which does not exist.
- *Undefined operand.* The 'undefined operand' error is reported, and no action is performed.
Example. Deleting a view number which does not exist.
- *Illegal operand.* The 'illegal operand' error is reported, and no action is performed.
Example. Attempting to delete the active view.

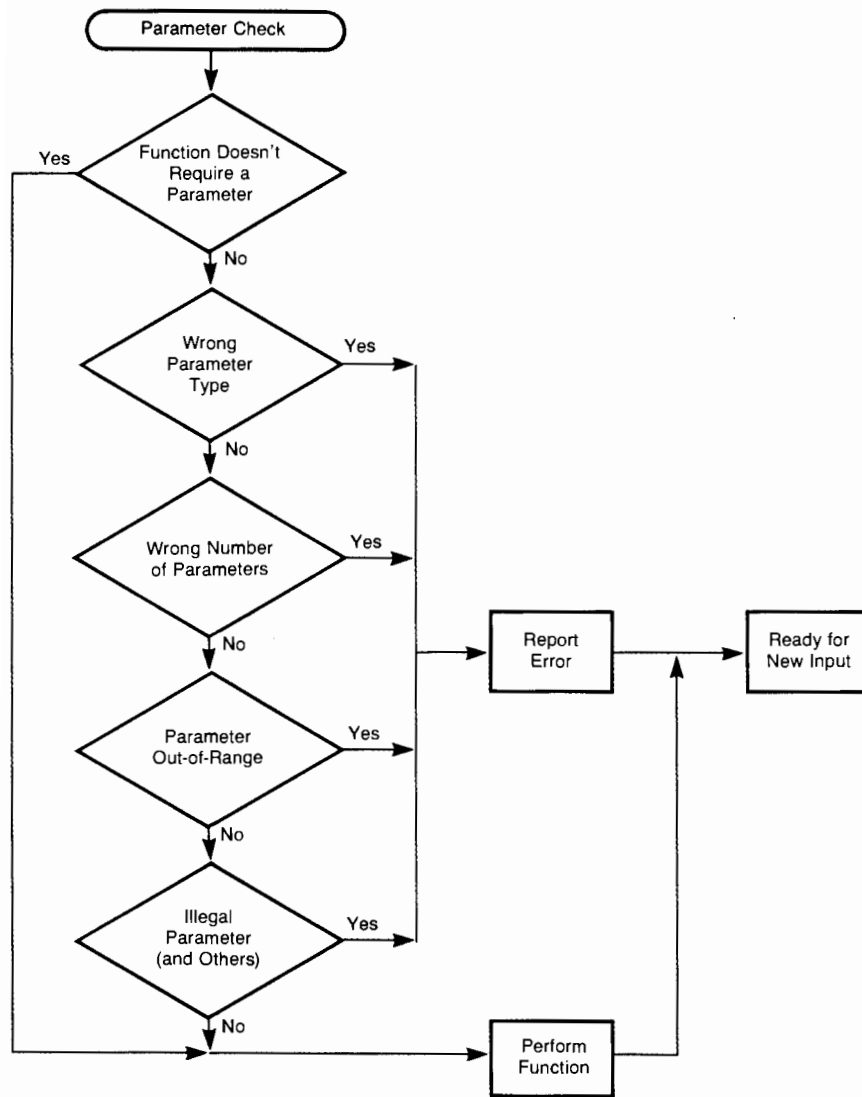


Figure 1-11. Parameter Checking for Graphic Escape Sequences

EXECUTION ERRORS. During the process of executing a graphics function, the following errors may be discovered:

- The *required amount of vector memory* for performing the operation is not available. The terminal will attempt to display any primitives received, but they will not be saved as part of a vector list.
- An *arithmetic overflow* may occur. Although this error is not major, unexpected results may occur.

Example. If an object transformation parameter is entered which causes an arithmetic overflow, the object (or a portion of the object) may disappear from the screen. However, the object is still in vector memory.

EXCEPTIONS. When an error is discovered, the function is aborted while parsing of the sequence continues. Two exceptions to the rule are:

- *Inquiry Masks* where the only error is a zero mask. The terminal will reply to the host with a null string.
- *Binary transfers* to the terminal ($\epsilon * \text{bU}, \text{v}, \text{w}$) where an error is discovered in the parameter string. The rest of the sequence (including the binary data) will be displayed on the screen. Note that unpredictable results may occur if the data contains a decimal 27 (the escape character) since the terminal may interpret the characters which follow as a command.

Collecting and Reporting Errors

When an error is discovered, it is reported, and the function is usually aborted. If other functions have been appended to the escape sequence, the processing of these commands will normally continue.

Errors are reported in binary form; the bit assignments are as follows:

<i>bit</i>	<i>meaning</i>
0	wrong parameter type
1	wrong number of parameters
2	parameter out of range
3	illegal parameter
4	undefined character
5	undefined font
6	undefined operand
7	illegal operand
8	arithmetic overflow
9	memory not available
10	graphics hardware failure

In order to find out the type of error which has occurred, you must use an inquire command and collect the error code in a string variable. The following BASIC program illustrates this process.

```
100 DIM A$(80);
110 REM Request error code.
120 PRINT CHR$(27)``*s0^''
130 LINPUT A$
```

SUMMARY

The entire process of storing graphic information in vector memory and displaying it on the screen can be summarized as follows:

1. The definition of graphics images in vector lists
2. The positioning of the vector lists as objects in virtual space
3. The mapping of virtual space to raster memory
4. The control of graphic data written to raster memory
5. The control of raster data sent to the color mapper
6. The selection of a palette for the color mapper

Figure 1-12 contains a diagram of the terminal's graphics operations and associated escape sequences.

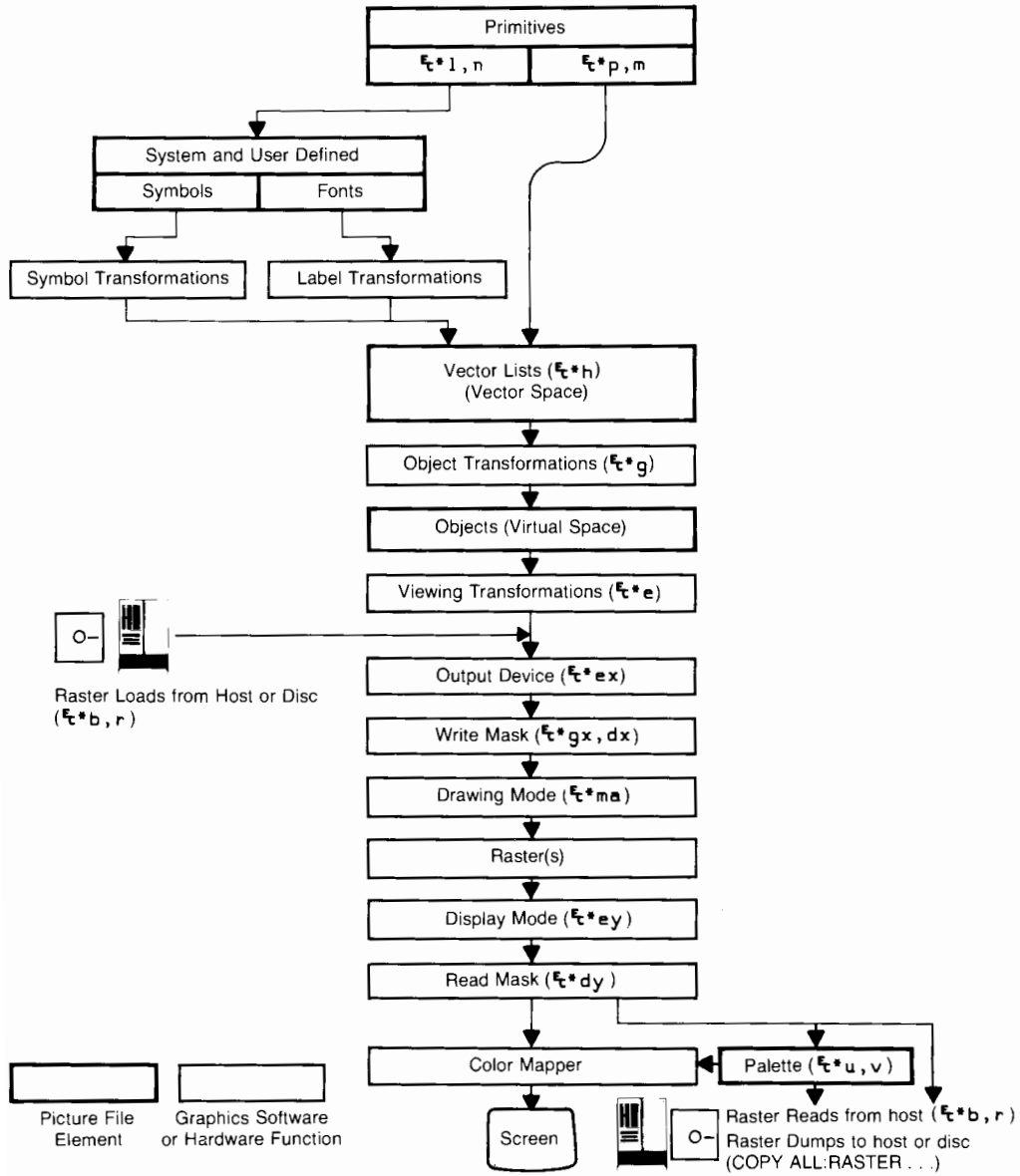


Figure 1-12. Graphics Concepts Summary



Drawing Fundamentals _____ 2

INTRODUCTION

This section describes some of the basic drawing functions that the terminal performs. Section 4. CREATING A PICTURE will explain how these functions are used with the terminal's advanced capabilities.

The terminal's drawing functions are organized into the following groups:

- Display control
- Cursor control
- Plotting Commands
- Vector Data Formats
- Graphics Relocatable Origin
- Vector Plotting Pens
- Line Types
- Area Fills
- Symbols
- Graphics Text
- Drawing Defaults

Table 2-1 lists the escape code sequences used to control the drawing functions.

Table 2-1. Drawing Functions

<i>Display Control</i>	<i>Cursor Control</i>
<code>␣*dc</code> Turn on the graphics display	<code>␣*dk</code> Turn on (and select) the graphics cursor
<code>␣*dd</code> Turn off the graphics display	<code>␣*dl</code> Turn off the graphics cursor
<code>␣*de</code> Turn on the alphanumeric display	<code>␣*do</code> Move the graphics cursor to absolute coordinates
<code>␣*df</code> Turn off the alphanumeric display	<code>␣*dp</code> Move the graphics cursor to incremental coordinates
<code>␣*di</code> Set zoom size	<code>␣*dq</code> Turn on and select the alphanumeric cursor
<code>␣*dj</code> Set zoom position	<code>␣*dr</code> Turn off the alphanumeric cursor
	<code>␣*du</code> Define virtual/display graphics cursor
	<code>␣*dx</code> Set box cursor size

Table 2-1. Drawing Functions (Cont.)

<i>Plotting Commands</i>		<i>Area Fills</i>	
⌘*dm	Turn on rubberband mode	⌘*md	Define user area fill pattern
⌘*dn	Turn off rubberband mode	⌘*me	Absolute area fill
⌘*pa	Lift pen	⌘*mf	Relocatable area fill
⌘*pb	Lower pen	⌘*mg	Select area fill pattern
⌘*pc	Use graphics cursor as new data coordinate	⌘*mh	Select area boundary pen; enable/disable boundary mode
⌘*pd	Draw a point at the pen position	⌘*ps	Begin polygon area fill
<i>Vector Data Formats</i>		⌘*pa	(With ⌘*ps) Close polygon area fill; begin new area fill
⌘*pf	Use ASCII Absolute	⌘*pt	Close area fill
⌘*pg	Use ASCII Incremental	⌘*pu, pv	Lift/Lower boundary pen
⌘*ph	Use ASCII Relocatable	<i>Drawing Modes</i>	
⌘*pi	Use Binary Medium Absolute	⌘*ma	Select drawing mode
⌘*pj	Use Binary Short Incremental	<i>Symbols</i>	
⌘*pk	Use Binary Long Incremental	⌘*ms	Set symbol size
⌘*pl	Use Binary Long Relocatable	⌘*mt	Set symbol mode
⌘*pm	Use Binary Long Absolute	⌘*mu	Define display/virtual symbol
<i>Graphics Relocatable Origin</i>		<i>Graphics Text</i>	
⌘*mj	Set relocatable origin to absolute coordinates	⌘*ds	Turn graphics text on
⌘*mk, pe	Set relocatable origin to pen position	⌘*dt	Turn graphics text off
⌘*ml	Set relocatable origin to graphics cursor position	⌘*l	Graphics label
<i>Pens</i>		⌘*mm, ns	Set graphics text size
⌘*mx	Select primary pen	⌘*mn, nr	Set graphics text rotation
⌘*my	Select secondary pen	⌘*mo, mp	Set/turn off graphics text slant
<i>Line Types</i>		⌘*mq	Set graphics justification/origin
⌘*mb	Select line type	⌘*nm	Set graphics text margins
⌘*mc	Define user line pattern	⌘*nw	Set character spacing
		⌘*nx	Set graphics text pen
		<i>Drawing Defaults</i>	
		⌘*mr	Set (selected) graphics defaults

DISPLAY CONTROL

Graphics Video On/Off

You can turn the graphics part of the screen display on or off.

Turn On Graphics Display: `␣*dc`

Turn Off Graphics Display: `␣*dd`

These functions enable or disable the routing of data from raster memory to the terminal display. The alphanumeric display and raster memory data are unaffected.

Alphanumeric Video On/Off

You can turn the alphanumeric screen display on or off.

Turn On Alphanumeric Display: `␣*de`

Turn Off Alphanumeric Display: `␣*df`

These functions enable or disable the routing of data from alphanumeric memory to the terminal display. The graphics display and alphanumeric data are unaffected.

Graphics Zoom

Set zoom factor: `␣*d<x factor><y factor>i`

where `<x factor>`, `<y factor>` are in the range (.001, . . . , 63.999) (.004). If no `<y factor>` is specified, it defaults to `<x factor>`.

Set zoom center: `␣*d<x><y>j`

where `<x>`, `<y>` are virtual space coordinates in the range (−16383, . . . , 16383)

WARNING

Although −16383, . . . , +16383 is the legal range, an “illegal operation” error may occur if the window boundary exceeds these bounds.

NOTE: These escape sequences are compatible with other Hewlett-Packard graphics terminals. The terminal will ignore `␣*dg` (Zoom On) and `␣*dh` (Zoom Off) commands. More information on zooming can be found in Section 4. CREATING A PICTURE, “Views”.

CURSOR CONTROL

Alphanumeric Cursor

You can turn the alphanumeric cursor on and off as well as select its shape.

Turn Alphanumeric Cursor On: `␣*d<cursor type>q`

where <cursor type> = 0 Underline
1 Blob

If you do not specify a cursor type, the last type specified will be used. The terminal uses the underline as the initial cursor type.

Turn Alphanumeric Cursor Off: `␣*dr`

Graphics Cursor

The graphics cursor can be turned on or off. When the cursor is initially turned on, it is shown at the current pen position. You can optionally select the type of cursor (large crosshair, small crosshair, or box) to be displayed. If you do not specify a cursor type, the last type specified will be used. The initial cursor used by the terminal is the short crosshair.

Turn Graphics Cursor On: `␣*d<cursor type>k`

where <cursor type> = 0 Short crosshair (1" x 1")
1 Large crosshair (full screen)
2 Box

Turn Graphics Cursor Off: `␣*dl`

The thumbwheels can be used to move the short crosshair and box cursors and to change the intersection of the long crosshair cursor. The vertical thumbwheel moves the horizontal line of the long crosshair up and down, and the horizontal thumbwheel moves the vertical line sideways across the screen.

NOTE: If the box cursor is selected, its position is determined by its lower left corner.

Define Display/Virtual Graphics Cursor

You can use the graphics cursor to represent a position in either virtual space or display space. If the virtual cursor is selected, the cursor position will be incremented by virtual units. If the display cursor is selected, the cursor will be incremented by display units. (See the Move Cursor command below.)

Select Display/Virtual Graphics Cursor: `␣*d<mode>u`

where <mode> = 0 Display Cursor
1 Virtual Cursor

Set Box Cursor Size

You can control the x and y dimensions of the graphics box cursor. If the y dimension is not given, it will default to the x dimension. The default size of the box cursor is 16 x 16 display units.

Set Graphics Box Cursor Size: $\text{t}*\text{d}\langle\text{x}\rangle,\langle\text{y}\rangle\text{x}$

where $1 < \langle\text{x}\rangle,\langle\text{y}\rangle < 32767$ for the virtual box cursor

$1 < \langle\text{x}\rangle,\langle\text{y}\rangle < 32767$ for the display box cursor

Move Graphics Cursor

The graphics cursor can be positioned using either absolute or relative coordinates. These coordinates can refer to the display space or virtual space, depending upon the mode set in the Specify Display/Virtual Graphics Cursor command above.

Move Graphics Cursor Absolute: $\text{t}*\text{d}\langle\text{x}\rangle,\langle\text{y}\rangle\text{o}$

where $0 < \langle\text{x}\rangle,\langle\text{y}\rangle < 511$ Display Space
 $-16383 < \langle\text{x}\rangle,\langle\text{y}\rangle < 16383$ Virtual Space

The cursor will be moved to the indicated absolute location.

Move Graphics Cursor Incremental: $\text{t}*\text{d}\langle\text{x}\rangle,\langle\text{y}\rangle\text{p}$

where $-511 < \langle\text{x}\rangle,\langle\text{y}\rangle < 511$ Display Space
 $-32766 < \langle\text{x}\rangle,\langle\text{y}\rangle < 32766$ Virtual Space

The cursor will be displaced from its current location by the amount specified.

ERROR CONDITIONS. The values for $\langle\text{x}\rangle$ and $\langle\text{y}\rangle$ must be integers within the range for the selected space. If the selected space is virtual and you attempt to move the cursor outside of the active window, the cursor moves to the edge of the corresponding screen viewport and blinks. At this point the cursor is undefined in display coordinates.

If the selected space is the display and you move the cursor outside of the active viewport, the cursor position will be undefined in virtual units. For example, if the display cursor is moved outside the active viewport and rubberband line mode is turned on (see $\text{t}*\text{d}\text{m}$), the rubberband line will not be displayed.

If the values are out of range for the selected space or if more or less than two integer values are given, the terminal will ignore the command and set error bits.

PLOTTING COMMANDS

Graphic data is made up of vectors (line segments). There is no explicit "draw vector" command. Instead, the terminal uses the concept of a "pen" in drawing vector data.

The general format for a plotting sequence is:

$\epsilon * p$ <pen state> <x1> <y1> <x2> <y2> ... Z (or any capital letter)

where <pen state> indicates whether the pen is up or down, and the values are delimited by spaces or commas. The capital "Z" (a non-operative) terminates the sequence.

When enough parameters have been received to specify a data point, the pen is moved from its current position to the new end point. If the pen is down, a vector will be drawn. If the pen is up, the pen is moved to the new point (without drawing a vector) and lowered. In either case, the new point becomes the *current pen position*.

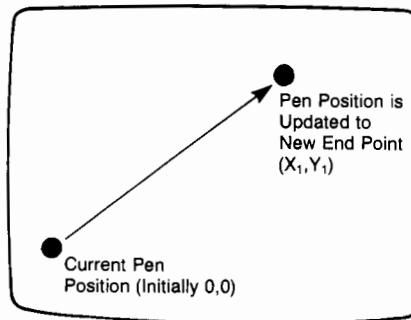


Figure 2-1. Current Pen Position and New End Point

Plotting sequences can extend indefinitely. In general, longer sequences are preferred as they minimize the overhead needed for a plot sequence. As the sequence length decreases, the percentage of prefix characters increases, and the drawing rate goes down. The worst possible case would be to send each vector in its own sequence; approximately 50% of the characters sent would be overhead, reducing vector speed by a factor of 2.

When a graphics hard reset is performed ($\epsilon * wr$), the pen is in the down state and positioned at virtual coordinates 0,0.

Lift Pen

L*pa

This command causes the imaginary plotting pen to be lifted from the drawing surface. Movement of the pen from the current position will not draw a line. The pen must be lowered (by supplying a coordinate or a lower pen command — L*pb) before a line can be drawn.

Example: Lift the pen, move it to 100,100, draw a vector to 200,200 and then to 50,300.

$\text{L*pa 100,100 200,200 50,300Z}$

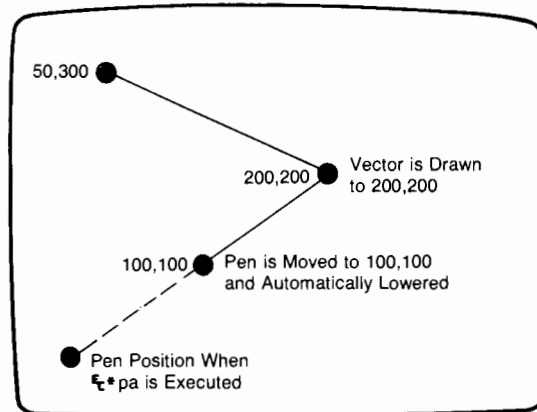


Figure 2-2. L*pa Example

NOTE: This command can also be used in conjunction with polygonal area fill commands. See "Polygonal Area Fills" in this section.

Lower Pen

L*pb

This command causes the imaginary plotting pen to be lowered to the drawing surface. Movement of the pen from the current position will draw a line.





Example: Draw a vector from the current pen position to 194,250.

L*pb 194 250Z

Use Cursor As Next Data Point

⌘*pc

This command causes the position of the virtual graphics cursor to be used as the next data point. The position is converted to an absolute location in the terminal virtual space. If the virtual cursor is undefined, the command is ignored, and an error is signaled.

Example: Position the cursor so it is visible on the screen. (Press , and  / ) Type ⌘*pac and continue to hold down the  key. Use the thumbwheels to manipulate the cursor and draw vectors on the screen.

Rubberband Line Mode

Turn Rubber Band Line On: ⌘*dm

Turn Rubber Band Line Off: ⌘*dn

“Rubberband Line” mode causes the terminal to display a temporary line connecting the virtual graphics cursor to the current pen position. As the cursor is moved (using the thumbwheels or move cursor commands), the temporary line will move, stretch, or contract as required to maintain the connection. The temporary line is “set” when the cursor position is entered as a new point (by executing the ⌘*pc command). The origin of the temporary “rubberband” line is then updated to the new point and the process may be repeated.

NOTE: If the graphics cursor is not already on, activating the rubberband line function will turn on the graphics cursor.

Draw A Point At The Current Pen Position

⌘*pd

This command draws a point at the current pen position. The pen is left in the “down” position. This command is ignored when encountered during an area fill sequence.

VECTOR DATA FORMATS

Vector coordinates are accepted in any one of the following formats:

- ASCII Absolute (default)
- ASCII Incremental
- ASCII Relocatable
- Binary Medium Absolute
- Binary Long Absolute
- Binary Short Incremental
- Binary Long Incremental
- Binary Long Relocatable

The data can be sent as a simple ASCII data or as packed binary bits. ASCII data is easy to generate and debug, binary data can be used for more efficient data transfers.

Vector plotting formats are specified in the command:

```
␣*p<format><pen state><x coordinate><y coordinate>...Z
```

The ␣*p<format> prefix should be sent for each escape sequence. Data formats can be changed at any time within a plotting sequence. If plotting commands are received and a data format has not been previously selected, the terminal will assume the data is in the ASCII absolute format.

NOTE: *Regardless of what plotting format is used to enter vector data, the terminal will store it as binary data. However, if the picture file is copied to a disc or datacomm file, the vector data is output as ASCII absolute.*

ASCII Formats

In the ASCII formats, coordinates are specified with the ASCII characters 0 through 9. Thus, numeric characters generated by a simple PRINT or WRITE statement can be used to specify X,Y pairs. The first value is used as the X coordinate, and the second as the Y coordinate. Spaces or commas must be used to separate the X and Y values. Extra spaces or commas are ignored. Digits following a decimal point are ignored (i.e., 123.456 is read as 123).

NOTE: Exponential notation cannot be used. If programs are used to generate data, the data must be in integer form. The number of characters needed to specify a single end point will depend on the magnitude of the coordinate values.

ASCII ABSOLUTE FORMAT

```
␣*pf
```

The values used in the ASCII absolute format can range between -16383 and 16383. Values outside of this range are ignored, and an error bit is set. Note that with the default window and viewport values (0,0 511,389 and 0,0 511,389 respectively), only points on the line where X is in the range 0 to 511 and Y is in the range 0 to 389 will be visible on the screen.

The following example draws vectors around the perimeter of the screen using ASCII absolute formatted data.

```
␣*pfa0,0 511,0 511,389 0,389 0,0Z
```

The `Ec*p` indicates a graphics plotting sequence follows. The "f" indicates that the data following will be in ASCII absolute format. If no format were indicated, ASCII absolute would be assumed. The "a" raises the pen. The pen is moved to 0 0, and lowered. Vectors are then drawn to 511,0 511,389 0,389 and back to 0,0. Note that the values are delimited by spaces or commas. The capital "Z" (a non-operative) terminates the sequence. Imbedded carriage return and line feed characters are ignored.

ASCII INCREMENTAL FORMAT

`t*pg`

To use ASCII incremental format, you specify a ΔX value and a ΔY value. These values are converted to integers, and the pen is moved by that amount relative to its current position. For example, to draw a square 100 units long and 50 units high from the current pen position, the following sequence could be used:

`t*pg100,0 0,50 -100,0 0,-50Z`

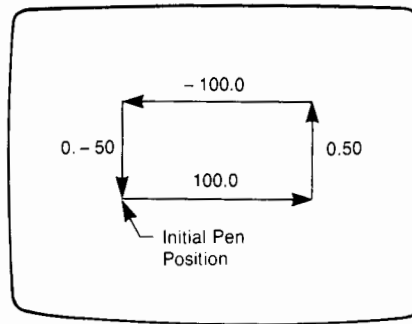


Figure 2-3. Example of ASCII Incremental Format

The drawing begins at the current pen position. The pen is moved 100 units to the right, 50 units up, 100 units to the left, and 50 units down. The same figure can be drawn at any screen location using the same set of data points by first positioning the pen to the desired starting point before sending the plotting data.

ASCII RELOCATABLE FORMAT

`t*ph`

The ASCII relocatable format allows you to use a relocatable origin to offset the incoming X and Y data values.

Using relocatable format you can easily use absolute data as if it were incremental merely by changing the relocatable origin. (See "Graphics Relocatable Origin" in this section.) The following example illustrates this technique.

Example: Draw a resistor symbol stored in absolute coordinates at screen locations 50,100 and 200,100.

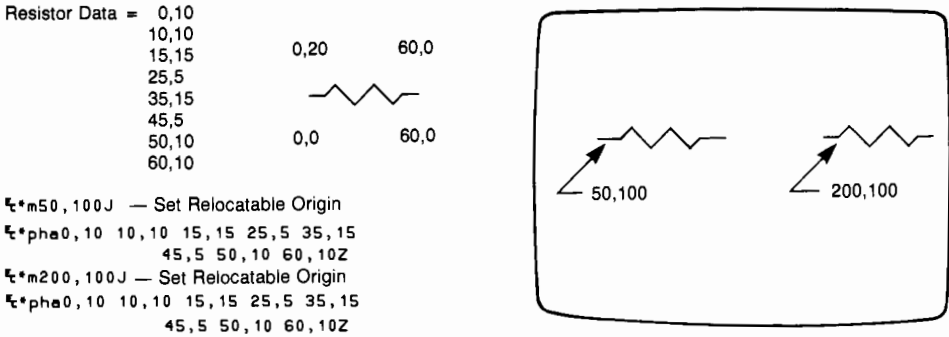


Figure 2-4. Example of ASCII Relocatable Format

Binary Formats

In binary format, all points are sent in a packed binary format. The coordinate values are sent using the bit patterns of ASCII characters. Table 2-2 contains the list of ASCII characters used to generate binary data. The number of characters required to specify a coordinate depends on the format used.

Table 2-2. ASCII Characters Used in Binary Data Formats

ASCII		ASCII	
Character	Bit Pattern	Character	Bit Pattern
SP	p01 0 0000	0	p01 1 0000
!	p01 0 0001	1	p01 1 0001
"	p01 0 0010	2	p01 1 0010
#	p01 0 0011	3	p01 1 0011
\$	p01 0 0100	4	p01 1 0100
@	p01 0 0101	5	p01 1 0101
&	p01 0 0110	6	p01 1 0110
'	p01 0 0111	7	p01 1 0111
(p01 0 1000	8	p01 1 1000
)	p01 0 1001	9	p01 1 1001
*	p01 0 1010	:	p01 1 1010
+	p01 0 1011	;	p01 1 1011
,	p01 0 1100	<	p01 1 1100
-	p01 0 1101	=	p01 1 1101
.	p01 0 1110	>	p01 1 1110
/	p01 0 1111	?	p01 1 1111

Drawing Fundamentals

The binary data can be short, medium, or long. The vectors are made up of 2, 4, or 6 bytes of coordinate information. The numbers are represented in a 5 bit per byte format.

Table 2-3. Binary Data Formats

Binary Format	Number of Bytes		Data Range						
Short	2		-16 to 15						
Medium	4		0 to 1023						
Long	6		-16383 to 16383						
<u>SHORT</u>									
BIT	7	6	5	4	3	2	1	0	
BYTE 1	p	0	1	X4	X3	X2	X1	X0	X value
BYTE 2	p	0	1	Y4	Y3	Y2	Y1	Y0	Y value
p = parity bit									
x4 = sign bit for x value									
y4 = sign bit for y value									
<u>MEDIUM</u>									
BIT	7	6	5	4	3	2	1	0	
BYTE 1	p	0	1	X9	X8	X7	X6	X5	High X value
BYTE 2	p	0	1	X4	X3	X2	X1	X0	Low X value
BYTE 3	p	0	1	Y9	Y8	Y7	Y6	Y5	High Y value
BYTE 4	p	0	1	Y4	Y3	Y2	Y1	Y0	Low Y value
p = parity bit									
<u>LONG</u>									
BIT	7	6	5	4	3	2	1	0	
BYTE 1	p	0	1	X14	X13	X12	X11	X10	High X value
BYTE 2	p	0	1	X9	X8	X7	X6	X5	Mid X value
BYTE 3	p	0	1	X4	X3	X2	X1	X0	Low X value
BYTE 4	p	0	1	Y14	Y13	Y12	Y11	Y10	High Y value
BYTE 5	p	0	1	Y9	Y8	Y7	Y6	Y5	Mid Y value
BYTE 6	p	0	1	Y4	Y3	Y2	Y1	Y0	Low Y value
p = parity bit									
x14 = sign bit for x value									
y14 = sign bit for y value									

Table 2-4. Conversion Table for Binary Medium Absolute Format (Continued)

NOTE: ■ indicates a "space" character; every coordinate address must consist of two characters.

	0	1	2	3	4	5	6	7	8	9		0	1	2	3	4	5	6	7	8	9	
600	28	29	2:	2;	2<	2=	2>	2?	3■	3!	800	9■	9!	9"	9#	9\$	9%	9&	9'	9(9)	
610	3"	3#	3\$	3%	3&	3'	3(3)	3*	3+	810	9*	9+	9,	9-	9.	9/	90	91	92	93	
620	3,	3-	3.	3/	30	31	32	33	34	35	820	94	95	96	97	98	99	9:	9;	9<	9=	
630	36	37	38	39	3:	3;	3<	3=	3>	3?	830	9>	9?	9■	9!	9"	9#	9\$	9%	9&	9'	
640	4D	4!	4"	4#	4\$	4%	4&	4'	4(4)	840	:(:)	!*	!+	!;	!-	!.	!/	!0	!1
650	4*	4+	4,	4-	4.	4/	40	41	42	43	850	:2	:3	:4	:5	:6	:7	:8	:9	::	::;	
660	44	45	46	47	48	49	4:	4;	4<	4=	860	:<	:=	:>	:?	:■	:!	:"	:#	:\$:%	
670	4>	4?	5D	5!	5"	5#	5\$	5%	5&	5'	870	;&	;'	;(;) ;*	;;	;-	;. ;/	;			
680	5(5)	5*	5+	5,	5-	5.	5/	50	51	880	;0	;1	;2	;3	;4	;5	;6	;7	;8	;9	
690	52	53	54	55	56	57	58	59	5:	5;	890	;;	;;	;<	;;	;;	;;	;;	;<	;;	;<	;<
700	5<	5=	5>	5?	6■	6!	6"	6#	6\$	6%	900	<\$	<%	<&	<'	<(<)	<*	<+	<,	<-	
710	6&	6'	6(6)	6*	6+	6,	6-	6.	6/	910	<.	</	<0	<1	<2	<3	<4	<5	<6	<7	
720	60	61	62	63	64	65	66	67	68	69	920	<8	<9	<:	<;	<<	<=	<>	<?	=■	=!	
730	6:	6;	6<	6=	6>	6?	7D	7!	7"	7#	930	="	=#	=\$	=%	=&	='	=(=)	=*	=+	
740	7\$	7%	7&	7'	7(7)	7*	7+	7,	7-	940	=,	=-	=.	=/	=0	=1	=2	=3	=4	=5	
750	7.	7/	70	71	72	73	74	75	76	77	950	=6	=7	=8	=9	=:	=;	=<	==	=>	=?	
760	78	79	7:	7;	7<	7=	7>	7?	8D	8!	960	>■	>!	>"	>#	>\$	>%	>&	>'	>(>)	
770	8"	8#	8\$	8%	8&	8'	8(8)	8*	8+	970	>*	>+	>,	>-	>.	>/	>0	>1	>2	>3	
780	8,	8-	8.	8/	80	81	82	83	84	85	980	>4	>5	>6	>7	>8	>9	>:	>;	><	>=	
790	86	87	88	89	8:	8;	8<	8=	8>	8?	990	>>	>?	?■	?!	?"	?#	?\$?%	?&	?'	
											1000	?(?)	?*	?+	?,	?-	?.	?/	?0	?1	
											1010	?2	?3	?4	?5	?6	?7	?8	?9	?:	?;	
											1020	?<	?=	?>	??							

BINARY LONG ABSOLUTE FORMAT

␣*pm

Binary long absolute format uses three bytes for each coordinate value. The values must be between -16383 and 16383. Two's complement is used for negative numbers.

Example: Draw a vector from 0,0 to 1600,1856 using Binary Absolute Format.

Coordinate values: $X_0=0, Y_0=0$

$X_0 = 00000\ 00000\ 00000\ Y_0 = 00000\ 00000\ 00000$

BYTE 1 = 01 00000 = SPACE (High X)
 BYTE 2 = 01 00000 = SPACE (Mid X)
 BYTE 3 = 01 00000 = SPACE (Low X)



BYTE 4 = 01 00000 = SPACE (High Y)
 BYTE 5 = 01 00000 = SPACE (Mid Y)
 BYTE 6 = 01 00000 = SPACE (Low Y)

Coordinate values: $X_1=1600, Y_1=1856$

$X_1 = 00111\ 11010\ 00000\ Y_1 = 00000\ 11101\ 00000$

BYTE 1 = 01 00111 = ' (High X)
 BYTE 2 = 01 11010 = : (Mid X)
 BYTE 3 = 01 00000 = SPACE (Low X)

BYTE 4 = 01 00000 = SPACE (High Y)
 BYTE 5 = 01 11101 = = (Mid Y)
 BYTE 6 = 01 00000 = SPACE (Low Y)

␣*p m a SP SP SP SP SP SP ' : SP SP = SP Z
 X₀ Y₀ X₁ Y₁

The ␣*p prefix selects a plotting sequence. The "m" specifies that the binary long absolute data format will be used. The "a" raises the pen up. The first 6 bytes (all space characters) move the raised pen to 0,0 where the pen is lowered. The next 6 bytes (' : SP SP = SP) specify the point 1600,1856. The pen is moved to the second point, and a vector is drawn. The "Z" terminates the escape sequence. Note that space characters or commas cannot be used to separate data in binary formats since they would be interpreted as data.

If a parameter byte is lost or garbled in transmission, all of the following end points will be improperly read. To minimize data errors caused by the loss of a data byte, any plotting command can be used to reset the parameter count and restore synchronization. Nops (z), redundant format, or pen down commands can also be inserted to insure synchronization if necessary.

BINARY SHORT INCREMENTAL FORMAT

⌘*pj

Binary short incremental format uses one byte for each coordinate value. The values must be between -16 and 15. The two's complement of a negative value is used.

Example. Move the pen from its current position -12 units in the X direction and 6 units in the Y direction using the Binary Short Incremental format.

X = -12 = 10100

Y = 6 = 00110

BYTE 1 = 01 10100 = 4

BYTE 2 = 01 00110 = &

⌘*p j 4 & Z
 | |
 XY

BINARY LONG INCREMENTAL FORMAT

⌘*pk

Binary long incremental format uses three bytes for each coordinate value, and is used for values between -16383 and 16383. The two's complement of a negative value is used.

Example. Move the pen from its current position -400 units in the X direction and 100 units in the Y direction.

Δ X = -400 = 11111 10011 10000

Δ Y = 100 = 00000 00011 00100

BYTE 1 = 01 11111 = ? (High X)

BYTE 2 = 01 10011 = 3 (Mid X)

BYTE 3 = 01 10000 = 0 (Low X)

BYTE 4 = 01 00000 = SPACE (High Y)

BYTE 5 = 01 00011 = # (Mid Y)

BYTE 6 = 01 00100 = \$ (Low Y)

⌘*p k ? 3 0 SPACE # \$ Z
 └───┬───┘ └───┬───┘
 X Y

BINARY LONG RELOCATABLE FORMAT

␣*p1

Binary long relocatable format uses three bytes for each coordinate value. The values must be between -16383 and 16383.

Example. Move the pen from its current position to a position -600 units on the X axis and 200 units on the Y axis from the current relocatable origin.

X = -600 = 11111 01101 01000

Y = 200 = 00000 00110 01000

BYTE 1 = 01 11111 = ? (High X)

BYTE 2 = 01 01101 = - (Mid X)

BYTE 3 = 01 01000 = ((Low X)

BYTE 4 = 01 00000 = SPACE (High Y)

BYTE 5 = 01 00110 = & (Mid Y)

BYTE 6 = 01 01000 = ((Low Y)

␣*p a 1 ? - (SPACE & (Z
 X Y

Mixing Data Formats

There are no restrictions on mixing data formats. Simply specify the new format to be used, and follow it with data in the new format. Note that by restricting data values to binary values between 32 and 63, (the printing graphic characters and numbers), the plotting commands "a-z" can be intermixed in the binary data without confusion.

Example. Move the pen to 360,180 in ASCII absolute format, then draw a box 10 units wide by 5 units wide using binary short incremental.

␣*p a f 360,180 j * SP SP % 6 SP SP ; Z
 1 2 3 4 5

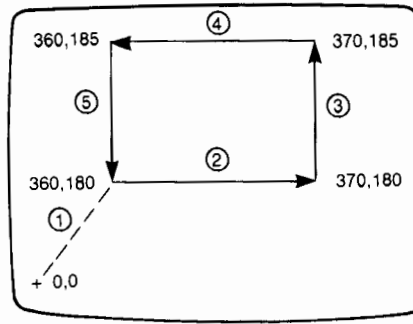


Figure 2-5. Example of Mixed Data Formats

GRAPHICS RELOCATABLE ORIGIN

The relocatable origin is a means of adding an offset to the x and y coordinate values of data points. The value of the relocatable origin is added to the value of the x and y coordinates sent by the host computer.

The relocatable origin allows you to use one set of data and drawing commands to display a figure at several different positions. The value of the relocatable origin is added to the relocatable data to obtain the coordinates used to draw the figure.

The terminal provides three commands for setting the graphics relocatable origin:

- Set Relocatable Origin In Absolute Coordinates
- Set Relocatable Origin To Current Pen Position
- Set Relocatable Origin to Current Cursor Position

Set Relocatable Origin In Absolute Coordinates

$\text{t}^*m\langle x \rangle, \langle y \rangle j$

where $-16383 < \langle x \rangle, \langle y \rangle < 16383$

This command sets the relocatable origin to the coordinates given in $\langle x \rangle$ and $\langle y \rangle$. The values of $\langle x \rangle$ and $\langle y \rangle$ are in the absolute data format. Once the relocatable offset has been added, the resultant data is stored in the terminal in absolute format.

Set Relocatable Origin To Current Pen Position

t^*mk OR t^*pe

This command sets the relocatable origin to the current position of the graphics pen. This allows you to position the pen using absolute, incremental, or relocatable data and then plot a figure at that location using relocatable data.

Set Relocatable Origin To Current Graphics Cursor Position

`Esc*m1`

This command takes the absolute coordinates of the virtual graphics cursor position as the relocatable origin.

NOTE: If the virtual graphics cursor is undefined, this command sets an error bit.

PENS

Each vector is assigned one of 16 pens (0-15). The color of the pen depends upon the palette in use. (See Section 3. COLOR, for more information on pens and palettes.)

Pen numbers are assigned using the escape sequences:

Set primary pen: `Esc*m<pen number>x`

Set secondary pen: `Esc*m<pen number>y`

where `<pen number>` is an integer in the range (0, . . . , 32767); the low four bits of the binary value determine the pen.

These assignments remain in effect until they are explicitly changed or the drawing defaults are reassigned.

Primary pens are used to draw most lines and area fills. Secondary pens are used in conjunction with specially defined line types and areas fills (depending upon the drawing mode in use). See "Drawing Modes" in this section.

LINE TYPES

You can select the pattern to be used when drawing vectors. Patterns can be selected from a predefined set, or you can define your own pattern. This feature may be used for distinguishing between different groups of plotted data or for use in such applications as engineering drawings, graphs, or fabric patterns.

Selecting A Line Type

The terminal allows you to choose a line pattern for your vectors. You select a line type using the command:

```
ℓ*m <type> b
```

where <type> is shown in figure 2-6.

1 Solid line (default)	1 = _____
2 User defined line pattern	
3 Current area fill pattern	
4 Predefined pattern	4 = - . - . - . - . -
5 Predefined pattern	5 = _____
6 Predefined pattern	6 = - - - - -
7 Predefined pattern	7 =
8 Predefined pattern	8 = - . - . - . - . -
9 Predefined pattern	9 =
10 Predefined pattern	10 = - . - . - . - . -
11 Predefined pattern	11 = . (POINT PLOT)

Figure 2-6. Predefined Line Type Patterns

Once a line type has been selected, all subsequent vectors are drawn using that line type. You can select a vector line type to be a solid line, a user defined line pattern, the current area fill pattern, one of seven predefined dot patterns, or a point plot. Point plot causes a single point to be plotted at the vector end points. This line type is useful for generating "scattergram" type graphs.

NOTE: Symbols may also be used for plotting vector end points. See "Symbols" in this section.

Example. Select line type 9, and draw a figure using the new line type.

```
ℓ*m 9B
ℓ*p a 0,0 200,0 200,100 0,100 0,0Z
```

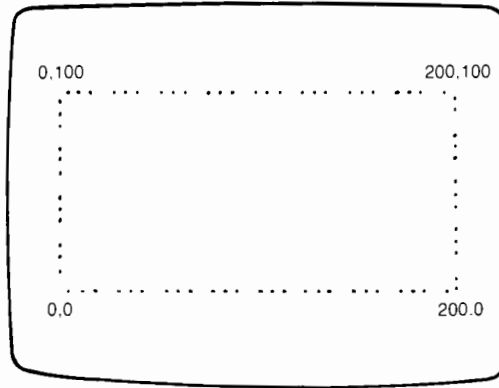


Figure 2-7. Plotting With Line Type 9

User Defined Line Types

The dot pattern used to draw vectors can be defined programmatically. Once a pattern is defined you must select "user defined line" as the line type (type=2) with the Select Line Type command.

A user defined line pattern is made up of a dot pattern and a scale factor. The dot pattern is a sequence of eight 1's and 0's. Using the default drawing mode, points indicated as a "1" in the patterns are drawn using the primary pen, and points indicated as a "0" are left unchanged. (See "Drawing Modes" in this section.)

The pattern is given as a decimal number between 0 and 255 that is the decimal equivalent of the 8-bit binary pattern. The default pattern is all "1"s (255). For example, = $10101010_B = 170$. The actual number used for the pattern can be between -32768 and 32767. The least significant 8 bits of the number's 2's complement equivalent are used to determine the pattern.

The scale factor indicates how many times each bit in the pattern should be repeated. The default scale factor is 1. For example, a scale factor of 3 applied to the pattern defined above would result in a pattern of or 111000111000111000111000_B.

The command for creating a user defined line type is:

```
⌘*m<pattern><scale>c
```

where <pattern> is an integer in the range (-32768 to 32767)
and <scale> is an integer in the range (1 to 255)

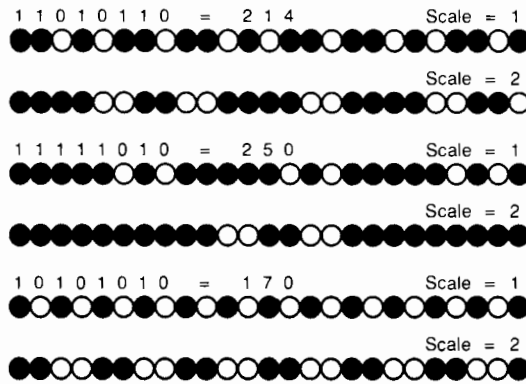


Figure 2-8. Examples of User Defined Line Patterns

Example. Define a pattern to generate the following vector:

```
11111111110011001111111111001100
```

```
pattern = 11111010 = 250
scale   = 2
```

```
^*m 250 2 C
```

AREA FILLS

The terminal has two types of area fill specifications, rectangular and polygonal. An area can be filled with one of a variety of predefined patterns or with a user defined pattern. The pattern can also be used to provide line patterns for horizontal or vertical lines when the area pattern is selected as the line type. (Refer to "Using Area Fills As Line Types".)

When an area fill pattern is selected, the entire virtual space is divided up into 8x8 cells. Every location is mapped to the corresponding bit in the pattern. When an area fill operation is performed, the area fill pattern is replicated to fill the area. The pattern starts at virtual coordinates 0,0 and is repeated in both the positive and negative directions as shown in figure 2-9.

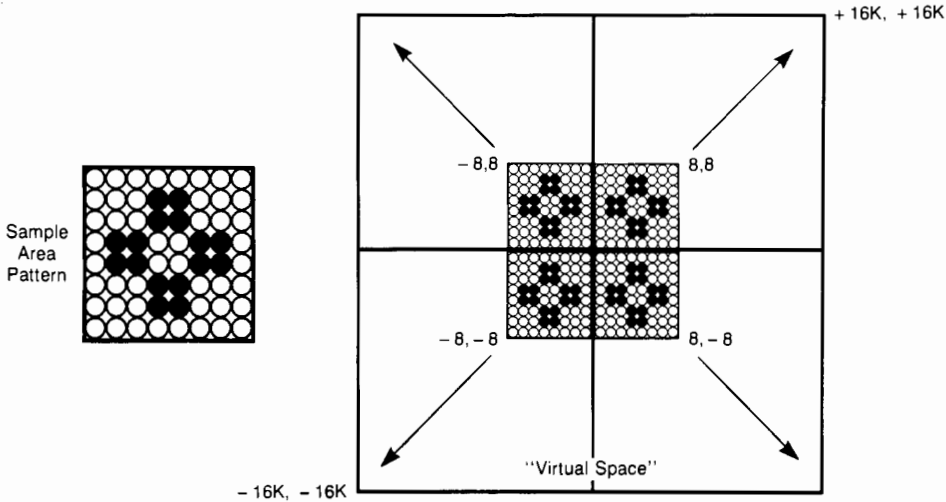


Figure 2-9. How the Area Fill Pattern is Mapped

Area fill patterns are unaffected by zooming, panning, or object transformations.



Figure 2-10. "Zoomed" Area Fill

Selecting An Area Fill Pattern

```
^*m <pattern> g
```

- where <pattern> = 1 Solid area pattern (default)
- 2 User defined area pattern
- 3 Predefined area pattern
- 4 Predefined area pattern
- 5 Predefined area pattern
- 6 Predefined area pattern
- 7 Predefined area pattern
- 8 Predefined area pattern
- 9 Predefined area pattern

When an area fill command is executed, it uses either a predefined area fill pattern or a user defined area fill pattern. The default pattern is a solid area fill.

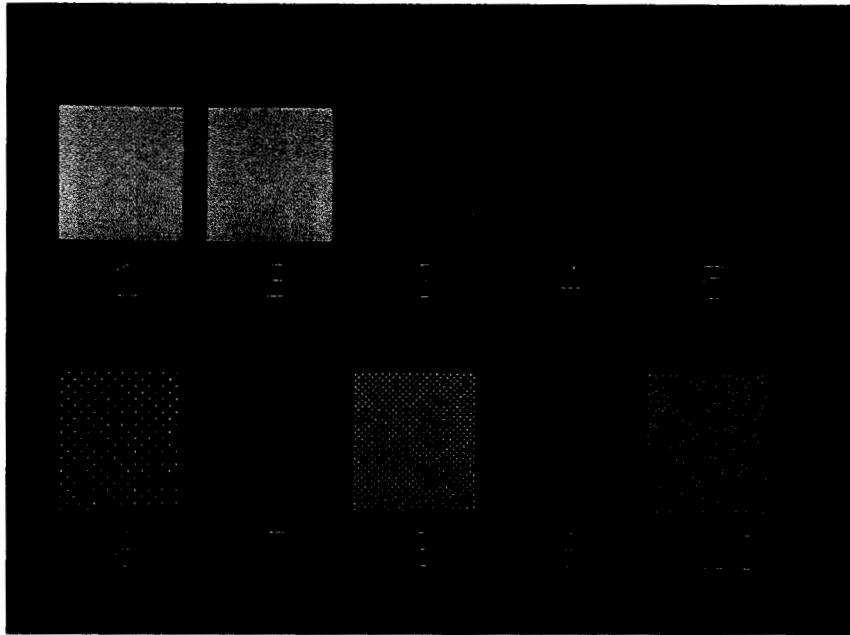


Figure 2-11. Predefined Area Fill Patterns

User Defined Area Fill Patterns

The user area pattern is defined using 8 parameters, one for every row of dots in the pattern. Each parameter is an integer in the range -32768 to 32767 . The number is interpreted as a 2's complement number, and the least significant 8-bits are used to obtain a value between 0 and 255. The 8-bit number (0 to 255) represents an 8-bit binary pattern. Bits set to '1' are drawn using the primary pen, and bits set to '0' are not drawn (depending upon the drawing mode in use).

The command which defines a user area fill pattern is:

```

t*m <row 0> <row 1> ... <row 7> d
    
```

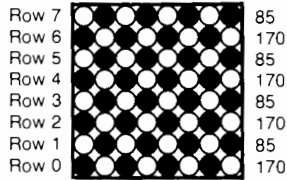
where <row 0> is the 8-bit pattern for row 0

·
·
·

<row 7> is the 8-bit pattern for row 7

Example. Define a simple checkerboard pattern.

Row 0 = 10101010 = 170
 Row 1 = 01010101 = 85
 Row 2 = 10101010 = 170
 Row 3 = 01010101 = 85
 Row 4 = 10101010 = 170
 Row 5 = 01010101 = 85
 Row 6 = 10101010 = 170
 Row 7 = 01010101 = 85



```

t*m 170 85 170 85 170 85 170 85 D
    |
row 0
    |
row 7
    
```

NOTE: The scale factor of an area fill pattern is always '1', unlike other Hewlett Packard graphics terminals.

Other examples of user defined area patterns are shown in figure 2-12.

Using Area Fill Patterns as Line Types

If you select type 3 for the Line Type command, the line pattern is created from the current area fill pattern. Horizontal and vertical lines are drawn using the appropriate row or column from the area fill pattern. Diagonal lines are always drawn using a solid vector; they do not follow the area fill pattern. If a line is longer than 8 dots, the pattern is used over and over to complete the vector.

Example. Plot three vectors (2,3)-(7,3), (9,3)-(9,12), and (7,5)-(2,10) using a user defined area fill pattern of 51, 204, 51, 204, 51, 204, 51, 204.

Step 1. Create a user defined area fill pattern.

```
t*m51,204,51,204,51,204,51,204D
```

Step 2. Select the user defined area fill pattern as the current area fill pattern.

```
t*m2G
```

Step 3. Select the current area fill pattern as the line type.

```
t*m3B
```

Step 4. Draw the vectors.

```
t*pa2,3 7,3a 9,3 9,12a 7,5 2,10 Z
```

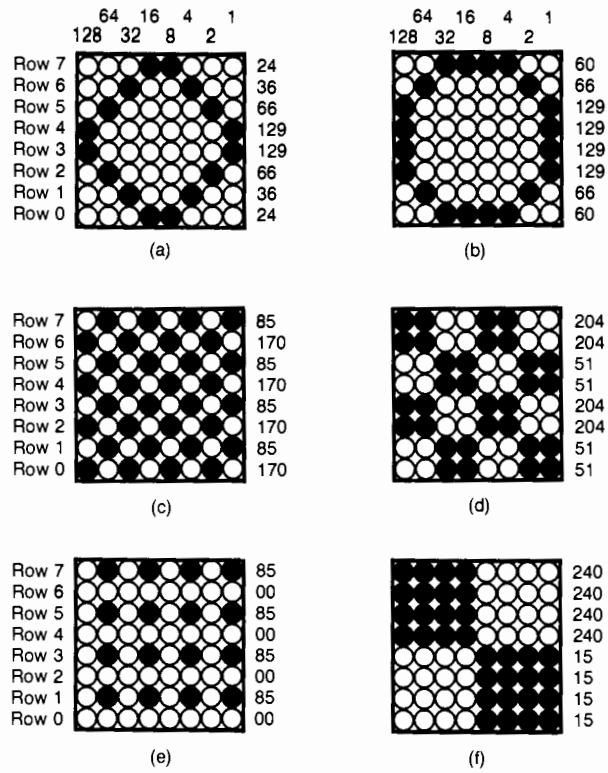


Figure 2-12. Examples of User Defined Area Fill Patterns

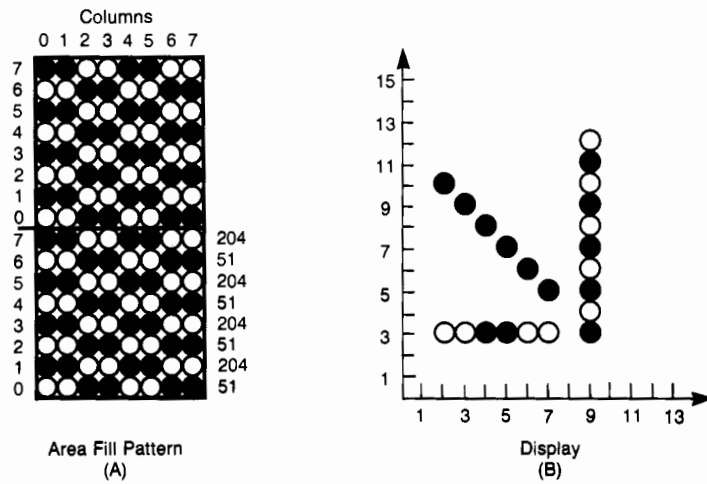


Figure 2-13. Using Area Fill Patterns As Line Types

Rectangular Area Fills

A rectangular area can be filled in with a pattern simply by sending the lower left and upper right coordinates in an escape sequence. The coordinates can be either in absolute or relocatable ASCII format. The pattern used for the area fill is determined by the Select Area Pattern command.

NOTE: The terminal can also fill irregular polygons. See "Polygonal Area Fills" in this section.

FILL RECTANGLE, ABSOLUTE

```
␣*m <x1> <y1> <x2> <y2> e
```

where <x1> <y1> are the absolute coordinates of the lower left corner of the rectangular area to be filled (−16383 to 16383), and

<x2> <y2> are the absolute coordinates of the upper right corner of the rectangular area to be filled (−16383 to 16383).

Example. Using area fill pattern 5, fill a rectangle defined by the diagonal 50,50 300,300.

```
␣*m 5g 50,50 300,300E
```

FILL RECTANGLE, RELOCATABLE

```
␣*m <x1> <y1> <x2> <y2> f
```

where <x1> <y1> are the relocatable coordinates of the lower left corner of the rectangular area to be filled (−32766 to 32766), and

<x2> <y2> are the relocatable coordinates of the upper right corner of the rectangular area to be filled (−32766 to 32766).

This command is used in conjunction with the relocatable origin.

Example. Load a function key with the command:

```
␣*m1 20,20 30,30 F
```

Use the thumbwheels to move the cursor while periodically pressing this function key.

Polygonal Area Fills

Begin Polygon Area Fill: `⌘*ps`
Close Polygon/ Begin New Polygon: `⌘*pa`
Close Polygon Area Fill: `⌘*pt`

The terminal allows you to define any polygon, and fill it with the current area fill pattern. The Begin Polygon Area Fill command causes subsequent coordinate pairs to be read as vertices of the polygon. When a lift pen command (`⌘*ps`) occurs in the middle of a polygon area fill sequence, a new polygon is started. (See example.) The Close Area Fill command (or any capital letter) causes the polygon to be filled using the current drawing mode, area pattern, area boundary color, and pen. Note that it is not necessary to specify a vector from the last point back to the first point; the polygon automatically closes itself at the end of the sequence.

If the polygon definition crosses over itself, the areas will be filled in an alternate order.

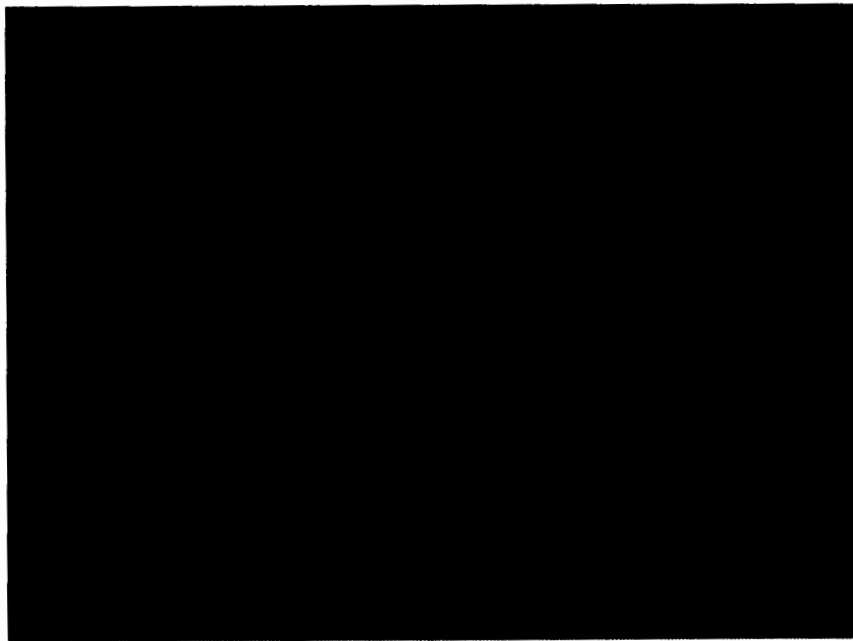


Figure 2-14. Overlapping Polygon Area Fills

Example. Move the pen to 33 0, and define and fill a pentagon 100 units on a side. Lift and move the pen to 40 10, and define another pentagon inside the first pentagon.

```

t*p a s 33,0 133,0 166,95 83,150 0,95
      a 40,10 12,91 83,138 153,91 125,10T

```

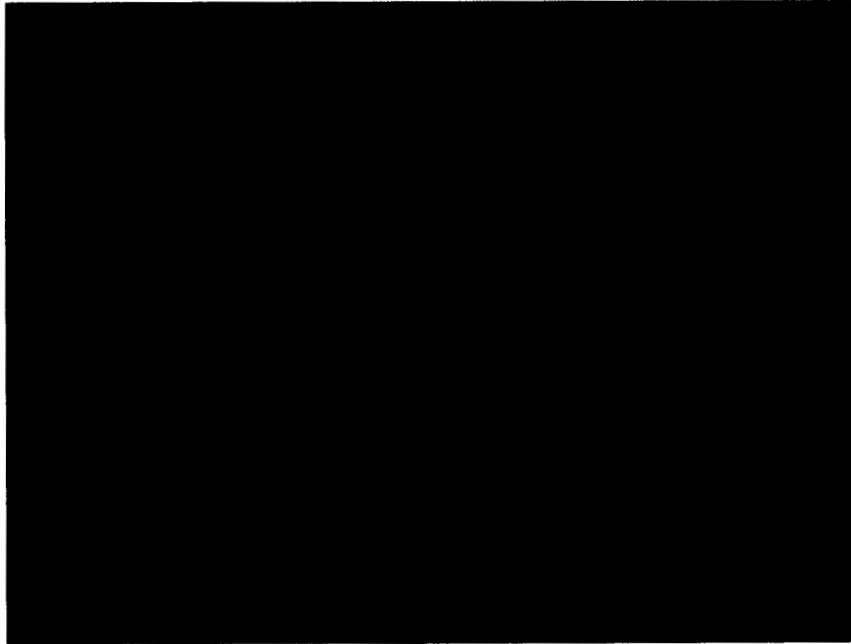


Figure 2-15. Polygon Area Fill Example

Area Boundary Pen

SELECT AREA BOUNDARY PEN. You can select the pen to be used to draw the boundary of an area fill. When a polygon area fill is performed, the vectors that are used to define the area are drawn using this pen. If the boundary pen is not defined, the edges of the boundary are the same color and pattern as the interior.

Set Area Boundary Pen: `t*m<pen number>h`

where <pen number> is an integer in the range (-32768, . . . , 32767); the low four bits are used to select a pen in the range (0, . . . ,15).

Disable Boundary Pen: `t*mh`

LIFT/LOWER BOUNDARY PEN

Lift Boundary Pen: `␣*pu`

Lower Boundary Pen: `␣*pv`

If a boundary pen has been defined by the `␣*mh` command (above), the pen may be “lifted” so that the edges of the polygon will be the same color and pattern as the interior. If the pen is “lowered”, the boundaries of the fill will be drawn as a solid vector using the `<pen number>` supplied in the `␣*mh` command.

NOTE: If the `␣*mh` command was executed without supplying a `<pen number>`, the `␣*pv` command has no effect.

DRAWING MODES

The terminal has eight drawing modes which affect the way in which raster pixels are set when a vector or area fill pattern is drawn. Using the default mode (JAM1), the following rules apply:

- If a bit in the pattern = 1, the corresponding raster pixel is drawn in the primary pen.
- If a bit in the pattern = 0, the corresponding raster pixel is not affected.

Drawing modes are set using the command:

`␣*m<mode>a`

where	<mode>	meaning
	0	NOP (Non-operative)
	1	CLEAR1
	2	JAM1
	3	COMP1
	4	JAM2
	5	OR
	6	COMP2
	7	CLEAR2

These modes are discussed in detail in Section 6. RASTER MANIPULATIONS, “Vector Drawing Modes”.

SYMBOLS

When symbol mode is enabled, the active symbol is plotted at each vector end point. The terminal has 16 predefined symbols and 1 default symbol, as shown in figure 2-16.

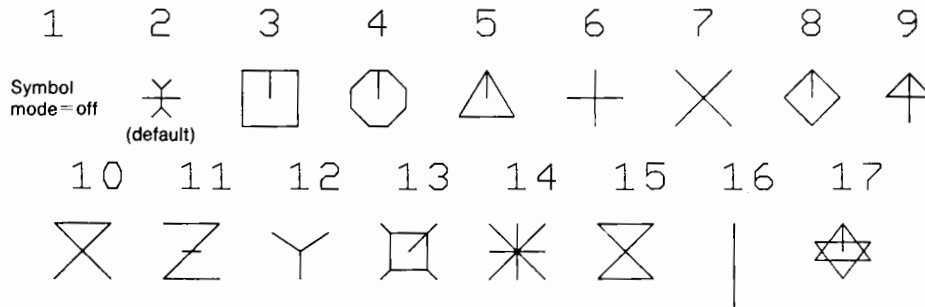


Figure 2-16. System Symbols

NOTE: A user defined character may also be used as a symbol. See Section 4. CREATING A PICTURE, “User Defined Characters and Fonts” for information on creating a user defined character.

Setting Symbol Mode

Step 1. If you want to use a user defined symbol, you must activate it by using the command:

```
⌘*n<character code>,<font number>u
```

where <character code> = (32, . . .,126); and = (3, . . ., 14)

(See Section 4. CREATING A PICTURE, “User Defined Characters and Fonts” for details.) If you have not selected a user defined symbol and later select mode = 2, the default symbol shown in figure 2-16 will be used.

Step 2. Specify the symbol mode for a user defined symbol or predefined symbol. If you specify a mode >17 the last selected mode will remain in effect.

```
⌘*m<mode>t
```

where <mode> symbol used

- | | |
|----|-----------------------------------|
| 1 | none, symbol mode is turned off |
| 2 | user defined symbol set with ⌘*nu |
| 3 | predefined symbol |
| . | . |
| . | . |
| . | . |
| 16 | predefined symbol |

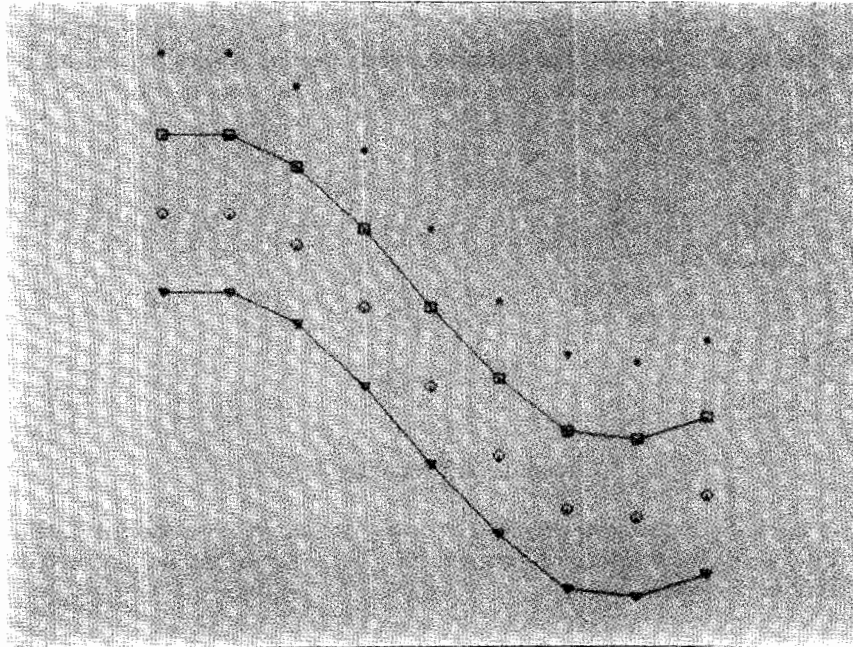


Figure 2-17. Plotting With Symbols

Symbol Attributes

SYMBOL SIZE

$\xi * m \langle size \rangle s$

where $\langle size \rangle$ is a real number in the range $(-127.996, \dots, 127.996)(.004)$

This command specifies the size of the symbol as a multiplier of the character definition cell. For system symbols, the cell is 7×10 character definition units.

DEFINE DISPLAY/VIRTUAL SYMBOL. You can define Display or Virtual Symbols with the command:

$\xi * m \langle mode \rangle u$

where mode = 0 Virtual Symbol
1 Display Symbol

Display symbols (also referred to as markers) are not subject to object transformations or zoom operations in that their size and orientation remains the same. Their location on the screen, however, will be affected by the transformations undergone by the vector.

GRAPHICS TEXT

The terminal has two system fonts used for graphics text, a standard font and an associated extended roman font, and will store up to 14 user defined fonts, depending upon the amount of available vector memory. The discussion of user defined fonts is left until Section 4. *CREATING A PICTURE*. The following discussion of graphics text applies regardless of whether you are using the system font or a user defined font.

This discussion will explain the following aspects of graphics text:

- How to enter graphics text mode
- How to create a graphics label
- Graphics text attributes

Graphics Text Mode

Graphics text mode lets you enter text directly from the keyboard, disc, or datacomm. The system font will be used unless you have specified a user defined font.

Graphics text mode can be turned on or off using the commands:

Turn graphics text mode on: `␣*ds`

Turn graphics text mode off: `␣*dt`

The backspace, carriage return, and line feed functions work as they do when entering alphanumeric data (even on inverted text), therefore, it is easy to add or edit titles and labels. You may also use CNTL-J (Tab 1 Character Down), CNTL-K (Tab 1 Character Up), CNTL-I (Tab 1 Character Right) and CNTL-M (Tab 1 Character Left).

NOTE: The cursor control keys also work in graphics text mode. However, unlike previous Hewlett-Packard graphics terminals, the cursor direction is determined by text rotation. Also, the backspace function does not erase the character.

The positioning of graphics text when this mode is enabled is determined by the following rules:

1. Text appears at the last cursor or pen change.
2. If the cursor was moved, but not turned on, the text will appear at the current pen position.

Example. Turn on graphics text mode, and position cursor at 100,100.

```
␣*dsk 100,100 □ ← position cursor at 100,100
      ↑
      turn on graphics cursor
```

Graphics Label

A graphics label is created using the `ESC+l` command. Characters following the `ESC+l` prefix are written in graphics text at the current pen position. The system font will be used unless you have specified a user defined font.

`ESC+l<text label><terminator>`

where `<terminator>` = `ESC`, `LF`, `ESC LF`, `LF ESC`, `ESC ESC`, `LF LF`, `ESC`, and the text string can be a maximum of 80 characters long.

The label must end with one of the terminators in the braces. A `ESC` moves the pen to its original position when the label command was first received. A `LF` moves the pen down one line (a "line" is determined by the height of the space character of a font). Note that the actual direction moved following a `ESC` or `LF` depends on the text orientation selected. (See discussion on text margins.) If an `ESC` character is used to terminate the label, a new escape sequence is initiated.

Example. `ESC+l This is a sample label ESC LF`

Graphics Text Attributes

Both the system font and user fonts may have the following attributes shown in figure 2-18 applied to them:

Justification	<code>ESC*mq</code>
Origin	<code>ESC*mq</code>
Margins	<code>ESC*nm</code>
Spacing	<code>ESC*nw</code>
Size	<code>ESC*mm,ns</code>
Rotation	<code>ESC*mn</code>
Slant	<code>ESC*mo,mp</code>
Color	<code>ESC*nx</code>

TEXT size

TEXT SLANT



Figure 2-18. Graphics Text Attributes

TEXT SIZE. There are two escape sequences which specify text size. The first specifies size as a multiple of the size of the character within the *character cell*. System font characters are contained within cells of 7×10 virtual units.

$$\text{\texttt{\textbackslash}m\langle x \text{ multiplier} \rangle \langle y \text{ multiplier} \rangle m}$$

where $\langle x \text{ multiplier} \rangle$, $\langle y \text{ multiplier} \rangle$ are real numbers in the range $(-16383.996, \dots, +16383.996)(.004)$. If $\langle y \text{ multiplier} \rangle$ is not specified, it defaults to $\langle x \text{ multiplier} \rangle$.

The second command sets the absolute size of the space character cell.

$$\text{\texttt{\textbackslash}n\langle x \text{ size} \rangle \langle y \text{ size} \rangle s}$$

where $\langle x \text{ size} \rangle$, $\langle y \text{ size} \rangle$ are integers in the range $(-16383, \dots, +16383)$. If $\langle y \text{ size} \rangle$ is not specified, it defaults to $\langle x \text{ size} \rangle$.

This command is translated and stored as an $\text{\texttt{\textbackslash}mm}$ command according to the following formula:

$$\langle x \text{ size} \rangle / \text{width of space character} = \langle x \text{ multiplier} \rangle$$

$$\langle y \text{ size} \rangle / \text{height of space character} = \langle y \text{ multiplier} \rangle$$

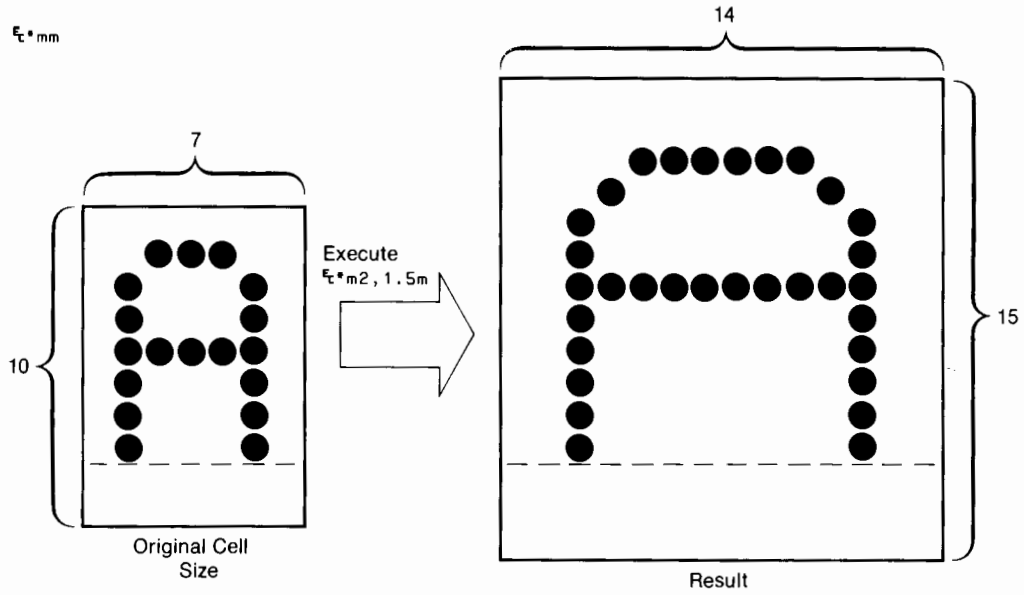
Therefore, when you execute an $\text{\texttt{\textbackslash}ns}$ command, it is stored as:

$$\text{\texttt{\textbackslash}m\langle x \text{ multiplier} \rangle \langle y \text{ multiplier} \rangle m}$$

The x and y multipliers are then used to determine the sizes of the other characters in the font. Note that if all the other character cells in the font are the same size as the space cell (as in the system font), the calculation of the other character cell sizes is trivial. However, you should be aware of this formula if you create user defined fonts with differing cell sizes.

Figure 2-19 illustrates the differences between the two size commands. Figure 2-20 shows the effect of a negative size value.

Drawing Fundamentals



To achieve the same effect with the E^*ns command...

Send $E^*n14, 15s$

Sets Absolute size of space character to 14 X 15.

Assuming an original size of 7 x 10 for the space character, the sizes of the other characters are determined by: $E^*m(14/7), (15/10)m = E^*m(2), (1.5)m$

Figure 2-19. Graphics Text Size

E^*m-11M E^*m11M

nut fun

nut fun

$E^*m-1-1M$ E^*m1-1M

Figure 2-20. Negative Text Size

TEXT ROTATION. The entire text label can be reoriented or rotated on the screen via two escape sequences. The first is compatible with previous Hewlett-Packard graphics terminals.

`␣*m<angle code>n`

where <angle code>	meaning
1	0 degrees
2	90 degrees
3	180 degrees
4	270 degrees

The second command allows you to rotate the text through any angle theta. Theta may be expressed in either of two ways:

`␣*n<run><rise>r`

where <run>,<rise> are real numbers in the range (-127.996, . . . , +127.996)(.004)

The text is rotated through the angle described by the direction vector relative to the X axis.

The angle of rotation may also be expressed in radians:

`␣*n<theta>r`

where <theta> is a real number in the range (-127.996, . . . , +127.996)(.004)

Figure 2-21 illustrates the differences between the two commands.

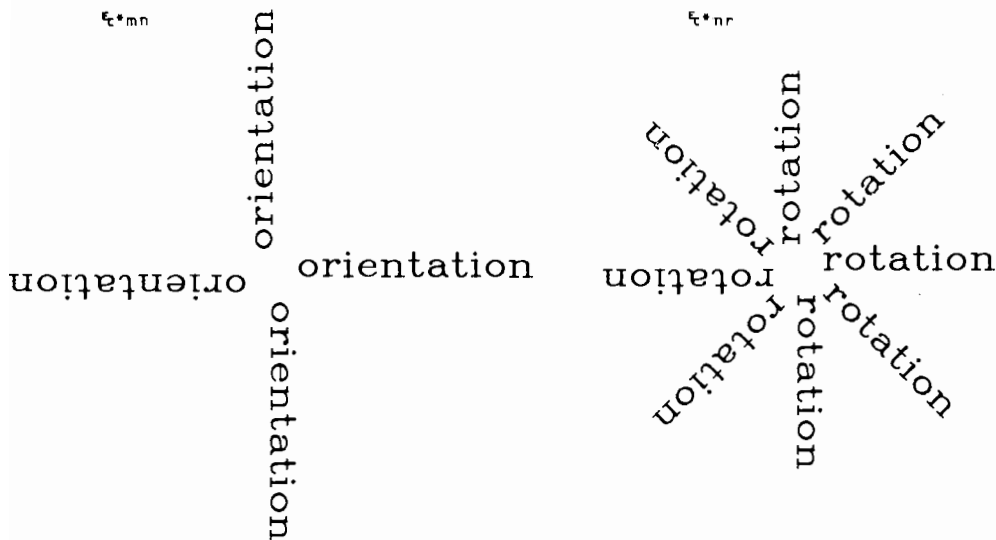


Figure 2-21. Text Rotation

Drawing Fundamentals

TEXT SLANT. Individual characters can be slanted through any angle between ± 45 degrees using the command:

`⋈*m<tan(slant angle)>o`

where `<tan(slant angle)>` is a real number in the range $(-1, \dots, +1)$ (.006). If `<slant angle>` is not specified, the previous value is used. The power on default is `tan(26.57) = 0.5`.

To turn off slant, use the command:

`⋈*mp`

TEXT JUSTIFICATION/ORIGIN. Text blocks can be automatically left, center, or right justified about a specific point.

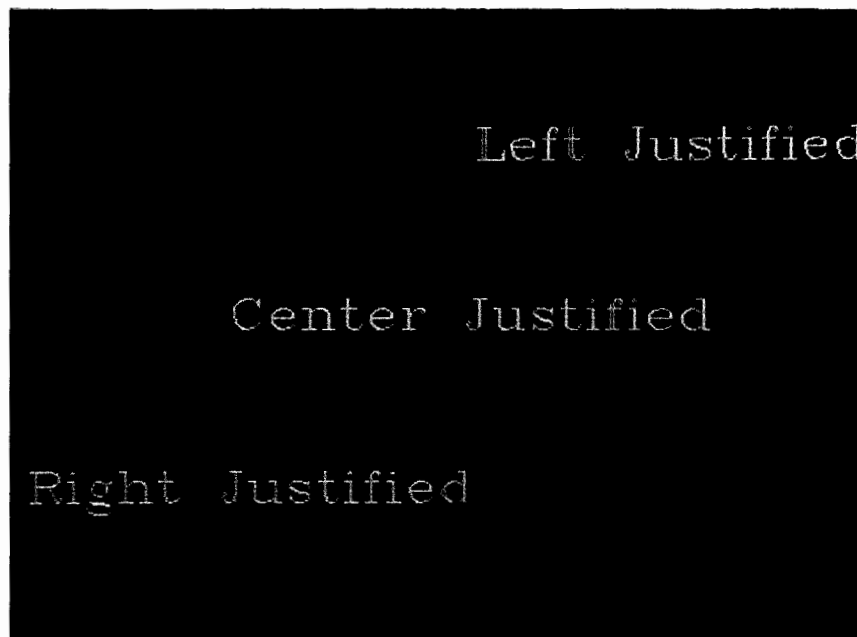


Figure 2-22. Text Justification

You can also set a baseline for the characters.

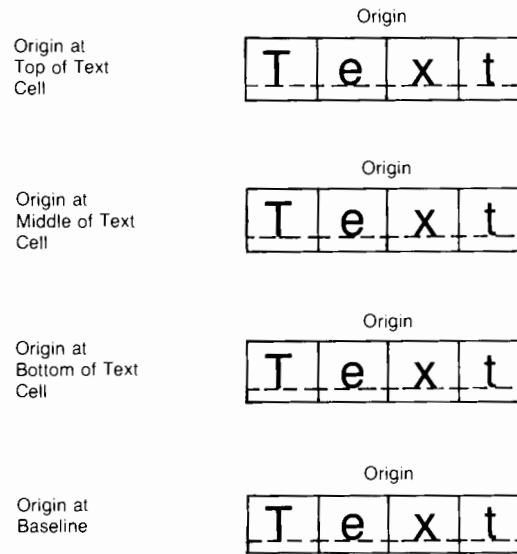


Figure 2-23. Text Origin

Text justification and origin are selected with one command:

```
⌘*m<justification/origin>q
```

where <justification/origin> is an integer in the range (0, . . . , 11)

The <justification/origin> is interpreted as follows:

left justified . . .

- 0 origin at text baseline
- 1 origin at bottom of text cell
- 2 origin at middle of text cell
- 3 origin at top of text cell

center justified . . .

- 10 origin at text baseline
- 4 origin at bottom of text cell
- 5 origin at middle of text cell
- 6 origin at top of text cell

right justified . . .

- 11 origin at text baseline
- 7 origin at bottom of text cell
- 8 origin at middle of text cell
- 9 origin at top of text cell

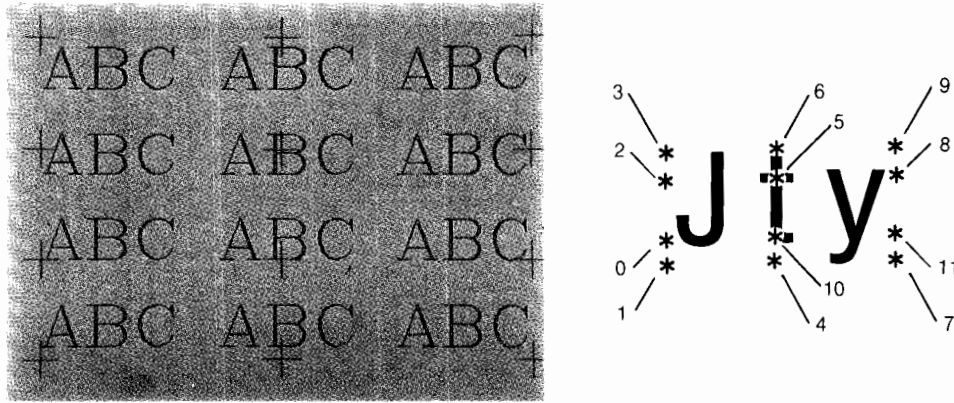


Figure 2-24. Set Text Justification/Origin

NOTES:

- If a center or left-justify operation is specified for a label which would extend beyond the bounds of virtual space, a justification/origin setting of '0' (left-justify) is selected, and the label is printed on the screen at the current pen position.
- The text baseline of a character cell is defined as the $y=0$ axis. See Section 4. CREATING A PICTURE, "User Defined Characters and Fonts."

TEXT MARGINS. The text margin defines the position of the pen after a ¶ (or ␣ after a graphics text label). After a ¶ , the pen is repositioned at the margin. After a ␣ , the pen is spaced down the height of one space character cell (parallel to the margin).

The text margin is set at the current pen position by executing the ¶*nm command. This command has two modes:

- ¶*n0m : The text margin is automatically reset to the current pen position whenever an ¶*p sequence is executed.

NOTE: In ¶*n0m mode, only ¶*p commands change the margins.

- ¶*n1m : The text margin is set permanently at the current pen position. It will remain at this position until a new ¶*nm sequence or graphics reset is performed.

Figure 2-25 shows the difference between the two modes.

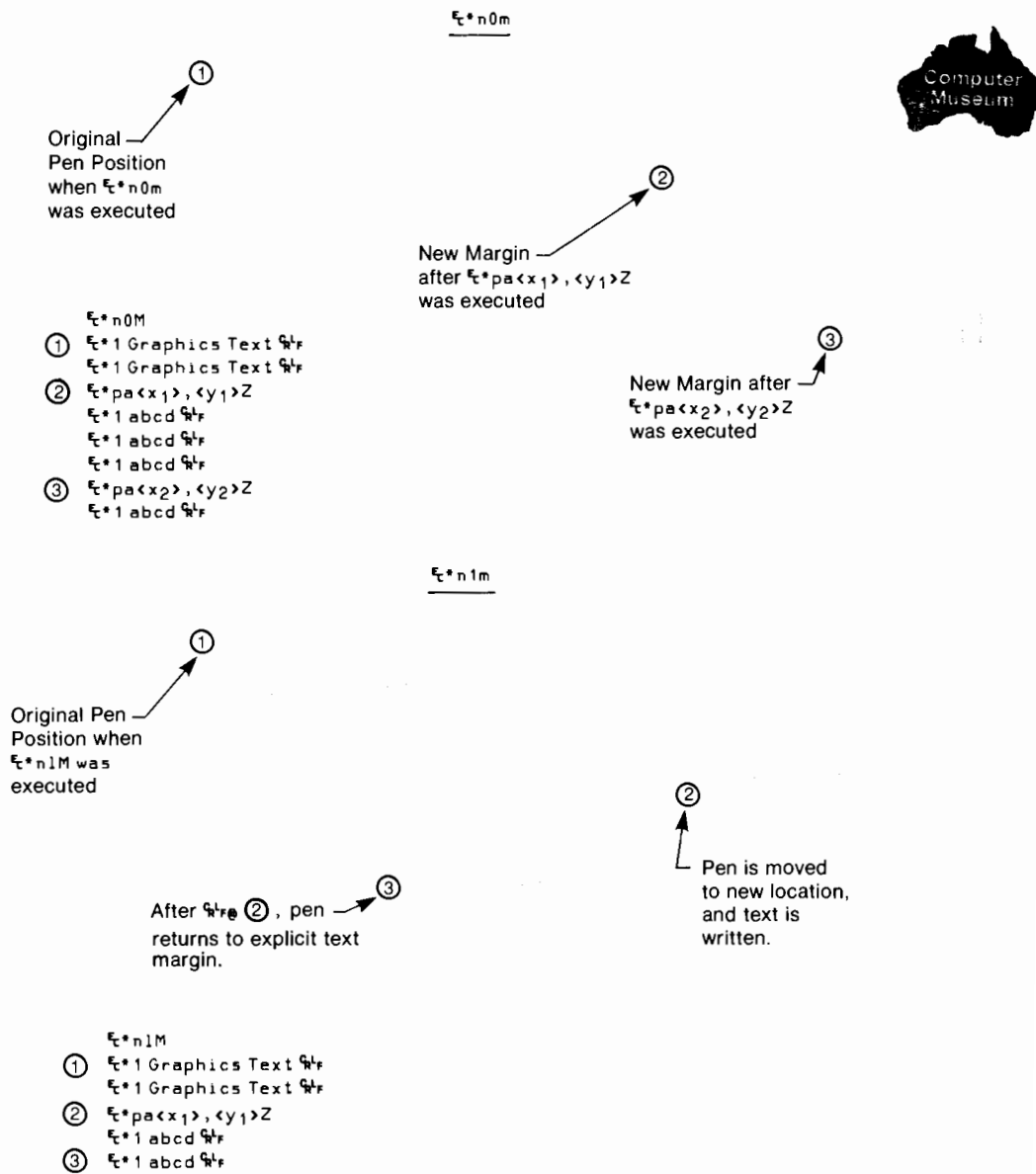


Figure 2-25. Text Margins

The text margin is set perpendicular to the text direction. If the text rotation changes, the margin is recomputed to be perpendicular to the new direction.

Drawing Fundamentals

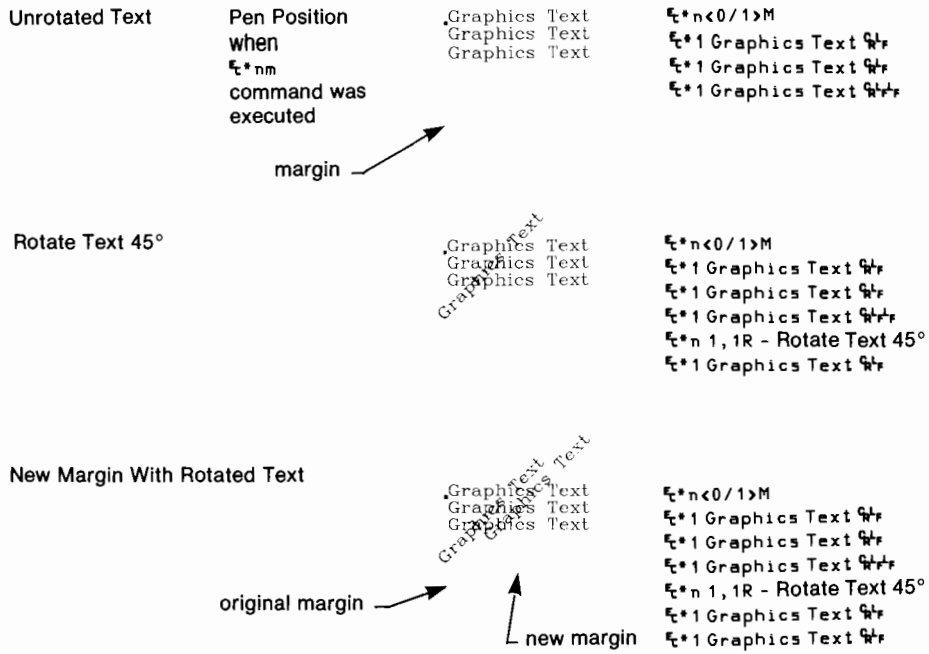


Figure 2-26. The Effect of Text Rotations on Text Margins

CHARACTER SPACING. You can compress or expand the spacing between characters with the command:

$\text{\n}^*n<spacing>w$

where $\langle spacing \rangle$ is a real number in the range $(-127.996, \dots, +127.996)$

This command sets the $\langle spacing \rangle$ as a fraction of the width of the blank character. The normal spacing is (0). Negative values compress the spacing and positive values expand the spacing.

	$\text{t}^*n\langle\text{spacing}\rangle w$
GRAPHICS	where: spacing = 0
G R A P H I C S	spacing = 2
█	spacing = -1
SCIHPARG	spacing = -2
S C I H P A R G	spacing = -3

Figure 2-27. Character Spacing

TEXT COLOR

The color of the graphics text can be set independently of the primary and secondary pens with the command:

$$\text{t}^*n\langle\text{pen number}\rangle x$$

where $\langle\text{pen number}\rangle$ is an integer in the range $(-32768, \dots, 32767)$; the low four bits are used to select a pen number in the range $(0, \dots, 15)$. Default: Pen 7

If $\langle\text{pen number}\rangle$ is supplied, it is used for all subsequent text and labels until explicitly changed or graphics defaults are set.

Use primary pen for graphics text: t^*nx

DRAWING DEFAULTS

The following command resets the drawing defaults:

`⌘*mr`

The drawing defaults are:

Move/draw flag	“Draw”
Line type	1(solid)
Drawing mode	2(Jam1)
User defined line pattern	255,1
User defined area pattern	255,255, . . . , 255
Relocatable origin	0,0
Primary Pen	15
Secondary Pen	0
Pick ID	0
Text size	1
Text direction	1
Text origin	1(left,bottom)
Text slant	off
Text angle	0 deg
Character Spacing	0
Text Pen	7(on)
Symbol mode	1(off)
Symbol scale	1.0
Symbol units	0(display)
Graphics text	off
Graphics video	0n
Alphanumeric video	0n
Graphics cursor	off
Graphics cursor position	0,0
Alphanumeric cursor	on
Rubberband line	off
Zoom	N/A
Zoom size	1,1

See also:

Section 3. COLOR

`⌘*vr` — Reset default palette colors.

Section 4. CREATING A PICTURE

`*mr` — Reset primitive defaults.

`*gb` — Reset object attribute defaults for active object.

`*eq` — Reset active view to default values.

Section 5. PICTURE FILE COMMANDS

`*wr` — Perform graphics hard reset.

Section 6. RASTER MANIPULATIONS

`*rd` — Reset raster default values.

Color Graphics 3

INTRODUCTION

If you have never used a color terminal or color output device such as a plotter, your color graphics terminal will provide you with a new dimension in graphics. The terminal has the ability to display graphics data using up to 16 different colors. These colors can be selected from a possible 4096.

Why Color?

The proper use of color in presenting graphic data can enhance almost any display. Some of the benefits of color displays are:

- Highlights data
- Allows rapid comparisons
- Aids understanding
- Increases data density

Using color makes it easier for technicians to monitor expensive or dangerous processes, helps business managers interpret financial data, and prevents errors in engineering design.

The average person can only distinguish between 13 levels of grey shading. And even if the display is limited to these levels there is a high probability of operator error when interpreting the data. By using color you can virtually eliminate these errors while at the same time provide more information and reduce operator fatigue. Figure 3-1 gives examples of displays using color.

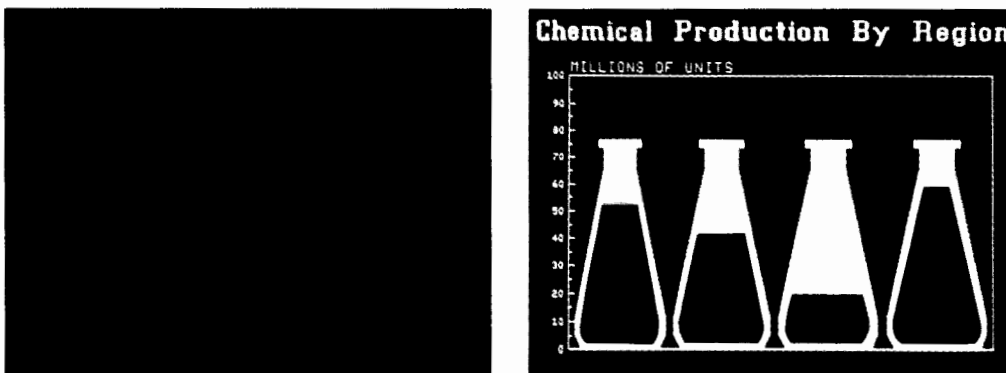


Figure 3-1. Examples of Color Graphics

You don't have to consciously read the headings on figure columns if they have been color coded. Similarly you don't have to identify a symbolic shape on a process diagram if you know that pumps are blue or that liquid sodium is red.

In chemistry for example, atoms used in molecular models are color coded to simplify their study. Oxygen can be white, carbon black, nitrogen blue, hydrogen red, etc. This eliminates the need to label each element and contributes to the scientist's ability to visualize his subject.

In computer aided circuit design of either printed or integrated circuits, color is frequently used to indicate the various layers of material. This provides a handy guide for determining the area being displayed and helps prevent costly errors.

What's in this Section

The remainder of this section will briefly discuss the concepts of color, the way the terminal generates color displays, and how you can use color programmatically. Additional reference information on using color is given in the appendices.

COLOR CONCEPTS

The following paragraphs describe the mechanics of color, the way people perceive color and the way the terminal uses color.

Perception of Color

The perception of color varies from person to person. People have individual preferences for certain colors and combinations of colors. How people perceive color can vary from time to time and under different sets of conditions. These variations are due to the way the brain receives color information.

When we "see" color we are really responding to a mixture of electromagnetic radiation at various frequencies. The normal human eye responds to radiation with wavelengths between 400 nanometers (blue) and 700 nanometers (red). Figure 3-2 shows the visible spectrum. The perceived color is a mixture of these wavelengths. The exact mixture of wavelengths determines the "chromaticity" or color content of the light without regard to its intensity. For a given color mix, the strength of the radiation is perceived as intensity.

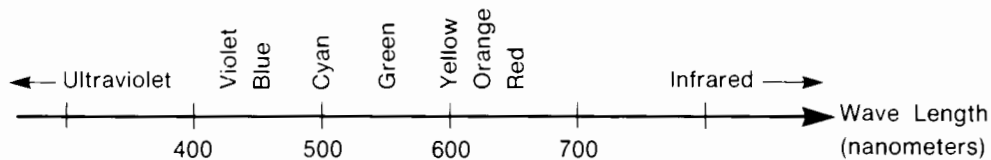


Figure 3-2. The Visual Spectrum

The eye uses two different structures to receive light, rods and cones. Rods are not sensitive to color and are used primarily for vision when there is little light such as at night. Cones are used when the light is of greater intensity. There are three types of cones, each more sensitive to a specific range or band of radiation. These frequency bands correspond roughly to red, blue, and green light. When the eye receives light, the cones provide the brain with the relative strengths of each of the three frequency bands. The brain integrates this information and we associate the mix of frequencies with a color such as yellow or pink. Figure 3-3 shows a diagram of this process.

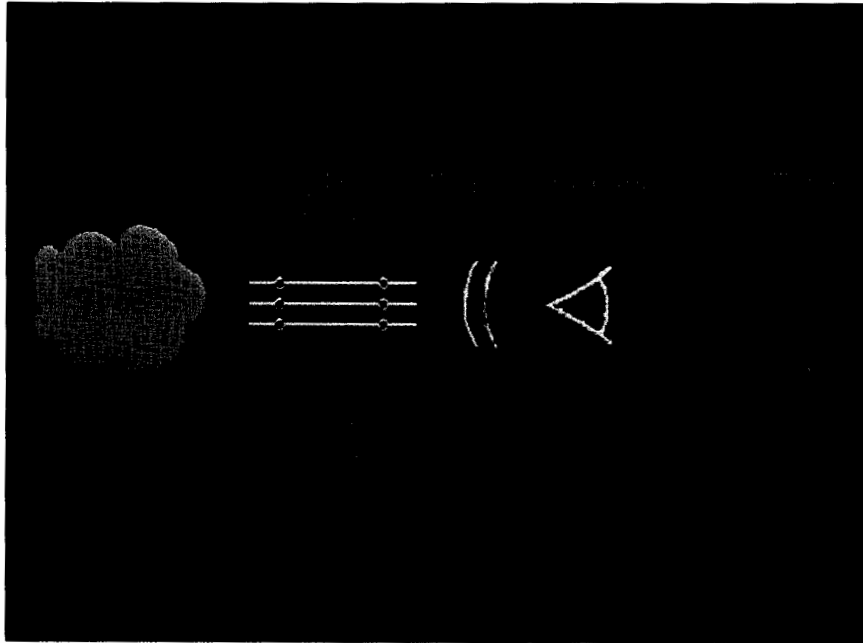


Figure 3-3. The Human Visual Process

Additive and Subtractive Models

The light that we see was broken into colors or frequency mixtures by the type of material generating or reflecting the light. Lights with colored filters allow only certain frequencies to pass, inks on paper absorb certain frequencies and reflect others (see figure 3-4). These two processes illustrate the “additive” and “subtractive” methods of color generation.

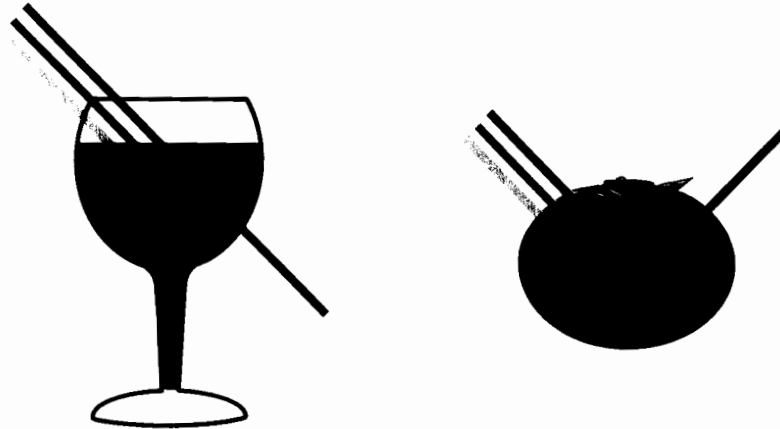
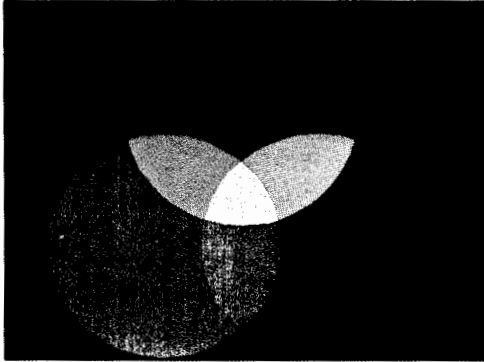


Figure 3-4. Color Absorption and Reflection

The additive and subtractive color processes allow a small number of “colors” to be used in certain proportions to create the effect of all of the possible colors. For example, most color printing is done using only three basic or primary colors: cyan, magenta, and yellow. By mixing various amounts of these primary colors together the printer can create most of the remaining colors. (Because the inks absorb light imperfectly, a few colors cannot be recreated.) Figure 3-5 gives examples of creating colors using additive and subtractive primaries.

The additive primaries are red, green, and blue. The Additive primaries are used to generate color light as in color terminals, color televisions, and theater spot lights. The subtractive primaries are cyan, magenta, and yellow. Subtractive primaries are used to generate reflected color in inks, paints, and dyes.

Additive



Subtractive

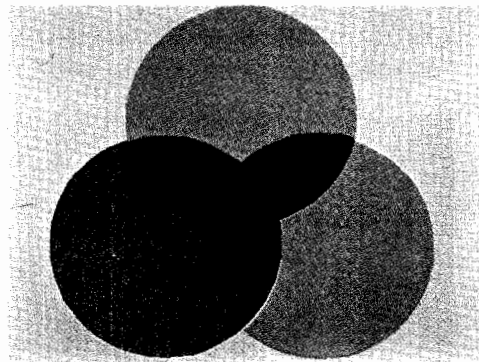


Figure 3-5. Creating Colors Using Color Primaries

Color Notation Systems

In order to manipulate color, various means of absolute color definition have been developed. These definition systems allow you to describe the mix of frequencies and the relative strength of each of the components. The color graphics terminal uses two of these color definition "methods". The first method is referred to as "HSL" or Hue, Saturation, and Lightness. The second is "RGB" or Red, Green, and Blue. Appendix B contains additional information on how the terminal converts between these two notational methods.

HSL. The Hue, Saturation, and Lightness method breaks color down into the mix of frequencies (Hue), the sharpness of the color (Saturation), and the intensity of the color (Lightness).

Hue. Hue is what we normally refer to as color. It refers only to the frequencies and not their sharpness or intensity. A rainbow is a simple display of hue. The various frequencies have been scaled and assigned numeric values between 0 and 1. Figure 3-6 shows the hue values used by the terminal. Note that the values have been laid out so that the scale can be circular. The hues range from red thru green and blue and back to red.



Figure 3-6. Color Hues

Saturation. Saturation indicates the amount of color present compared to the maximum possible. Figure 3-7 gives an example of a single hue, red, displayed at various saturation levels.

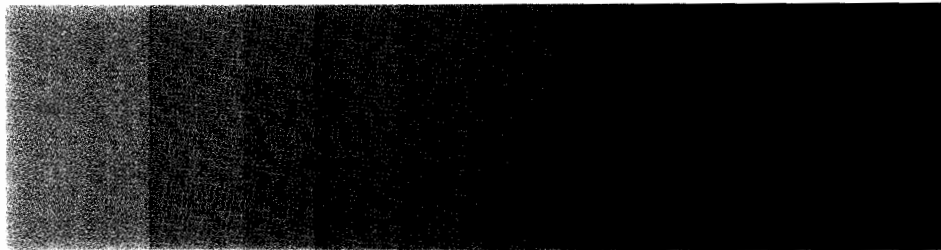


Figure 3-7. Saturation

Lightness. Lightness is scaled between 0 (black) and 1 (white). Figure 3-8 illustrates the effect of the lightness parameter on color selection. A definition of lightness as used in this terminal and equations of transformation between the RGB and HSL notation methods are given in Appendix B of this manual.



Figure 3-8. Lightness

The HSL color notation method is frequently described using a cylinder as the model. The angular position denotes hue, the radial distance out from the central axis of the cylinder indicates the saturation, and the height gives the lightness. Figure 3-9 shows the HSL model.

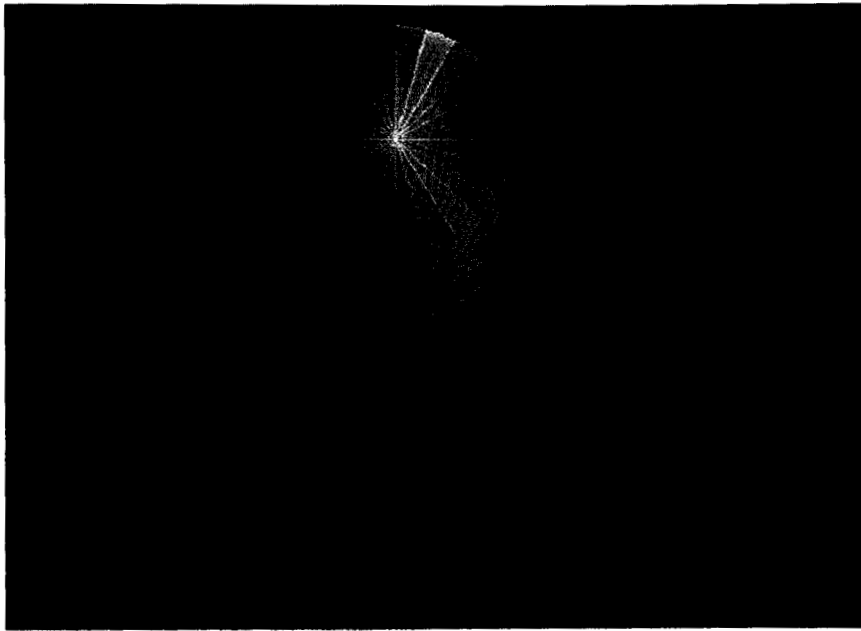


Figure 3-9. HSL Cylinder Color Model

RGB. The RGB method of notation allows you to specify directly the amounts of red, green, and blue light to be generated. The ratio of red, green, and blue correspond roughly to hue, while amounts of red, green, and blue used compared to the maximum amounts that could be used correspond roughly to saturation and lightness.

The R, G, and B values range between 0 and 1. They combine in an additive process to create a resultant color. Figure 3-10 shows sample colors created using R, G, and B values.



Red	Green	Blue	Result
100%	0%	0%	
100%	100%	0%	
100%	0%	100%	
93%	33%	0%	

Figure 3-10. RGB Color Examples

The RGB notation uses a cube as its color model. The R, G, and B values make up the three axes of the cube (see figure 3-11).

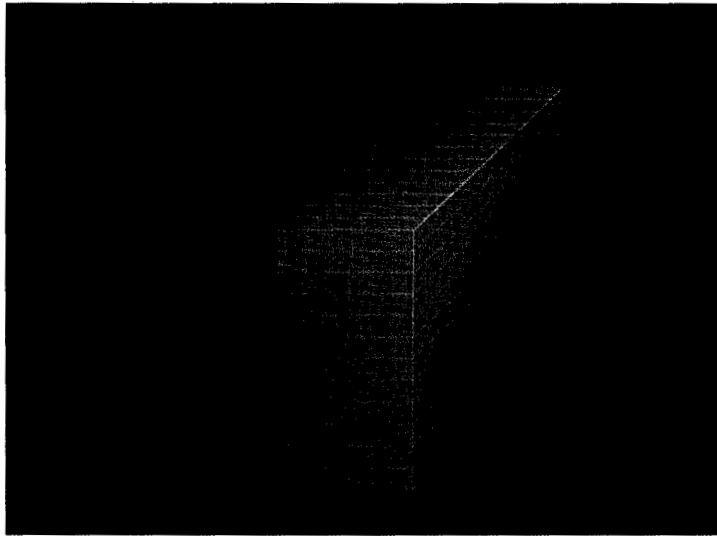


Figure 3-11. RGB Cube Color Model

HOW COLOR IS GENERATED IN THE TERMINAL

The terminal uses the additive primary colors red, green, and blue to generate colors. Figure 3-12 illustrates the terminal's color generation system.



Figure 3-12. Terminal Color System

Color CRT

The terminal uses three separate electron beams to energize three different phosphor coatings on the face of the CRT. One phosphor glows red when hit by the beam, the others green and blue. The phosphors are so placed and shielded with a mask that only the beam of electrons intended for the red phosphor can reach it. The same is true of the electrons from the green and blue guns. By causing the R, G, and B phosphor coatings in a particular area to glow at various intensities, almost any resultant color can be generated. Figure 3-13 illustrates this process.

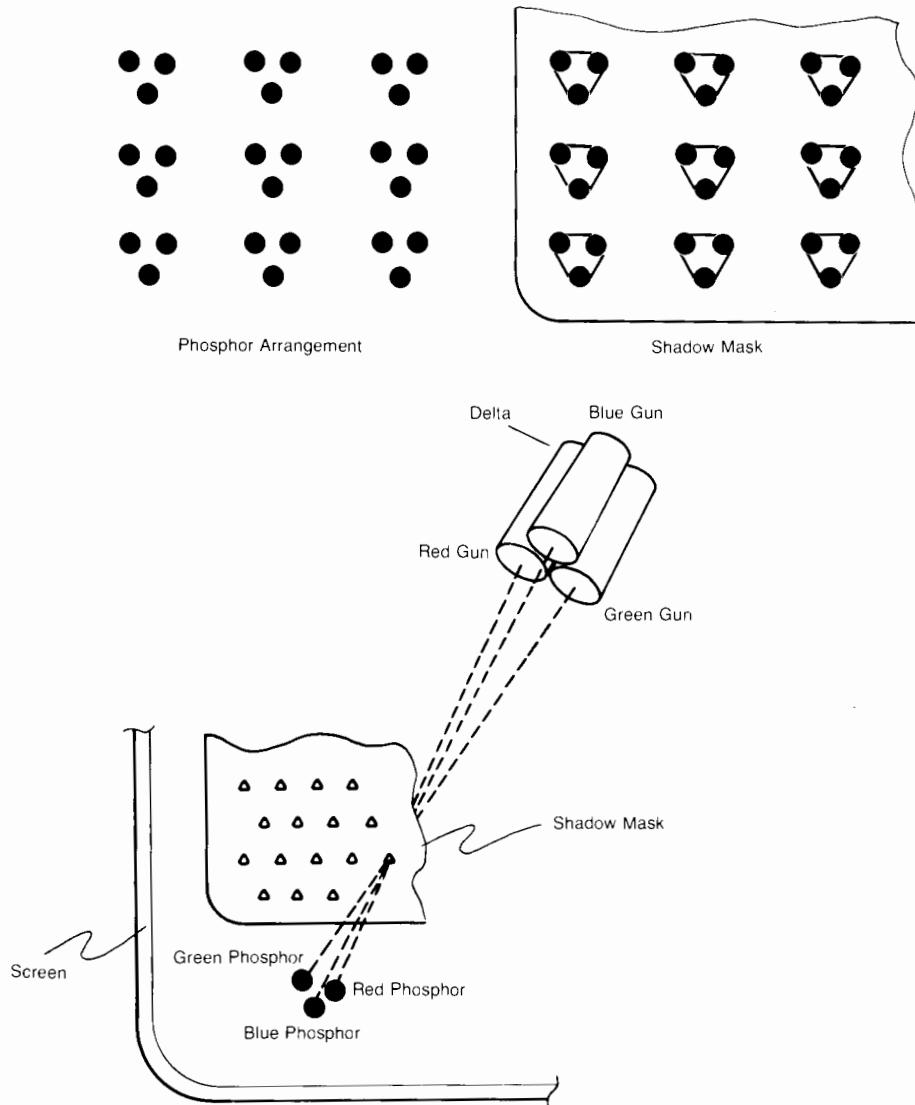


Figure 3-13. Color CRT

Raster Memory

The terminal stores the image to be displayed on the screen in a special area called “raster memory”. The raster memory is used to refresh the terminal screen and is made up of four planes of bits. There are 512 rows of 512 bits in each plane. The planes are assigned binary values so that a bit on the first plane is a “1”, a bit on the second plane is a “2”, a bit on the third plane is a “4”, and if a bit is set on the fourth plane it has a value of “8”.

For any given point on the terminal screen, the corresponding bits in each memory plane are added to produce a resultant value. The resultant value is in the range 0–15 and corresponds to the pen number used to draw the figure. The result of combining the data in the four memory planes is to create a “paint by the numbers” picture. Figure 3-14 illustrates this concept. Figures 3-15 and 3-16 illustrate how this information is used to produce the final picture.

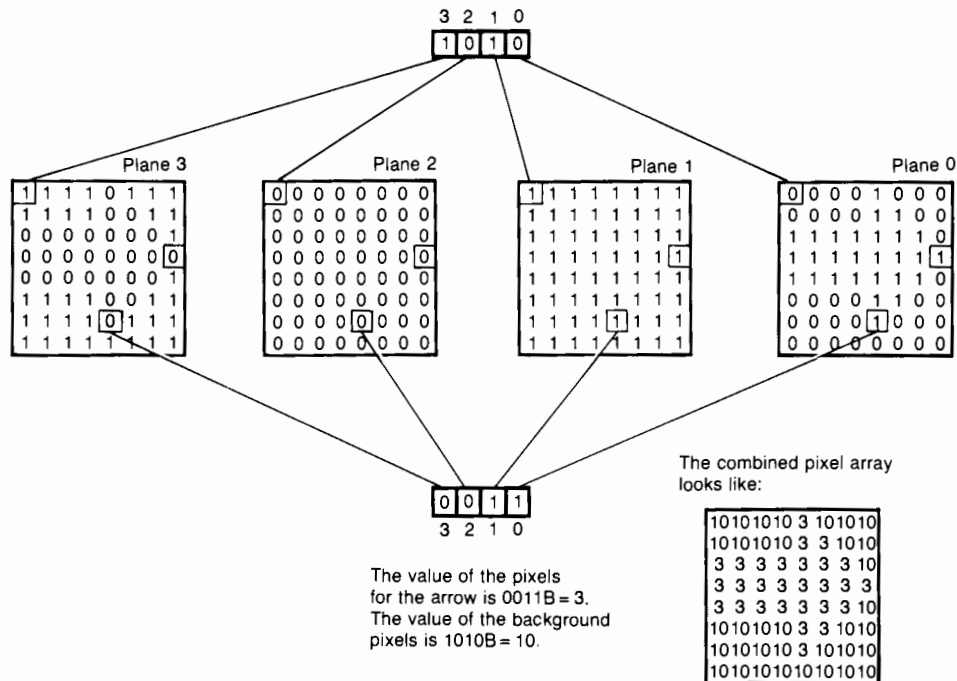


Figure 3-14. Raster Memory Planes

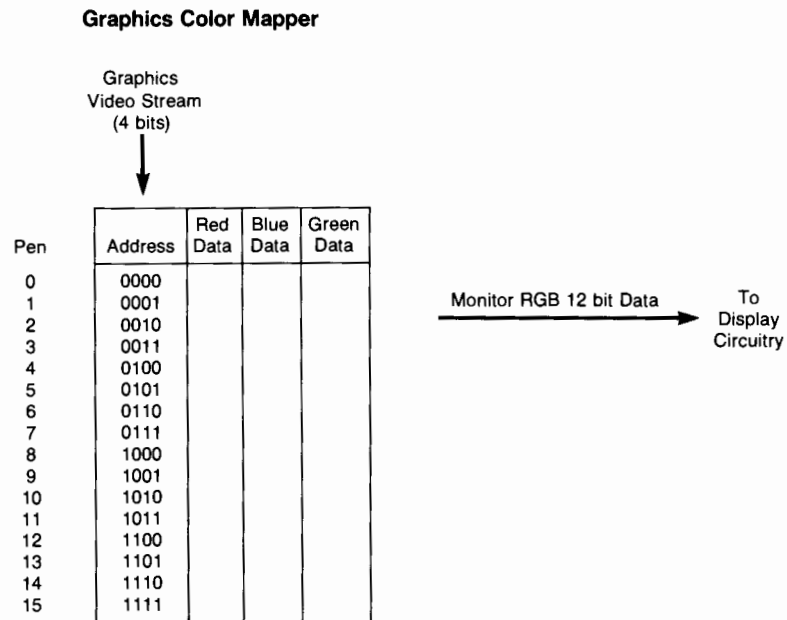
The output of the raster memory can be controlled with a mask to allow only some of the memory planes to send their data to the color mapper. Masking is discussed in Section 6.

The terminal can be equipped with a secondary or auxiliary raster memory. This auxiliary raster memory is merely a second set of four memory planes. The terminal screen can be switched to display data from either the primary or the secondary raster memory. Additional information on the secondary or auxiliary raster memory is given in Section 6.

Color Mapper

The color mapper holds the R, G, and B color components used to generate the colors assigned to each of the 16 pens. When a color palette is selected, the R, G, and B values for each of the pens is stored in the color mapper. These pen colors are used to fill in the “paint by numbers” picture provided by the raster memory. This separation of color information from pen number allows you to easily change the color assignments without having to edit the drawing or change the pen number used to draw a line.

Figure 3-15 illustrates the function of the color mapper. The data from raster memory is used to select a pen for each point on the screen. The pen number selects the red, green, and blue intensity levels to be sent to the CRT. The red, blue, and green intensity levels are used to control the number and strength of electrons striking the three screen phosphors. These phosphors glow at various intensities to generate the desired color.



Pens and Palettes

The terminal allows you to draw figures using 16 different pens. As each line is drawn, the terminal stores coordinate information for the line along with the current pen. (You can change pens at any time using a "Select Pen" command.) The data defining coordinates and pen is used to write into the raster memory. The raster memory planes actually store the binary equivalent of the pen number (see figure 3-16).

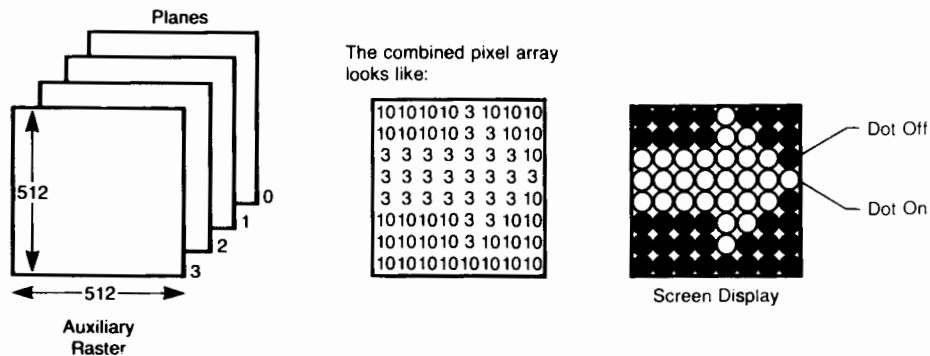


Figure 3-16. Drawing with Pens

NOTE: The values written into the raster memory can be controlled by setting a binary mask. The mask has the effect of blocking the data from being stored in selected raster planes. The masking process is discussed in Section 6.

These pens correspond to the 16 colors that can be displayed at any given time. When the drawing is displayed, the pen numbers are used to select one of the 16 colors in the current palette. In this sense, selecting a pen is the same as selecting a color. But since only the symbolic pen number is actually stored, the 16 colors can be changed without affecting the drawing data.

Selecting the 16 colors is a separate process from selecting the pens. A selection of 16 colors is called a "palette". You can define up to 256 different palettes. The palettes are numbered 0-255. Palette 0 is a scratch or temporary palette used to hold the current colors. When you select a palette for use, a copy of the color definitions for that palette is copied into palette 0. The terminal uses palette 0 to control the color mapper.

USING COLOR

Using color consists of two steps:

- defining palettes
- selecting palettes and pens

The first step, defining palettes, allows you to assign RGB or HSL color components to individual pens on a particular palette. This is normally done prior to or at the beginning of the drawing process. The second step, selecting a palette and pen, occurs continuously throughout the drawing process.

In addition to these methods, you can use masking and direct raster manipulation to achieve special color effects.

Color Assignment

















Color assignment is the process of setting the colors of pens on a particular palette. The escape sequences used to control color assignment are given in table 3-1.

Table 3-1. Graphics Color Control Sequences

Command	Description	Default
<code>Ec*v</code>		
<code><value>a</code>	Set the 1st color parameter (Hue or Red).	0
<code><value>b</code>	Set the 2nd color parameter (Saturation or Green).	0
<code><value>c</code>	Set the 3rd color parameter (Lightness or Blue).	0
<code><palette#>d</code>	Delete the indicated palette.	0
<code>e</code>	Delete all palettes.	0
<code><pen#>i</code>	Select pen.	0
<code>l</code>	Load the selected palette into palette 0.	0
<code><method>m</code>	Select HSL or RGB color method.	0
<code><palette#>p</code>	Select palette.	0
<code>r</code>	Set the selected palette to the power-on values.	0
<code><mask><pen#><palette#>^</code>	Report the status of a palette.	0

There are 4096 possible colors that can be assigned to 16 pens on 256 palettes. One of the palettes, palette 0 is referred to as the system palette. The system palette is the one that is used to display data. Palette switching is accomplished by loading the colors of a selected palette into the system palette. You can display the current palette by pressing the `COLOR` key.

When the terminal is turned on or after a hard reset, the system palette is set to a set of default colors. The default palette colors are shown in figure 3-17.

Pen #	Color	R	G	B	H	S	L
0		0	0	0	0	0	0
1		1	0	0	1	0	0
2		0	1	0	0.33	1	1
3		1	1	0	0.16	1	1
4		0	0	1	0.66	1	1
5		1	0	1	0.83	1	1
6		0	1	1	0.5	1	1
7		1	1	1	0	0	1
8		0.39	0.13	0	0.05	1	0.39
9		0.93	0.33	0	0.05	1	0.93
10		0.59	1	0.06	0.23	0.93	1
11		0.93	0.73	0	0.13	1	0.93
12		0.33	0.33	1	0.66	0.66	1
13		0.59	0.13	1	0.75	0.86	1
14		0	1	0.53	0.42	1	1
15		0.79	0.79	0.79	0	0	0.79

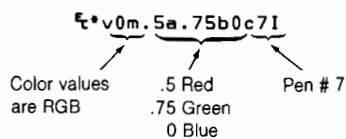
At this time the remaining palettes (1-255) have no color assignments. If you read the contents of the picture file you will find only palette 0 defined. If you reference a palette that has not been assigned colors, the terminal will initialize it to the colors contained in the system palette (palette 0). Figure 3-17 illustrates how this works.

If you select a palette that has been defined, the colors in the selected palette are unaffected. If you then load the palette into the system palette, any changes made to the system palette using the `COLOR` key functions will also affect the stored palette.

The steps in color assignment are:

- Select a palette (default is 0 or last palette)
- Select the color notation method (default is 0, RGB)
- Specify RGB or HSL values (defaults are 0)
- Select the pen # to be changed

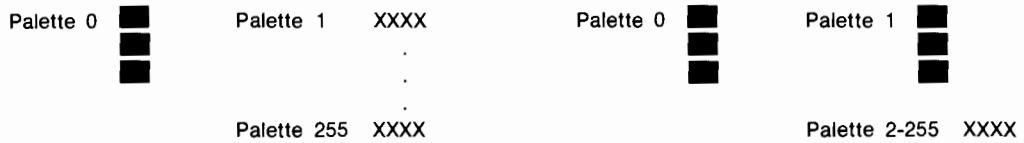
A typical control sequence would be:



COLOR PARAMETERS. The color parameters are entered as values between 0 and 1 followed by the letters “a”, “b”, or “c”. The letters indicate the RGB (red, green, blue) or HSL (hue, saturation, lightness) depending on the color method in effect.

Code	Function
<value> a	Red or Hue
<value> b	Green or Saturation
<value> c	Blue or Lightness

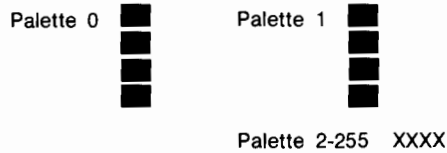
The values for color parameters are initialized to 0 at the start of each τ^*v sequence. This means that if you do not specify one or more of the color parameters, and do specify a pen #, the missing parameters will be assigned 0 values.



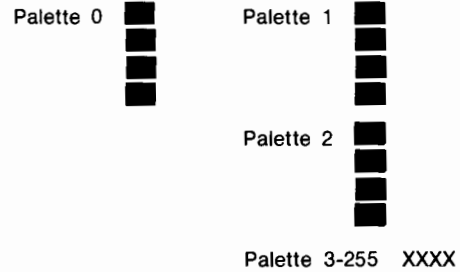
A. Power On



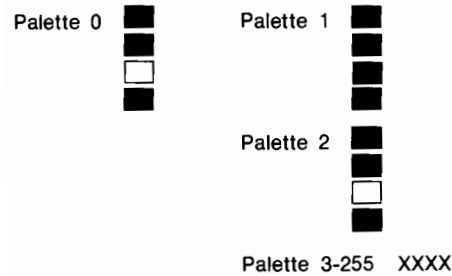
B. Select Palette 1



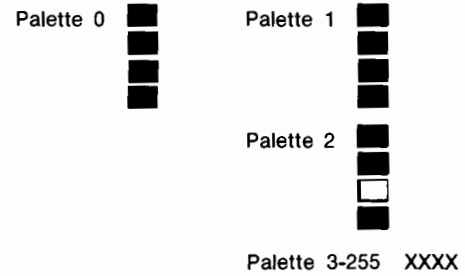
C. Change Pen 3 to Black



D. Select Palette 2



E. Change Pen 2 to White



F. Select Palette 1

□ = Defined
X = Undefined

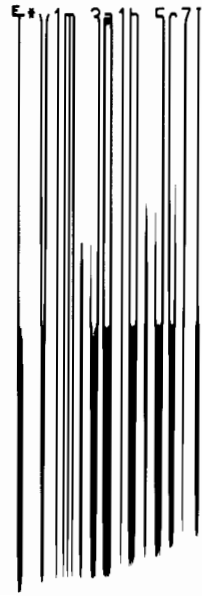
Figure 3-17. Color Palettes

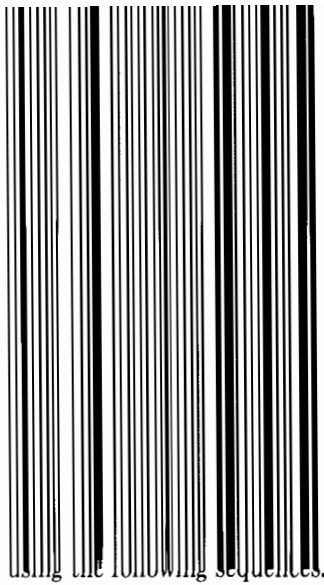
The color values do not take affect until a pen # is specified. At least one pen # must be included following the color parameters. If no pen # appears, the entire escape sequence is ignored.

Example: Set pen #4 to RGB values of .5, .8, and .6.

```
␣*v0m.5a.8b.6c4l
```

Example: Set pen #7 to HSL values of .3, 1, and .5.





`␣*m<primary pen>x<secondary pen>y`

When you assign the primary and secondary pens you should remember that the color assignments may affect your drawing. Depending on the drawing mode used (Section 2 and Section 6), the pens will affect the raster memory in different ways. For example, if you draw using a dot-dash line pattern and the terminal is set to draw using JAM2 mode, the secondary pen is used to draw the spaces between the dots and dashes. If the secondary pen is black (RGB=0,0,0), then the secondary pen will normally have no effect on the underlying drawing. If the secondary pen is assigned color values, the results can vary greatly depending on the drawing modes used.

Background Color

The same considerations used in selecting primary and secondary pens should be used in selecting a background color. You should select a background color (`␣*e<pen#>B`) that goes well with the colors that you intend to use in your drawing.

Color Status

You can request the terminal to list the color assignments for any of the palettes. You can select the type and format of data returned by setting mask bits in the request.

Palette Status: `␣*v<mask>[<pen#>[<palette#>]]^`

where: `<mask>` = -32768 to 32767 (note: 0 returns no information)
`<pen#>` = 0 to 15
`<palette#>` = 0 to 255

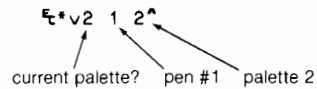
The six least significant mask bits are interpreted as follows:

Bit	Meaning	Data Format
0	Palette exists (0/1)	n
1	Palette is current palette (0/1)	n
2	Color method - RGB or HSL (0/1)	n
3	Current palette	+nnnnn
4	RGB values for current pen	n.nnnn,n.nnnn,n.nnnn
5	HSL values for current pen	n.nnnn,n.nnnn,n.nnnn

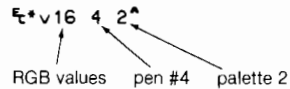
If a palette is not specified, the status of palette 0 is returned. If a pen is not specified, the status of pen 15 is returned.

NOTE: If a palette is specified without specifying a pen, the palette # will be interpreted as a pen #. This will result in erroneous status information.

Example: Check to see if palette 2 is defined.



Example: Request the status of pen 4 on palette 2. Use the RGB method. (Note that the mask setting RGB values requires a decimal 16 to set the 4th bit in the mask.)



Sample Program: The following sample program reads the status of palettes 0 through 2 and lists pens 0 through 3 for each palette.

```

COLORTST
 10 DIM A$(255)
 20 PRINT
 30 FOR P=0 TO 2
 40   PRINT '27"*v15 0";P;"^";
 50   INPUT A$
 60   IF A$[1,1]="0" THEN PRINT "Palette #";P;"(NOT defined)"
 70   ELSE DO
 80     IF A$[3,3]="0" THEN PRINT "Palette #";P;"(NOT loaded)"
 90     ELSE DO
100       PRINT "Palette #";P;"(loaded)"
110       PRINT "          RED          GREEN          BLUE"
120       FOR I=0 TO 3
130         PRINT '27"*v16";I;P;"^";
140         INPUT A$
150         PRINT "Pen #";I;A$[1;6],A$[8;6],A$[15;6]
160       NEXT I
170       PRINT
180     DOEND
190   DOEND
200 NEXT P

```



```

>run
COLORTST

Palette # 0      (loaded)
                RED          GREEN          BLUE
Pen # 0         0.0000      0.0000      0.0000
Pen # 1         1.0000      0.0000      0.0000
Pen # 2         0.0000      1.0000      0.0000
Pen # 3         1.0000      1.0000      0.0000

```

```

Palette # 1      (NOT Defined)
Palette # 2      (NOT Defined)

```

```
>
```

PRINTERS AND PLOTTERS

If you output graphics data to a printer that cannot produce the colors used in the drawing, the terminal can convert the color data to various black and white patterns. Instructions for configuring the terminal for output to graphics devices are given in Section 7.

There are two modes of non-color output. The first is called "Black and White", the second is called "Halftone".

Black and White

If Black and White output has been selected, the terminal will output a black dot if a bit is on in any of the four raster planes (pixel>0). This is true for all pens (pixel values) except for the background pen. When a picture is sent to a black and white output device, the background pen is treated the same as a pixel with a value of 0. This allows the drawing background to take on the color of the paper used in the printer or plotter (normally white). If for any screen dot location all of the bits in raster memory are zero (pixel=0), then the output dot will be white. This mode of operation is intended for simple output devices with a low number of dots per inch.

Halftone

If Halftone output has been selected, the terminal will convert the color pens into grey scale equivalents. The RGB component values are used to select a grey scale pattern according to the following equation:

$$\text{Grey Scale} = .3R + .59G + .11B$$

The pattern used for each pen is selected from the 64 patterns shown in figure 3-18.

There are two modes of halftone operation, high contrast and normal. In high contrast mode the grey scale values for each of the pens is calculated and the pens are then ranked in order of their grey scale. The figure is then drawn using pattern 1 for the lightest (lowest grey scale value) pen, pattern 8 for the next lightest, pattern 12 for the next and so on up to pattern 64 for the darkest pen. (Note: pattern 4 is not used.) If two pens evaluate to the exact same grey scale value, the same pattern will be used for both.

In normal mode, the grey scale is used directly to select one of the 65 patterns. If the grey scale value is 0, then pattern 0 is used (no dots on). If the grey scale is 1, then pattern 64 is used (all dots on).

1		9		4		12	
13		5		16		8	
3		11		2		10	
15		7		14		6	

Patterns 1-16

17		25		20		28	
29		21		32		24	
19		27		18		26	
31		23		30		22	

Patterns 17-32

33		41		36		44	
45		37		48		40	
35		43		34		42	
47		39		46		38	

Patterns 33-48

49		57		52		60	
61		53		64		56	
51		59		50		58	
63		55		62		54	

Patterns 49-64

Figure 3-18. Halftone Patterns

Selecting Color Output Modes (f_t*u_a)

You can select the mode of color output using the following escape sequence: f_t*u<mode>a

where <mode> = 0 Autotracking off
 1 High Contrast
 2 Normal (default)

Setting autotracking off allows you to define your own halftone patterns for pens. If colors are changed in the palette, the patterns will not change.

Color Graphics

Selecting high contrast will use 16 evenly distributed patterns for the pens. Patterns assigned to pens will automatically change if the palette colors are changed.

Normal mode will select the halftone pattern that most nearly matches the lightness (apparent grey scale) of the palette colors. The patterns will automatically be changed if the palette colors are changed.

Setting the Halftone Defaults (ξ^*_{ub})

You can select the halftone default patterns (patterns for the default palette) using the escape sequence ξ^*_{ub} . Table 3-2 lists the grey scale values and pattern numbers used for the default patterns.

Table 3-2. Default Halftone Patterns

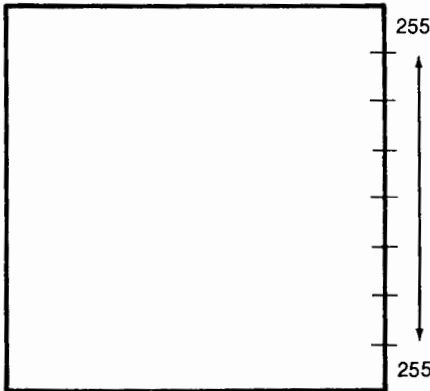
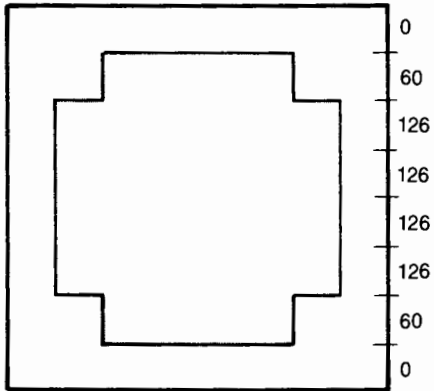
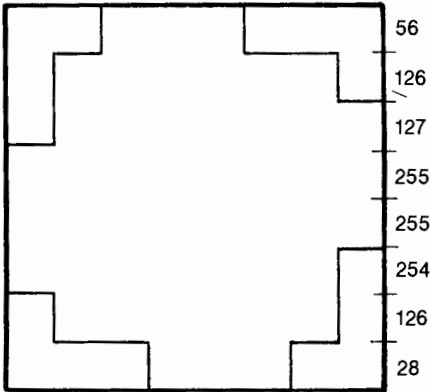
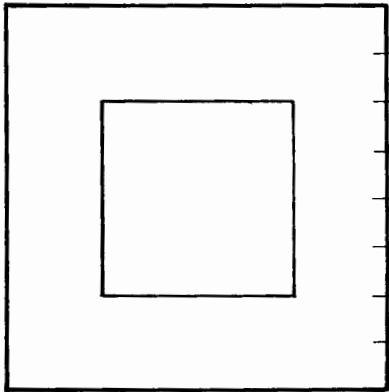
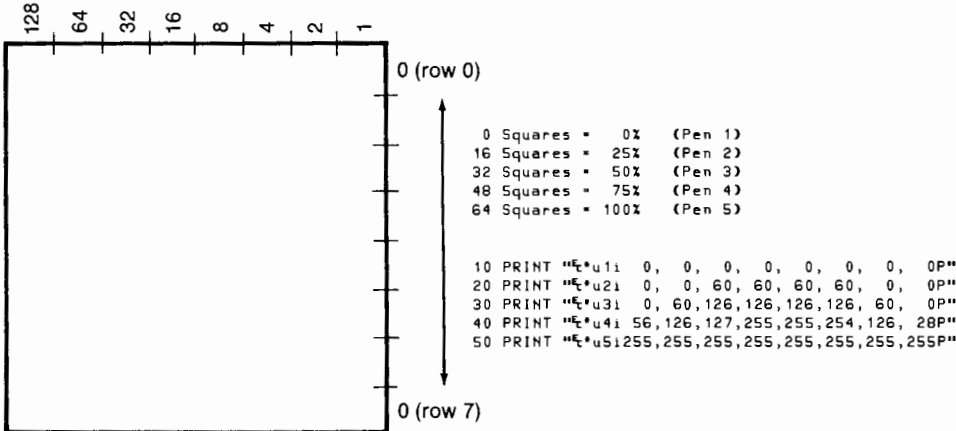
PEN #	GREY SCALE	PATTERN #
0	1/16	4
1	5/16	20
2	9/16	36
3	13/16	52
4	3/16	12
5	7/16	28
6	11/16	44
7	15/16	60
8	2/16	8
9	6/16	24
10	10/16	40
11	14/16	56
12	4/16	16
13	8/16	32
14	12/16	48
15	16/16	64

User Defined Patterns ($\xi^*_{ui,p}$)

You can supply your own patterns for a given pen. This is done in a manner similar to the way area fill patterns are defined. First select the pen for which a pattern is to be defined, and then output an 8 by 8 bit pattern. This function is required to produce grey scales that correspond to a dot of increasing diameter in the center of a field as opposed to dots that are distributed evenly in the pattern. (This is useful when creating output that is to be used in professional printing applications.)

The user defined patterns are not saved as a part of the picture file. They are lost if the terminal is reset or powered off. If color autotracking is enabled and a palette color is changed, the pattern will be replaced with one of the predefined normal or high contrast patterns.

Example: Define five grey scale patterns for 0, 25%, 50%, 75%, and 100% values. Make the assignments to pens 1 through 5.



Halftone Status

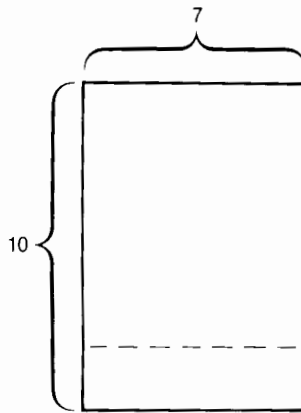
You can request information of the current halftone mode and patterns. Refer to Section 8, Graphics Inquiry and Status for additional information.

ALPHANUMERIC COLOR CONTROL

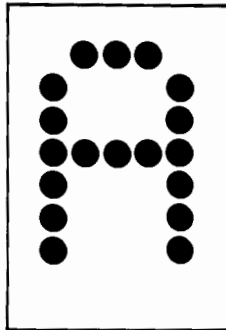
You can control the color of alphanumeric characters and their backgrounds. This process is completely separate from graphics color control. The technique of alpha color control is similar to that used in graphics color. You use R, G, and B (red, green, and blue) or H, S, and L (hue, saturation, and lightness) color selection methods and color combinations are assigned to pens.

Pens

There are 16 alphanumeric color pens (0-15). Each pen can be assigned a color for characters and a separate color for the character's background.



The background color is used for all dots in the character cell except those used to draw the character. For example, if the letter "A" is drawn using pen "0" and pen "0" has been assigned white for the letter and black for the background, the character cell will appear as follows:



NOTE: Alphanumeric characters overlay or mask out the graphics display. To allow graphics data to be seen, select black (clear) background colors for the alphanumeric pens.

STANDARD PEN ASSIGNMENTS. Some parts of the alphanumeric display, the command, message, and label lines have pens permanently assigned to them. The remaining display has pen "0" assigned as the default pen and you are free to reassign pens to this display area on a character by character basis. You can change colors assigned to any pen (0-15) or assign any pen to be used for alpha characters.

Pen	Display Assignment
0	Default alpha pen
7	Labels F1-F8
12	Message line
13	Command line
14	Labels F9-F16
15	Configuration menus

The current color assignments can be displayed by executing a terminal self-test. The character set display will be shown using the 16 alpha pens. Figure 3-19 shows location of the alpha colors in the self-test pattern.

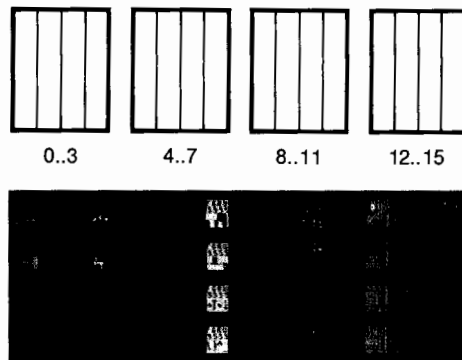


Figure 3-19. Alpha Colors in Self-Test Pattern

Color Graphics

Alpha color control differs from graphics color in that you are limited to a selection from only 64 different colors. Also, alpha color control can only be done through escape sequences. The escape sequence for color control is:

Alpha Color Control: `ESC & v <parameters >`

where the parameters consist of one or more of the following color commands:

Command	Description	Default
<code><0/1>m</code>	Color specification method (RGB/HSL)	0
<code><decimal>a</code>	Red (or hue) color value	0
<code><decimal>b</code>	Green (or saturation) color value	0
<code><decimal>c</code>	Blue (or lightness) color value	0
<code><decimal>x</code>	Red (or hue) color of background	0
<code><decimal>y</code>	Green (or saturation) color of background	0
<code><decimal>z</code>	Blue (or lightness) color of background	0
<code><0-15>i</code>	Pen to which color is to be assigned	0
<code><0-15>s</code>	Pen to be used for characters	0

Specifying Colors

You specify colors using the following steps:

- Select color method
- Assign colors to pens
- Select a pen

COLOR METHOD. Alpha colors are specified using either RGB or HSL color notation. The color method is set with the following color command:

`<0/1>m`

where: 0=RGB, 1=HSL

The color method command tells the terminal how it should interpret the parameters used with the color selection commands (a,b,c,x,y,z). If no method command is present in the escape sequence containing a color specification, the terminal will use the last method entered. The method selected will remain in effect until programmatically changed, a full reset, or the terminal is powered on. At power on or when a full reset is performed, the color method is set to RGB. The terminal sets the RGB method as default at power on.

Example: Set the color method to HSL.

`ESC & v 1 m ...`

COLOR ASSIGNMENT. Colors are assigned to specific pens by sending color commands followed by a pen assignment.

```
Ⓔ & v < > a < > b < > c < > x < > y < > z < pen # > i
```

Colors are specified using R, G, B, or H, S, L methods. If the RGB method is used you can enter one of four possible values (0, .33, .66, or 1) for each of the color components.

SAMPLE ALPHA COLORS

R	G	B	Color	H	S	L
0	0	0	black	0	0	0
0	0	1	blue	.66	1	1
0	1	0	green	.33	1	1
0	1	1	green/blue	.5	1	1
1	0	0	red	1	1	1
1	0	1	purple	.83	1	1
1	1	0	yellow	.16	1	1
1	1	1	white	0	0	1



At the beginning of a color sequence, each of the RGB or HSL values are assumed to be 0. If the color specification for the next pen does not specify one or more of the components, of a specification, it is assumed to be 0. These values control the red, green, and blue elements of the terminals display (0 = off, 1 = full on). When the HSL method is used, the H value selects a particular hue, the S value controls the saturation of the color, and the L value controls the lightness of the color. The HSL values can range between 0 and 1.

Example: Specify a foreground color of yellow and a background color of purple. Assign the colors to pen 5.

```
Yellow = red + green
Purple = red + blue
```

```
foreground = yellow
Ⓔ & v 0 m 1 a 1 b 1 x 1 z 5 I
background = purple
```

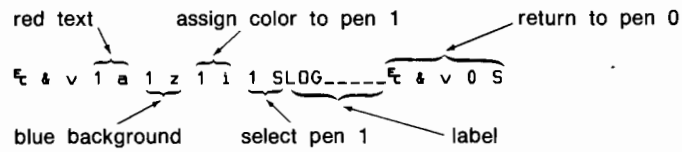
Pen Selection

Once colors have been assigned to the pens, you can select the pens to be used for alpha characters with the pen selection command.

Select an Alpha Pen: `␣&v<pen#>S`

The display is normally set to pen 0. To create a small form using the red on blue colors you should use a different pen. If pen 0 is changed to red on blue, the entire screen will be set to a blue background and any characters on the screen will become red. This defeats the purpose of highlighting a specific field. Instead, assign the red on blue to another pen such as pen 1. Then specifically select pen 1 for the label only. When the label is complete, return to pen 0.

Example: Print the word "LOG" in red on blue followed by a field of 5 blank characters.



Creating a Picture _____ 4

INTRODUCTION

This section explains the essentials of building a picture file (vector lists, object attributes, views, and user defined characters/fonts) using escape sequence programming. The following discussion assumes that you have read Section 1. *GRAPHICS CONCEPTS*, "Picture Organization".

Picture files are normally created by:

1. defining one or more vector lists,
2. creating objects by combining object attributes and vector lists
3. defining virtual window/screen viewport pairs (views), and optionally,
4. creating user defined fonts for graphics labels.

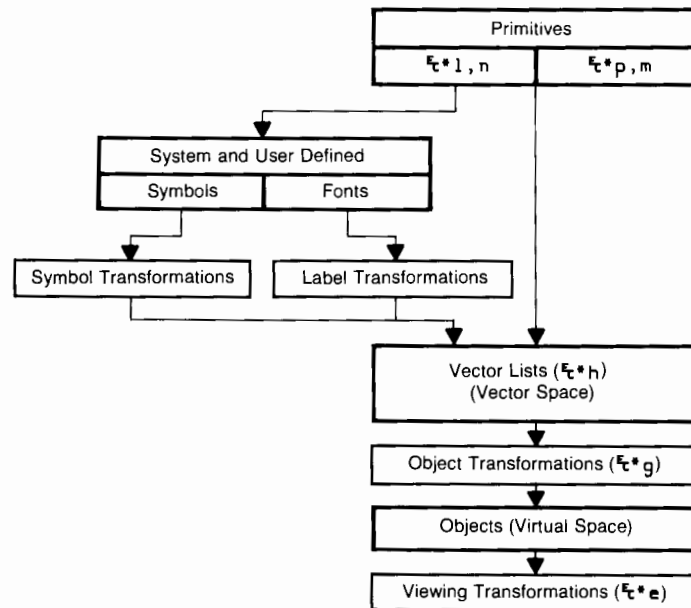


Figure 4-1. Creating a Picture File

Table 4-1. Picture File Creation Escape Sequences

<i>Vector List Commands</i>	
␣*ha	Define hidden vector list
␣*hb	Clear vector list
␣*hc	Close vector list
␣*hd	Delete vector list
␣*he	Delete vector list and all associated object attribute lists
␣*hl	Delete vector list library
␣*hn	Rename vector list
␣*ho	Replace vector list's references
␣*hs	Set vector list name for status request
␣*h^	Inquire vector list
<i>Object Commands</i>	
␣*ga	Activate Object
␣*gb	Set default attributes of active object
␣*gc	Create object (use with ␣*ga)
␣*gd	Delete object attribute list
␣*gh	Set object highlighting
␣*gk	Draw object
␣*gl	Delete all object attribute lists
␣*gn	Rename object attribute list
␣*go	Replace object vector list reference
␣*gp	Set pick priority
␣*gr	Rotate object
␣*gs	Scale object
␣*gt	Translate object
␣*gu	Undraw object
␣*gv	Set object visibility
␣*gw	Set object write mask
␣*gx	Set transformation center
␣*g^	Inquire object
<i>View Commands</i>	
␣*ea	Set auto redraw
␣*eb	Set viewport background color
␣*ed	Delete view from memory
␣*eh	Highlight view
␣*ei	Activate view
␣*el	Delete all inactive views
␣*em	Set redraw mode

Table 4-1. Picture File Creation Escape Sequence (Cont.)

$\text{\textbackslash}^*er$	Redraw view
$\text{\textbackslash}^*ev$	Set viewport limits
$\text{\textbackslash}^*ew$	Set window limits
$\text{\textbackslash}^*e^{\wedge}$	Inquire view
<i>User Defined Character/Font Commands</i>	
$\text{\textbackslash}^*nb$	Define character cell
$\text{\textbackslash}^*nc$	Plot character data
$\text{\textbackslash}^*nd$	Define character coordinate
$\text{\textbackslash}^*ne$	Delete user defined font(s)
$\text{\textbackslash}^*nf$	Select text font

VECTOR LISTS

Definition

A vector list is a data structure composed of drawing elements which define a graphical entity. A vector list is not displayed on the screen until it is linked with object attributes to form an object. One vector list may be used to create multiple objects all with the same form but with differing scale, rotation, and translation factors.

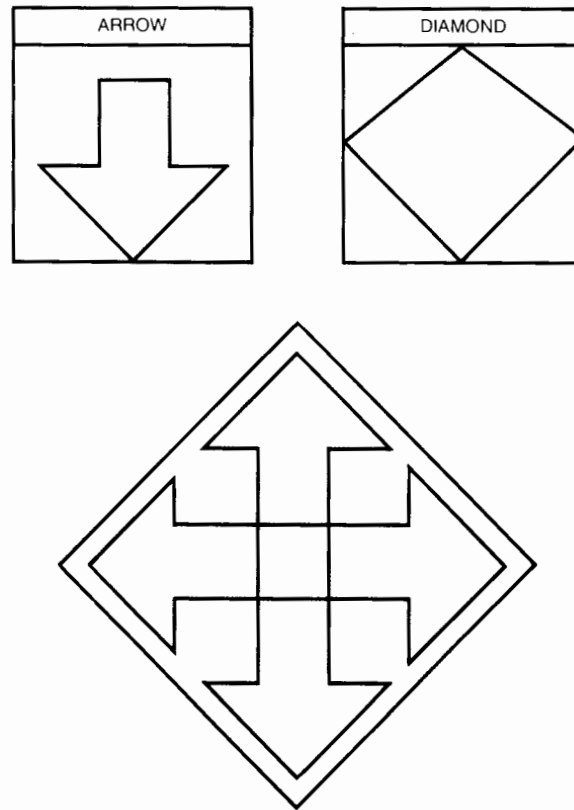


Figure 4-2. Two Vector Lists; Five Objects

Vector List Elements

Vector lists are composed of drawing elements called primitives (vectors, area fills, labels) and primitive attributes (pens, line types, area fill patterns, label attributes, et cetera.). The following is a list of escape sequences used to enter primitives and primitive attributes into a vector list. Most of these commands are explained in Section 2. DRAWING FUNDAMENTALS.

Table 4-2. Vector List Primitives and Primitive Attributes

<i>Plotting Commands</i>	
ϵ^*pa	Lift pen
ϵ^*pb	Lower pen
ϵ^*pc	Use graphics cursor as new pen point
ϵ^*pd	Draw point at pen position
ϵ^*ps, pt	Begin/end polygon area fill
ϵ^*pu, pv	Lift/lower boundary pen
ϵ^*me, mf	Absolute/relocatable rectangular area fill
<i>Vector Drawing Modes</i>	
ϵ^*ma	Select drawing mode
ϵ^*mb	Select line type
ϵ^*mc	Define user line pattern
ϵ^*md	Define user area fill pattern
ϵ^*mg	Select area fill pattern
ϵ^*mh	Select boundary pen number for area fill
<i>Graphics Text and Attributes</i>	
ϵ^*mm	Specify text size (multiplier)
ϵ^*ns	Specify text size (absolute)
ϵ^*mn	Specify text rotation (90 degree increments)
ϵ^*nr	Specify text rotation (arbitrary angle)
ϵ^*mo, mp	Turn on/off text slant
ϵ^*mq	Specify text origin and justification
ϵ^*nm	Specify text margins
ϵ^*nw	Specify character spacing
ϵ^*nx	Specify text pen
ϵ^*ds, dt	Turn on/off graphics text mode
ϵ^*l	Graphics label
<i>Symbols</i>	
ϵ^*ms	Set symbol size
ϵ^*mt	Set symbol mode
ϵ^*mu	Set symbol units
ϵ^*nf	Select user defined font

Table 4-2. Vector List Primitives and Primitive Attributes (Cont.)

<i>Pens</i>	
$\epsilon * mh$	Select boundary pen number for area fill
$\epsilon * mx$	Select primary pen
$\epsilon * my$	Select secondary pen
$\epsilon * nx$	Select text pen
 <i>Pick ID</i> (See Section 7. <i>SYSTEM INQUIRY/STATUS REQUESTS</i> .)	
$\epsilon * mi$	Set Pick ID

Vector List Structure

Vector lists consists of a header, primitives, and primitive attributes. A vector list "header" contains the values of all primitive attributes in effect when the list was initially created. For example, if the last primary pen used in a previously created vector list was pen 4, then pen 4 is listed in the header of the newly created vector list. It will be used to draw all subsequent vectors until it is explicitly changed by the $\epsilon * mx$ command. When the terminal is turned on, the values in the vector list header are the system default values which are listed in "Primitive Defaults" in this section.

Figure 4-3 shows the structure of a vector list.

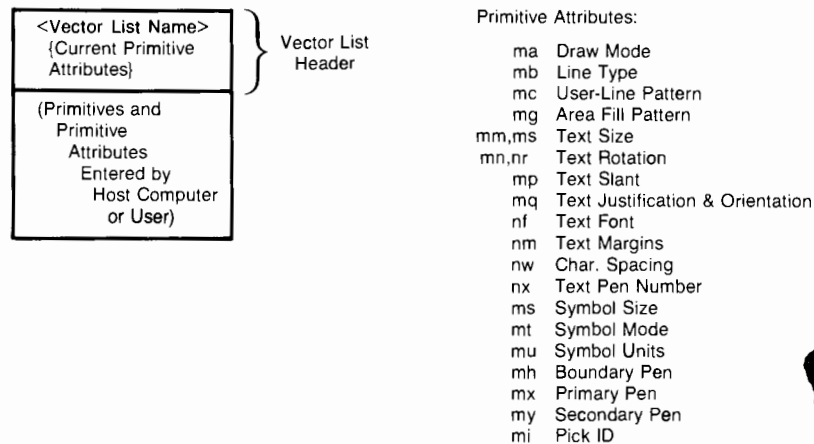


Figure 4-3. Vector List in Terminal Memory

How To Create a Vector List

There are 3 steps involved in creating a vector list:

- Defining the vector list type
- Entering the drawing elements (primitives)
- Closing the vector list

DEFINING THE VECTOR LIST TYPE. The terminal always has one (and only one) *open* vector list; that is, a vector list ready to accept and store primitives. The open vector list is one of two types, *displayed* or *hidden*.

Displayed Vector Lists. A displayed vector list is open and ready to receive primitives when the terminal is turned on. As primitives are stored in the displayed vector list, they may be viewed on the screen through a set of object attributes maintained by the terminal. The displayed vector list and terminal maintained object attributes form a "psuedo object". (The psuedo object is discussed in the section on object attributes.)

Displayed vector lists are most likely to be used by someone who is entering primitives from the keyboard. This arrangement is also beneficial for running programs written for other Hewlett Packard graphics terminals.

Hidden Vector Lists. Hidden vector lists, unlike displayed vector lists, must be defined explicitly using the `␣*ha` command (explained below). Hidden vector lists save all incoming primitives in vector memory, but do not display them on the screen.

Define hidden vector list: `␣*ha`

A new hidden vector list is opened and the current (last entered) primitive attributes are copied into the header. All subsequent primitives and primitive attributes are stored in this vector list until it is closed.

NOTE: These primitives may not be viewed until linked with object attributes.

It is faster to use hidden vector lists than displayed vector lists when you are downloading primitives from a host computer or disc file.

ENTERING PRIMITIVES. Once a vector list is open, you enter primitives and primitive attributes using the escape sequences listed in table 4-2. If at any time you wish to clear the vector list and start over, use the command `␣*hb`. The vector list will be cleared, and the last primitive attribute values entered will be stored in the header.

“CLOSING” THE VECTOR LIST. After the primitives and primitive attributes have been entered, the vector list must be “closed” in order to save it in the picture file. (This applies both to displayed and hidden vector lists.) When a displayed vector list is closed, it disappears from the screen. When any vector list is closed, a new *displayed* vector list is automatically opened.

Close Vector List: `␣*h<<vector list name>>C`

where `<vector list name>` is a string of up to 12 characters; the first character being alpha and subsequent characters being any from the set (0-9,A-Z,a-z,_). The `<vector list name>` must be enclosed by angle brackets.

To close a vector list with the name “square”, you would use the following command:

`␣*h<square>C`

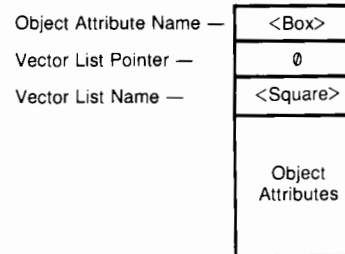
Note that the case of the characters does not distinguish two otherwise like-named vector lists.

ADDITIONAL NOTES

- Once a vector list is closed, it may not be reopened or edited. If you make a mistake, you must delete the vector list and start over.
- If you open a new vector list without closing the first one, all primitives from the first vector list will be lost. The current primitive *attributes* will be copied into the header of the newly created displayed vector list.

- If object attributes exist in the picture file which contained references to a vector list which did not formerly exist, they will be assigned to this vector list after it is created. (i.e., If a set of object attributes "box" had been created with a reference to the vector list "square" (which did not formerly exist), the vector list pointer in "box" would now point to "square". See figure 4-4.)

First Case: Vector list "square" does not exist in the picture file. The vector list pointer for object attributes "box" is nil.



Second Case: Vector list "square" is created. Vector list pointer for object attributes - "box" - now points to "square".

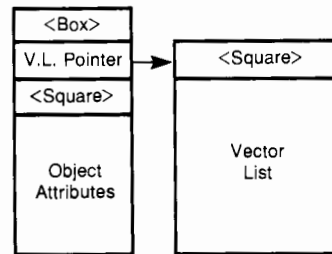


Figure 4-4. Storing Vector Lists in Vector Memory

- If a vector list name is not supplied in the `ε*hc` command, the current vector list is aborted, and a new displayed vector list is created with current primitive attributes copied into the header.

Example. Create a vector list "square". Fill in one half of the square (using solid area fill) in pen 3 and the other half of the square in pen 4.

Step 1. Perform a hard reset of the graphics memory.

`ε*wR`

The default view will be defined. Recall that a displayed vector list is already defined.

Creating a Picture

Step 2. Enter the primary pen for the first half of the square.

```
⌘*m3X
```

Step 3. Enter the data points for the first area fill. (NOTE: Solid area fill is a default value, so we need not select it explicitly.)

```
⌘*pas 40 40 90 40 90 240 40 240 T
```

Step 4. Enter the primary pen for the second half of the square.

```
⌘*m4X
```

Step 5. Enter the data points for the second area fill.

```
⌘*pas 90 40 140 40 140 240 90 240 T
```

Step 6. Close the vector list by naming it "square".

```
⌘*h<square>C
```

The image on your screen should disappear at this point.

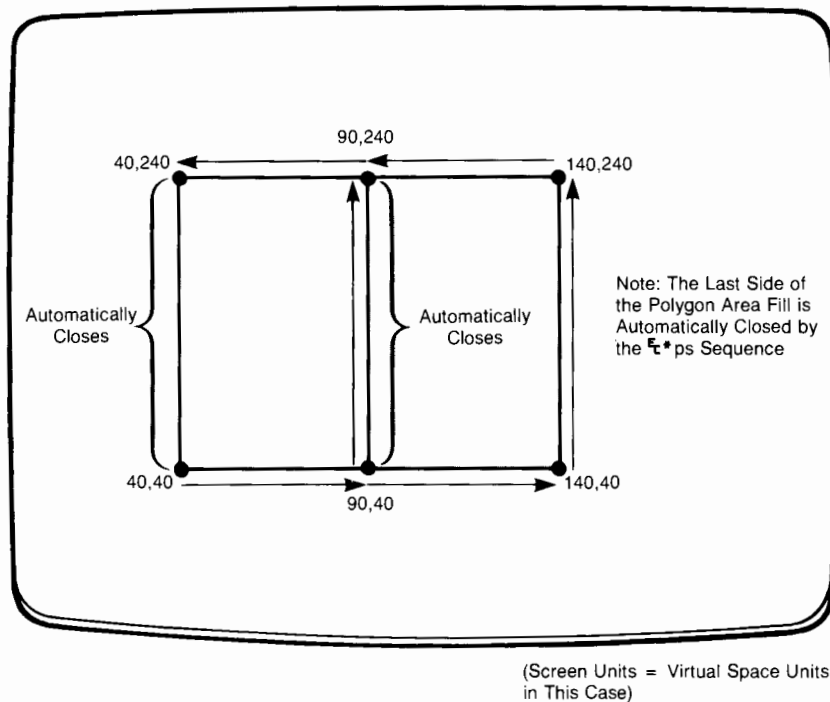


Figure 4-5. Vector List Example

The vector list is now named and stored in the terminal for future use. It can be referenced by object attributes to create different objects. After the vector list has been closed, a new displayed vector list is opened, and the primitive attributes of the last vector list are copied into the header.

Primitive Defaults

The default settings for primitive attributes are set using the escape sequence:

```
␣*m1r
```

These defaults are:

Move/draw flag	"Draw"
Line type	1(solid)
Drawing mode	2(Jam1)
User defined line pattern	255,1
User defined area pattern	255,255, . . . , 255
Primary Pen	15
Secondary Pen	0
Pick ID	0
Text size	1
Text direction	1
Text origin	1(left,bottom)
Text slant	off
Text angle	0 deg
Character Spacing	0
Text Pen	7(on)
Symbol mode	1(off)
Symbol scale	1.0
Symbol units	0(display)

Note that this command is very similar to `␣*mr` (reset drawing primitives). If you execute the `␣*mr` command while creating a vector list, it will be saved as `␣*m1r` so as not to disturb the graphics cursor, graphics video, et cetera, each time the vector list is redrawn.

See also:

Section 3. COLOR

`␣*vr` — Reset default palette colors.

Section 4. CREATING A PICTURE

`␣*gb` — Reset object attribute defaults for active object.

`␣*eq` — Reset active view to default values.

Section 5. PICTURE FILE COMMANDS

`␣*wr` — Perform graphics hard reset.

Section 6. RASTER MANIPULATIONS

`␣*ri` — Reset raster default values.

Vector List Library Commands and Status Requests

The following commands are used to maintain vector list libraries. (See Section 5. PICTURE FILE COMMANDS.)

- `␣*h<<vector list name>>d` — Delete vector list.
- `␣*h1` — Delete all vector lists in the picture file.
- `␣*h<<new vector list name>><<old vector list name>>n` — Rename vector list.
- `␣*h<<new vector list name>><<old vector list name>>o` — Change vector list name, and update all object attributes which reference this vector list.

The following vector list inquiry and status requests are explained in detail in Section 8. INQUIRY COMMANDS AND STATUS REQUESTS.

- `␣*h<<vector list name>>s` — Specify vector list name for status request.
- `␣*h<mask>^` — Request vector list status information.

OBJECT ATTRIBUTES, OBJECTS

Definition

The image stored in a vector list cannot be viewed on the screen until object attributes have been applied to it. Object attributes translate the image from vector space to virtual space where the user may choose to display it.

The combination vector list + object attributes is referred to as an object. By applying different sets of object attributes to one vector list, you can create several objects.

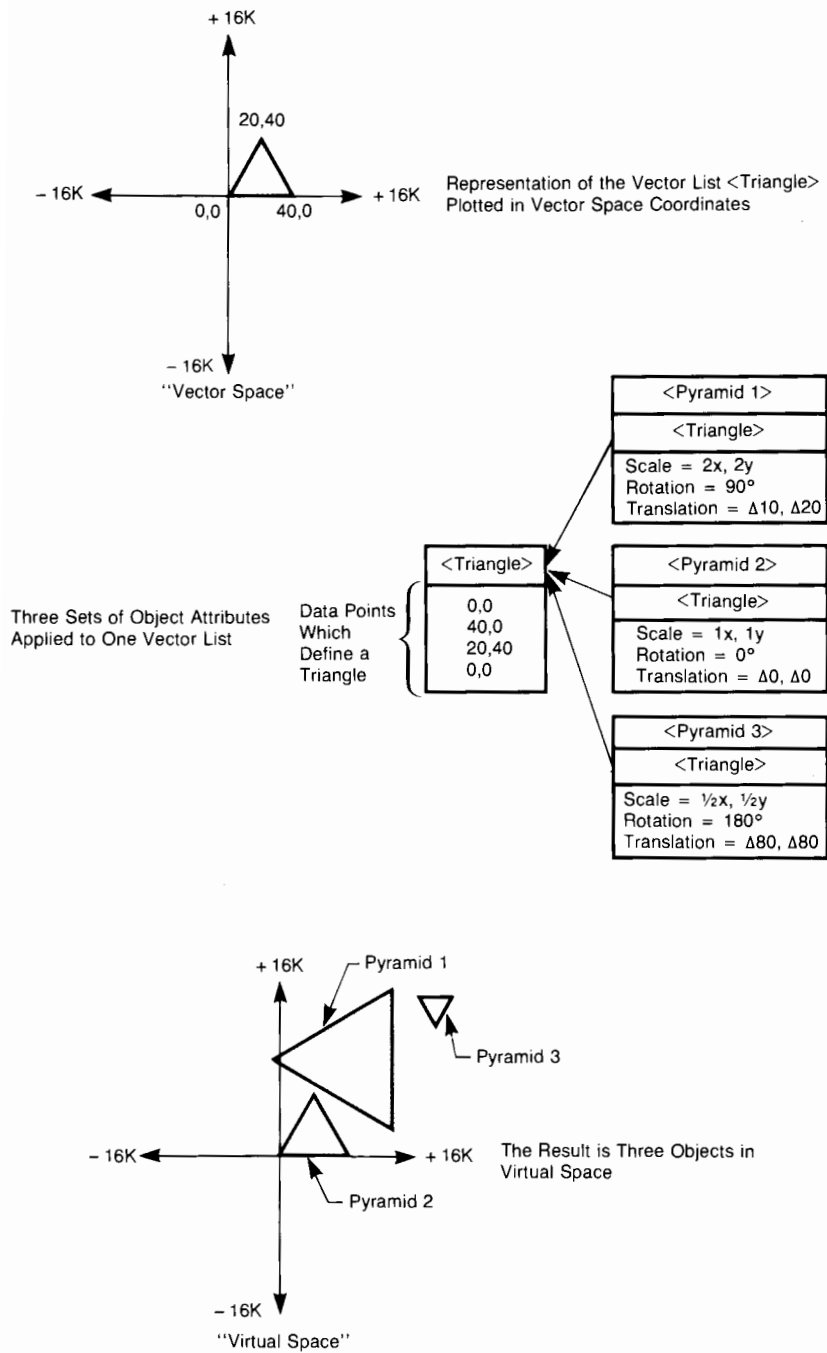


Figure 4-6. Vector Lists, Object Attributes, and Objects

How to Create/Activate an Object

An object is created by using an escape sequence which links the object attributes with the desired vector list. The format is:

```
␣*g<<object attributes name>>a<<vector list name>>C
```

where <object attributes name> is a string with a maximum of 12 characters — the first character must be alpha and subsequent characters are any from the set (0-9, A-Z, a-z, _). *Object attribute names and vector list names must all be unique.*

When an object is initially created, *the object attributes applied to the vector list will be those of the pseudo object until they are explicitly changed by the user.* See “System Object Attributes and the Psuedo Object” in this section.

To create the object <pyramid1> from vector list <triangle>, you would enter the command:

```
␣*g<pyramid1>a<triangle>C
```

Object attributes may be modified, or they may be applied to a new vector list. You can only modify the attributes of the “active” object. To activate an object, you address its object attributes by name in the command:

```
␣*g<<object attribute list name>>A
```

All subsequent object attributes commands (␣*g) affect the active object.

NOTE: These object attributes need not reference a vector list for the command to be effective.

Object Attributes

Each object has an associated list of object attributes. These attributes consist of the following:

- Transformation Attributes (transformation center, scaling, rotation, translation)
- Highlighting
- Pick Priority
- Visibility
- Write Mask

THE TRANSFORMATION ATTRIBUTES. The attributes which specify the position of the vector list in virtual space are:

- transformation center (X,Y)
- scaling (X and Y multipliers)

- rotation (run-rise or theta in radians)
- translation ($\Delta X, \Delta Y$)

The scaling, rotation, and translation of an object take place (in that order) about the transformation center. The equations for these transformations may be found in Appendix B.

Object Transformation Center

$$\text{t}^*g\langle X \rangle \langle Y \rangle X$$

where $\langle X \rangle, \langle Y \rangle$ are integers in the range $(-16383, \dots, +16383)$ (the limits of virtual space)

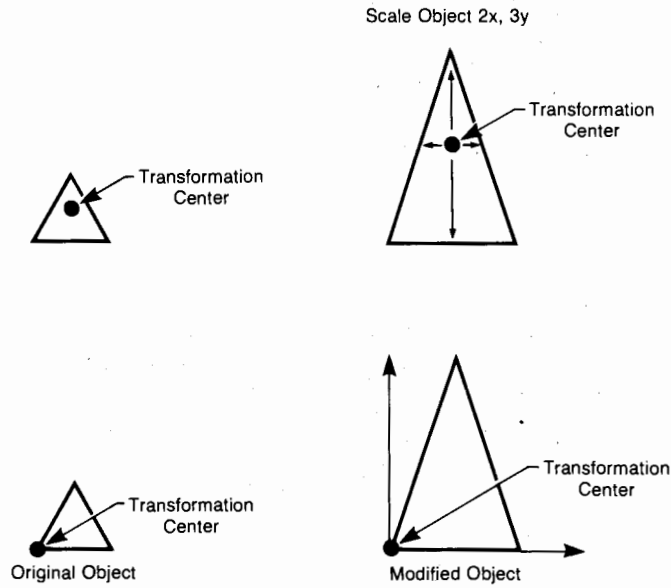
This command defines the object's transformation center. Scaling, rotation, and translation will take place about this point.

Object Scaling

$$\text{t}^*g\langle Sx \rangle \langle Sy \rangle S$$

where $\langle Sx \rangle, \langle Sy \rangle$ are real numbers in the range $(-127.996, \dots, +127.996)(.004)$

This command defines the object's X and Y scale factors.



The Object is Scaled Outward From the Transformation Center.

Figure 4-7. Set Object Scaling

Creating a Picture

NOTE: When the scale factor is negative, the object is reflected as follows:

Negative x scale factor = object reflected about y axis
Negative y scale factor = object reflected about x axis
Negative x and y scale factor = object reflected about origin

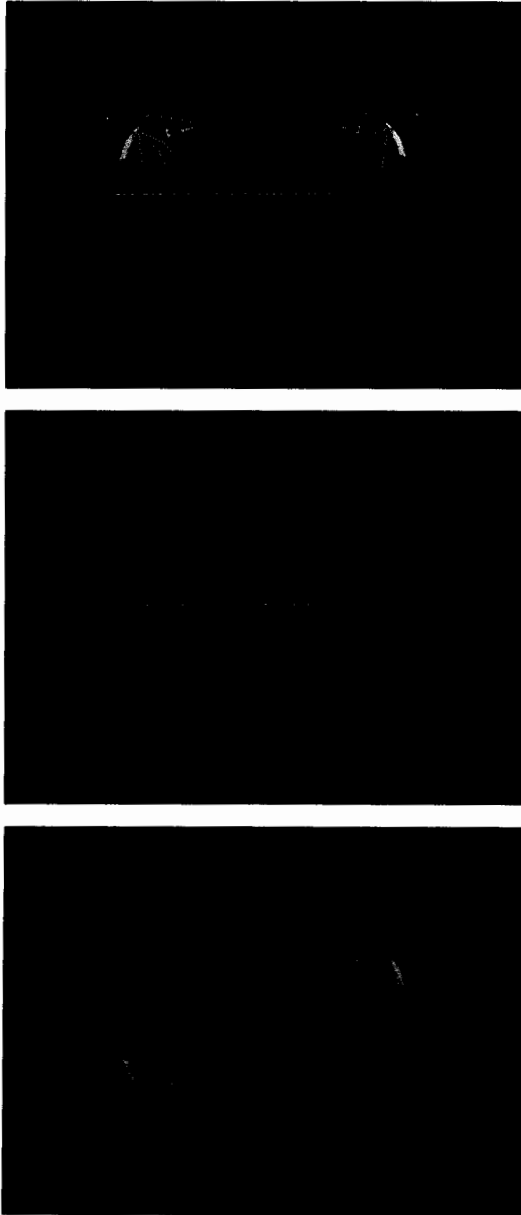


Figure 4-8. The Effects of Negative Scaling

Object Rotation. You may use either of two commands to rotate an object:

$$(1) \text{ } \epsilon * g \langle \text{run} \rangle , \langle \text{rise} \rangle r$$

where $\langle \text{run} \rangle , \langle \text{rise} \rangle$ are real numbers in the range $(-127.996, \dots, +127.996)(.004)$

This command defines the rotation of the object by using a direction vector. The active object will be rotated through the angle described by the direction vector relative to the X axis.

$$(2) \text{ } \epsilon * g \langle \text{theta} \rangle r$$

where $\langle \text{theta} \rangle$ is a real number in the range $(-127.996, \dots, +127.996)(.004)$

This command rotates the object $\langle \text{theta} \rangle$ radians counter-clockwise.

NOTE: $1 \text{ radian} = \frac{360 \text{ degrees}}{2 \pi (= 6.28318)}$

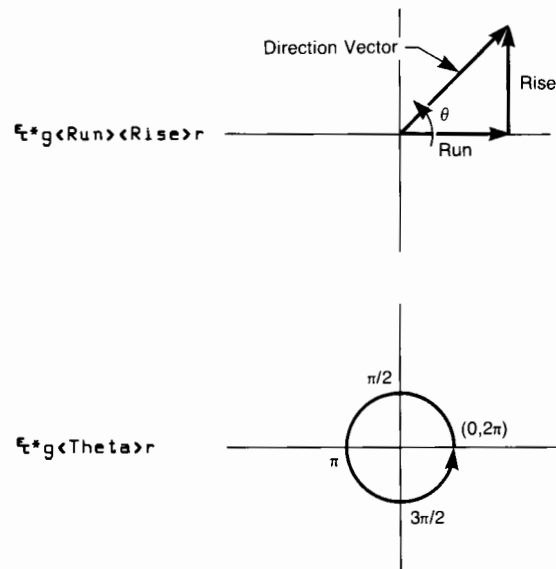


Figure 4-9. Object Rotation Commands

Creating a Picture

Object Translation

`ε*g<Tx><Ty>T`

where $\langle Tx \rangle, \langle Ty \rangle$ are integers in the range $(-32766, \dots, +32766)$

This command specifies the translation of the object in virtual space (by $\langle Tx \rangle$ and $\langle Ty \rangle$) relative to the original points of the object.

Example. Using the vector list `<triangle>` illustrated in figure 4-6, create an object `<pyramid1>` with the following object attributes:

- Transformation center in the center of the triangle
- Scale factors of 2X and 3Y
- Rotation of +90 degrees
- Translation of $\Delta 50$ (X direction) and $\Delta 100$ (Y direction)

Step 1. Create the vector list.

`ε*pas 0,0 40,0 20,40 0,0T`

`ε*h<triangle>C`

Step 2. Create the object.

`ε*g<pyramid1>a<triangle>C`

Step 3. Set the transformation center at the center of the triangle.

`ε*g 20 20X`

NOTE: There is no change in object appearance.

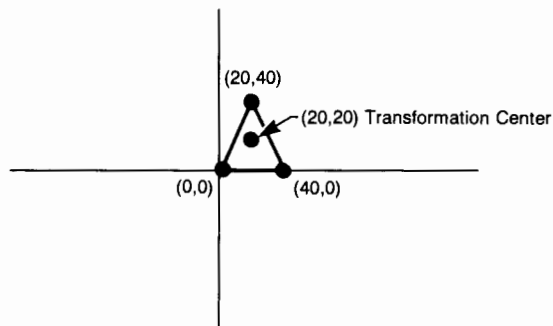


Figure 4-10. Set Transformation Center (Example 1)

Step 4. Scale the triangle by a factor of 2 in the X direction and 3 in the Y direction.

`t*g 2 3 S`

Note that the triangle is scaled outward from the transformation center.

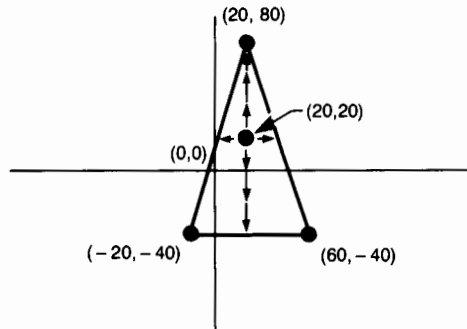


Figure 4-11. Specify Scaling Factors (Example 1)

Step 5. Rotate the triangle +90 degrees.

`t*g 0 1 R(run, rise)`

or

`t*g 1.5706 R(radians)`

The triangle is rotated about its center.

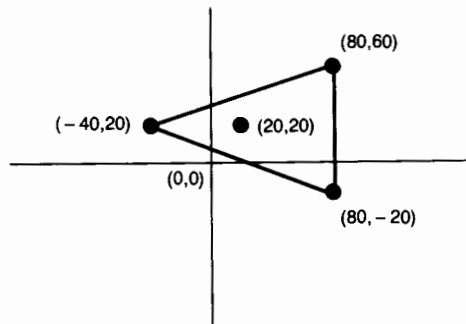


Figure 4-12. Specify Rotation (Example 1)

Step 6. Translate the triangle 50 points in the X direction and 100 points in the Y direction.

```
␣*g 50 100 T
```

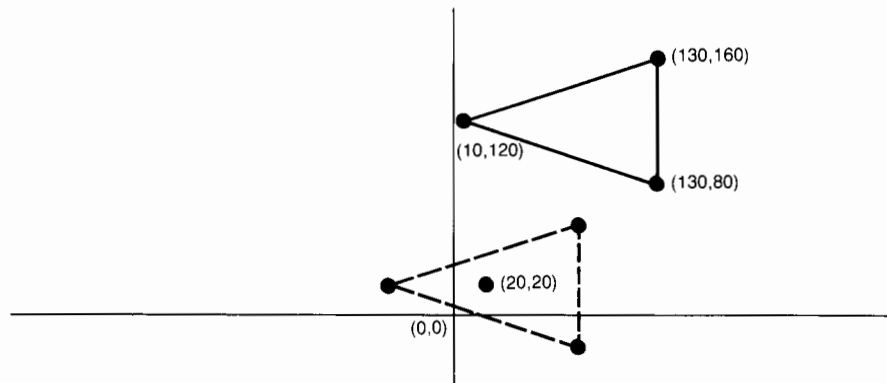


Figure 4-13. Specify Translation (Example 1)

Solution:

```
␣*g<pyramid1>a<triangle>c20 20x2 3s0 1r50 100T
```

The pyramid is now a viewable object in virtual space.

Example 2. Reactivate the object <pyramid1>. Repeat Example 1 using a transformation center at $-10, -10$.

Step 1. Reactivate the object.

```
␣*g<pyramid1>A
```

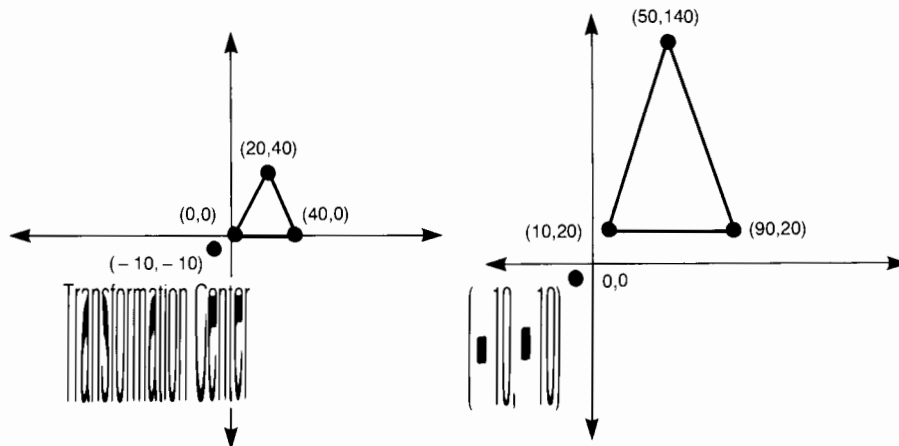
Step 2. Reset the object attributes to their default values.

```
␣*gb
```

Step 3. Apply object transformations.

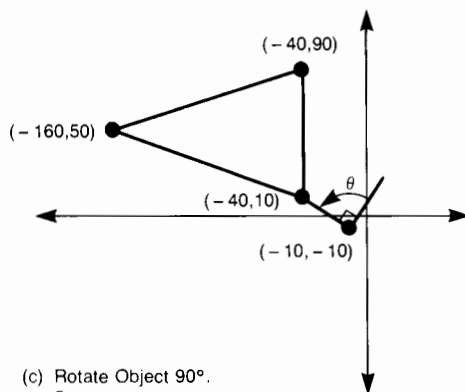
```
␣*g -10 -10x 2 3s 0 1r 50 100T
```

Step 4. If auto redraw is on (see "Redrawing the Active View" in this section), these effects take place immediately without invoking an explicit redraw.

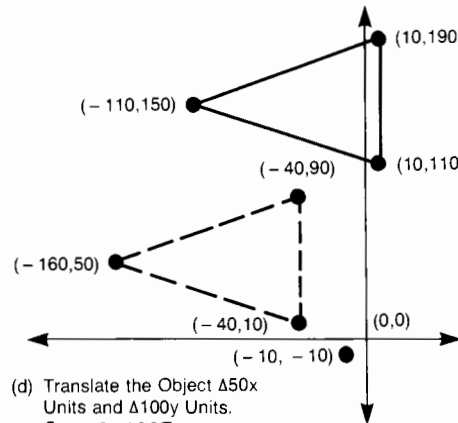


(a) Set Transformation Center at $(-10, -10)$
 $\text{Ft} * \text{g} -10, -10 \text{x}$

(b) Scale Object $2\text{x}, 3\text{y}$
 $\text{Ft} * \text{g} 2, 3 \text{S}$
 The Object is Scaled Outward From the Transformation Center



(c) Rotate Object 90° .
 $\text{Ft} * \text{g} 0, 1 \text{R}$
 The Object is Rotated About the Transformation Center



(d) Translate the Object $\Delta 50\text{x}$ Units and $\Delta 100\text{y}$ Units.
 $\text{Ft} * \text{g} 50, 100 \text{T}$

Figure 4-14. Example 2

OBJECT HIGHLIGHTING. You can highlight an object by redrawing it in complement mode — drawing mode 3. (See Section 6. RASTER MANIPULATIONS, “Vector Drawing Modes”.)

Set Object Highlighting: $\text{Ft} * \text{g} \langle \text{mode} \rangle \text{h}$

where $\langle \text{mode} \rangle = 0$ (disabled)
 1 (enabled)

PICK PRIORITY. The terminal has an "Inquire Pick" command which returns information about an object "picked" by the graphics cursor. An object's "pick priority" resolves an ambiguous pick (among objects in the same vicinity) in favor of the object with the highest pick priority. (See Section 8. INQUIRY COMMANDS AND STATUS REQUESTS.)

Set Pick Priority: $\text{t}^*g\langle\text{value}\rangle p$

where $\langle\text{value}\rangle$ is an integer in the range $(0, \dots, +32767)$

OBJECT VISIBILITY. Objects are drawn in order of increased visibility, thus objects with a high visibility may be drawn over objects with a low visibility.

Set Object Visibility: $\text{t}^*g\langle\text{value}\rangle v$

where $\langle\text{value}\rangle$ is an integer in the range $(-32768, \dots, +32767)$

There is no way to determine the drawing order of objects with the same visibility. Objects with a visibility of zero (or a visibility lower than the object visibility fence) are not displayed on the screen. (See Section 5. PICTURE FILE COMMANDS, "Object Visibility Fence".)

NOTE: The visibility of the pseudo object may be changed, however, the pseudo object will always be drawn as the most visible object on the screen. Its visibility will affect subsequent objects which are always initially created with the current object attributes of the pseudo object.

OBJECT WRITE MASK. You can control which raster planes store an object by setting an object write mask. (See Section 5. RASTER MANIPULATIONS, "Object Write Masks".)

Set Object Write Mask: $\text{t}^*g\langle\text{mask}\rangle w$

where $\langle\text{mask}\rangle$ is an integer in the range $(-32768, \dots, 32767)$

Default Object Attributes

The initial default values of object attributes are:

- Transformations
 - Transformation center 0,0
 - Scaling 1,1
 - Rotation 0
 - Translation 0,0
- Visibility 1
- Pick Priority 1
- Highlighting .off
- Object Write Mask 15

To reset the object attributes of an object to its default values, execute the command:

`⌘*gb`

How Objects are Stored in the Picture File

Figure 4-15 shows how object attributes and their associated vector lists are stored in vector memory.

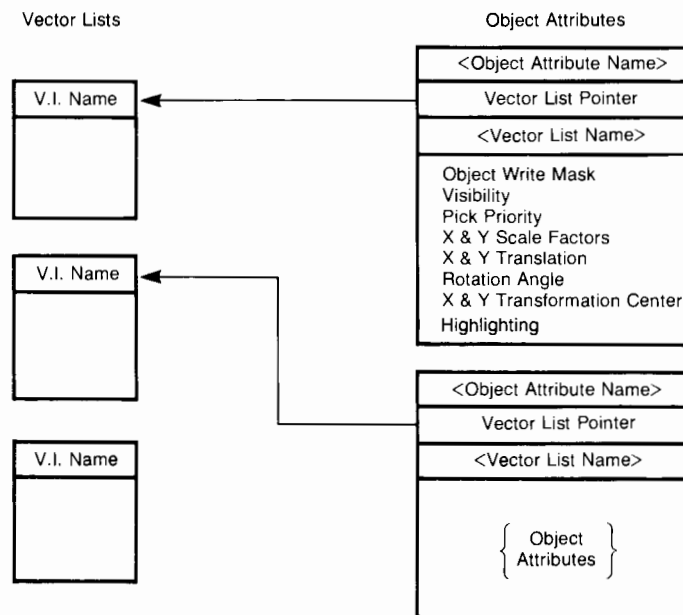


Figure 4-15. Objects in Vector Memory

System Object Attributes and the Pseudo Object

The terminal maintains a set of object attributes at all times through which the displayed vector list may be viewed. The displayed vector list and system object attributes form a "pseudo object". When the displayed vector list is closed with the `␣*h<<vector list name>>C` command, the bond between the vector list and system object attributes is broken. The system object attributes become attached to the new displayed vector list, (unless a hidden vector list is subsequently defined), to form a new pseudo object. This pseudo object is created with system default values, all of which may be altered by the user.

To change the system object attributes, use the command:

```
␣*ga
```

Example. Change the scale factor of the system object.

```
␣*ga 2 2 5
```

These values will remain in effect until they are explicitly changed by the user or a graphics hard reset is performed.

NOTE: All objects are initially created with the object attribute values of the pseudo object.

Selective Object "Undraw" and "Redraw"

Normally, in order to perceive a change to the picture file, you must redraw the entire

active view. Because this procedure may be inconvenient for detailed drawings, the terminal has selective object draw and undraw functions.

Draw Object: `␣*gk`

Undraw Object: `␣*gu`

The following steps describe how to perform the selective erase function:

Step 1. Activate the object with the `␣*g<<object attributes name>>a` command.

Step 2. Execute the Undraw Object Command. The object will be redrawn with the current viewport background pen.

Step 3. Modify the object.

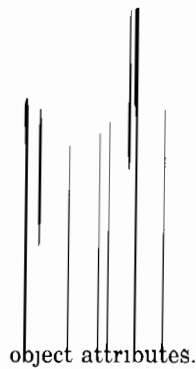
Step 4. Execute the Draw Object Command. The modified object will appear on the screen.

NOTE: If the object to be erased overlaps another object, part of the second object will be erased, too. You must then activate the second object, and execute the Draw Object command. Sometimes an entire redraw of the active view is the best recourse. See "Redrawing the Active View" in this section.

Object Attribute Library Commands and Status Requests

The following commands are used to maintain object libraries. (See Section 5. PICTURE FILE COMMANDS for details.)

- t^*gd — Delete active set of object attributes. (NOTE: You cannot delete the system set of object attributes.)
- t^*gl — Delete all sets of object attributes in the picture file, (except the system set of object attributes).
- $\text{t}^*\text{g}\langle\langle\text{new object attribute name}\rangle\rangle\text{n}$ — Rename the active set of object attributes.
- $\text{t}^*\text{g}\langle\langle\text{vector list name}\rangle\rangle\text{o}$ — Replace the vector list reference of the active set of



The following status command is discussed in detail in Section 8. SYSTEM INQUIRY/STATUS REQUESTS.

- $\text{t}^*\text{g}^{\wedge}$ — Inquire object.

VIEWS

Definition

A view is the mapping of the contents of virtual space to an area of the screen. In order to modify a view, you must first specify the area of virtual space you wish to see. This area is called the "virtual window"; and it may be all or just a portion of virtual space.

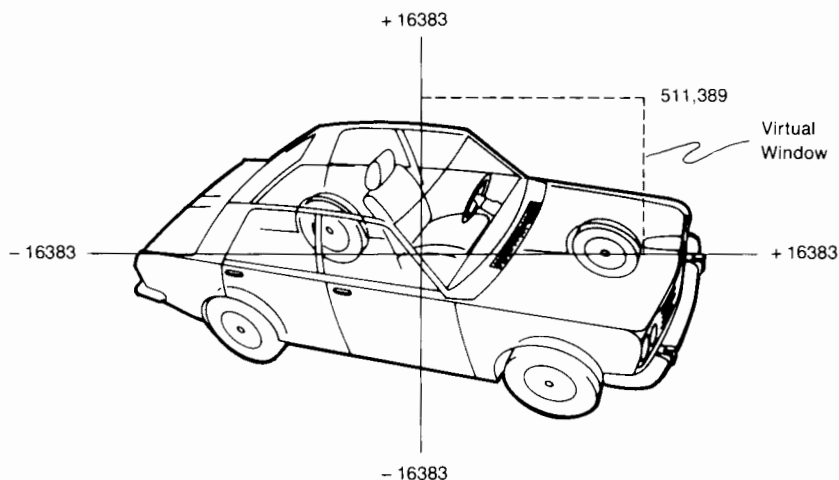


Figure 4-16. Virtual Window

Creating a Picture

Second, you must define the section of the screen in which to display the data of the virtual

window. This area is referred to as the "screen viewport".

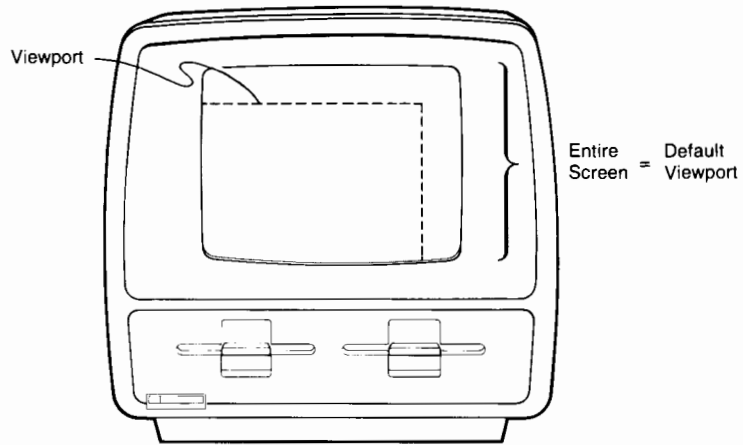


Figure 4-17. Screen Viewport

The virtual window/screen viewport pair is a view. Data which falls outside the virtual

window is "clipped". Figure 4-18 shows the view defined by the above examples.

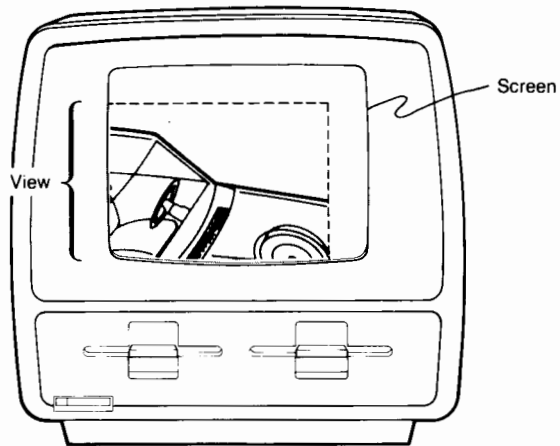


Figure 4-18. View

How to Define a View

CREATING/ACTIVATING A VIEW. The terminal allows you to store up to 256 views. Each view has a reference number, just as vector lists and object attributes have names.

The system always has one and only one active view at a time. The term “active” implies that subsequent view commands ($\text{f}*\text{e}$) will affect this view.

A new view is created, or an existing view is activated by the command:

```
 $\text{f}*\text{e}\langle\text{view number}\rangle\text{i}$ 
```

where $\langle\text{view number}\rangle$ is an integer in the range (0, . . . , 127)(128, . . . , 255)

NOTES:

- Views 128, . . . , 255 are reserved for system use. You can create them, but you cannot save them to a disc or datacomm file when the picture file is copied. See Section 5. PICTURE FILE COMMANDS.

Creating a Picture

- If you try to activate an undefined view, a new view (with that number) is created with the system default values.
- At power on, the default view is 0.

SETTING VIRTUAL WINDOW/SCREEN VIEWPORT LIMITS. The virtual window is created by defining its lower left (ll) and upper right (ur) boundaries in the escape sequence:

```
^*e<xll> <yll> <xur> <yur>w
```

where <xll>,<yll>,<xur>,<yur> are integers in the range (-16383, . . . , +16383)

The screen viewport is created in a similar fashion using the escape sequence:

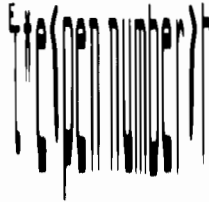
```
^*e<xll> <yll> <xur> <yur>v
```

where <xll>,<yll>,<xur>,<yur> are integers in the range (0, . . . , 511)†

At power on, the default values for these settings are:

```
virtual window=screen viewport=0,0,511,389
```

VIEWPORT HIGHLIGHTING AND BACKGROUND COLOR. The user may choose to outline a view in any pen number he wishes. This is called "highlighting", and is invoked by the command:



where <pen number> is an integer in the range (-32768, . . . , 32767); the low four bits are used to determine the pen value.

If no <pen number> is specified, highlighting is turned off.

The user may also choose to set the raster memory within the boundaries of the viewport to a particular color before redrawing the objects. The command which sets the viewport background color is:

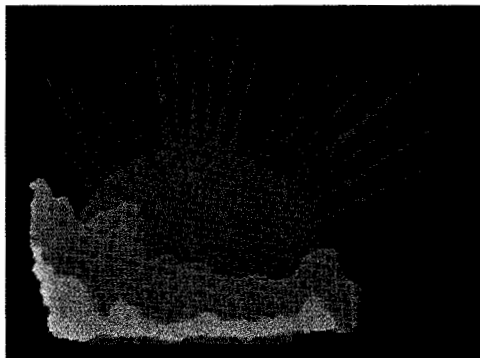
```
^*e<pen number>b
```

where <pen number> is an integer in the range (-32768, . . . , 32767); the low four bits are used to determine the pen value.

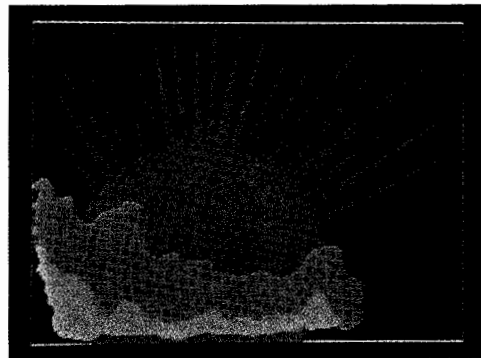
† The limits of raster memory are 512×512 , although only 512×390 pixels are displayable on the screen. The portion of raster memory between 0,390 and 512,512 may be viewed through certain external monitors and cameras or via a raster dump.

NOTE: A redraw mode must be enabled to redraw the view in the background color. See "Redraw Modes" in this section.

Both the highlighting and background color commands take effect immediately if auto redraw is on. (See "Redrawing the Active View" in this section.)



Normal



Highlighting



Background Color

Figure 4-19. View Highlighting and Background Color

View Default Attributes

The default view attributes are:

- virtual window: 0,0,511,389
- screen viewport: 0,0,511,389
- highlighting: off
- viewport background color: pen 0

To reset the active view to default values use the command:

```
τ*eq
```

Redrawing the Active View

Changes in the picture file are not reflected in the active view unless auto-redraw mode is enabled or an explicit redraw is performed. The auto-redraw commands are:

Enable auto-redraw: `␣*e1a`

Disable auto-redraw: `␣*e0a`

If auto-redraw is off, you may invoke an explicit redraw with the command:

`␣*er`

When a redraw occurs, only the active viewport is redrawn.

Redraw Modes

The final result of a redraw command is dependent upon the redraw mode in effect. The redraw mode has three component states:

- The entire raster can be cleared to pen 0 (default black) before redrawing the picture.
- The active viewport can be set to the background color before redrawing the picture.
- If your terminal has an auxiliary raster buffer, you can put the terminal into "double buffer" mode; that is, while the current picture is displayed on the screen, the updated picture will be drawn on the non-displayed raster buffer. After the picture has been redrawn, the non-displayed raster buffer will be turned on and the displayed buffer will be turned off. (See Section 1. GRAPHICS CONCEPTS, "Double Buffer Mode".)
- The picture can be redrawn such that all area fills are drawn with halftone patterns. This allows you to *preview* a raster dump to printer with the halftone setting enabled. See Section 7. GRAPHICS INPUT/OUTPUT OPERATIONS for details.
- The terminal can be set to any combination of the above.

Redraw modes are set with the command:

`␣*e<mode>m`

where <mode> is an integer in the range (-32768, . . . , +32767)

The low three bits are used to determine the mode according to the table below:

<i>bit</i>	<i>meaning</i>
0	clear screen to pen 0
1	clear active screen viewport to background pen color
2	double buffer mode
3	halftone mode

Multiple Views

To display multiple views on the screen at one time, you must set bit 0 of the `ε*ε<mode>m` mask to zero to ensure that the screen will not be cleared when the active view is redrawn. Bit 2 should also be set to zero (disable double-buffer mode) so that all the views are drawn in the same raster buffer.

View Library Commands and Status Requests

The following commands are used to maintain view libraries. (See Section 5. PICTURE FILE COMMANDS.)

- `ε*ε<view number>d` — Delete view.

where `<view number>` is an integer in the range (0, . . . , 255)

NOTE: The view specified may not be the active view.

- `ε*εl` — Delete all inactive views.

The following status command is discussed in Section 8. INQUIRY COMMANDS AND STATUS REQUESTS.

- `ε*ε^` — Inquire view.

USER DEFINED CHARACTERS AND FONTS

Definition

A user defined character is a collection of moves, draws, and area fills which defines a symbol or letter. Up to 95 characters may be assigned to a "font". The terminal can store up to 14 user defined fonts depending upon the amount of available memory.

How to Design a Character

A character is defined in one continuous `εc*n` sequence. The general format is:

```
ε*n <character code>,<font number>a <character cell size>b
  <plot command>c <x1,y1>d <plot command>c <x2,y2>d... 0C
```

Each portion of this command is described in detail below.

SPECIFY THE CHARACTER CODE AND FONT NUMBER. Before a character is defined, it is assigned a character code and a font number. The character code is also associated with a graphic ASCII character according to the table below for use by graphics text.

Table 4-3. ASCII Codes for Symbols

	30	40	50	60	70	80	90	100	110	120
0	n/a	(2	<	F	P	Z	d	n	x
1	n/a)	3	=	G	Q	[e	o	y
2	space	*	4	>	H	R	\	f	p	z
3	!	+	5	?	I	S]	g	q	{
4	"	,	6	*	J	T	^	h	r	!
5	#	-	7	A	K	U	=	i	s	}
6	\$.	8	B	L	V	`	j	t	~
7	@	/	9	C	M	W	a	k	u	n/a
8	&	0	:	D	N	X	b	l	v	n/a
9	'	1	;	E	O	Y	c	m	w	n/a

The character code and font number are specified in the command:

```
␣*n<character code>,<font number>a
```

where <character code> is an integer in the range (32, . . . , 126);
and is an integer in the range (3, . . . , 16)

NOTES:

- Font 1 is the system font; font 2 is the roman extension font.
- Every character must be assigned to a font, but a font need not define every character.

DEFINE THE CHARACTER CELL. Like vector lists, a symbol is “drawn” in its own definition space called a *character cell*. Valid sizes for character cells are in the range of integers (−63, . . . , +63), and are specified using the escape sequence:

```
␣*n<xll><yll><xur><yur>b
```

where “ll” indicates lower left corner of cell, and “ur” indicates the upper right corner. The default size is −63, −63, +63, +63.

Variable cell sizes are useful for proportional spacing.

NOTE: The last character cell specification remains in effect until it is explicitly changed or a graphics hard reset is performed.

PLOT THE CHARACTER DATA. A character plotting command is specified by the sequence:

$$\text{\textbackslash} * n \langle \text{command} \rangle c$$

where $\langle \text{command} \rangle = (0,1,2,3)$

$\langle \text{command} \rangle$	<i>meaning</i>
1	Move To Next Coordinate (with pen up)
2	Move To Next Coordinate (with pen up)/Start Area Fill
3	End Area Fill/Move to Next Coordinate (with pen up)
0	End Symbol Definition

NOTES:

- Commands (0) and (2) perform an End Area Fill if an area fill has been started but not ended.
- Any character definition which is not terminated with the $\text{\textbackslash} * n 0 c$ (End Symbol Definition) command is thrown away.
- A character definition may contain a maximum of 256 coordinate pairs. The commands $\text{\textbackslash} * n 2 c$ and $\text{\textbackslash} * n 3 c$ also count against this maximum.
- The move to the first coordinate is always performed with the pen up. You do not need to specify $\text{\textbackslash} * n 1 c$ for the first coordinate.
- Area fills are always solid. They are not affected by the current area fill pattern or halftoning.
- An entirely new font is created whenever the first character in that font is created. Therefore, unless a space has been explicitly defined, (#32), a blank character is automatically created with a character cell size equal to that of the first character of that font to be defined. Note the space characters are the determining factor for $\text{\textbackslash} * n 5$ size specifications and for linefeed sizes.

Data coordinates for the character definition are supplied by the following command:

$$\text{\textbackslash} * n \langle x \rangle , \langle y \rangle d$$

where $\langle x \rangle$ and $\langle y \rangle$ are integers in the range $(-63, \dots, +63)$

Example. Define the character 'H' using the data points given in the figure 4-20. Assign it to character number 72 (see table 4-3) and font 8. Specify a character cell size of $(-10, -10 \ 22,25)$.

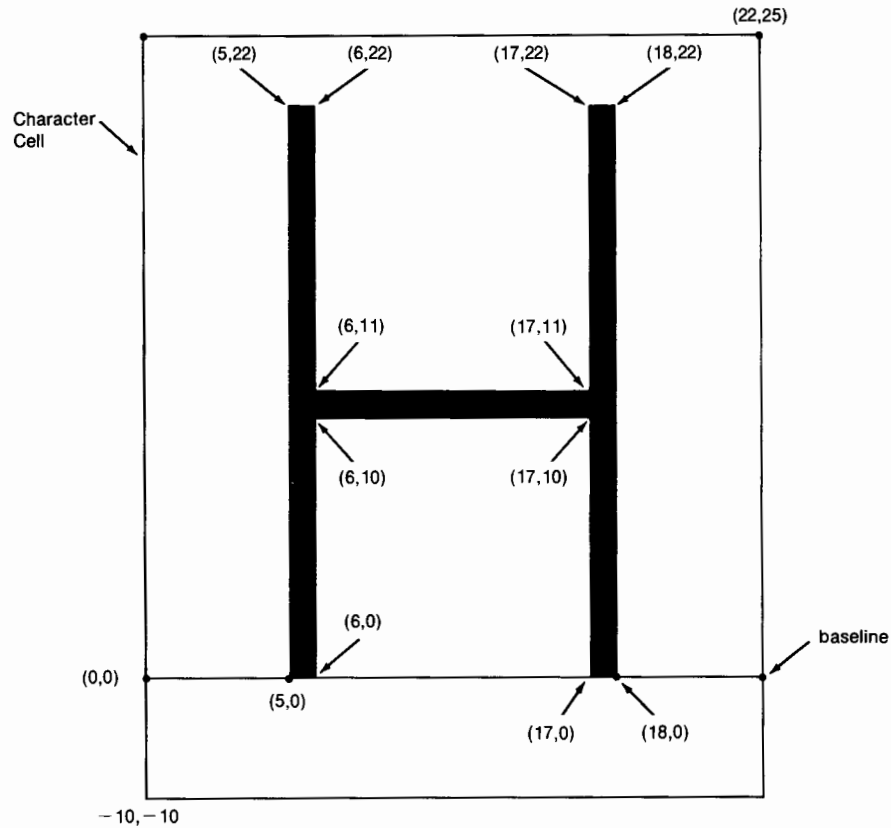


Figure 4-20. The Character 'H'

The definition is entered using one *continuous* escape sequence:

```

t*n 72 8a -10 -10 22 25b 2c 5 0d 5 22d 6 22d 6 11d 17
11d 17 22d 18 22d 18 0d 17 0d 17 10d 6 10d 6 0d 0c
    
```

Considerations for Character Definitions

DATA POINTS OUTSIDE THE CHARACTER CELL. Character coordinates may lie outside the character cell, but if the entire character cell is outside the virtual window, no portion of the character will be displayed.

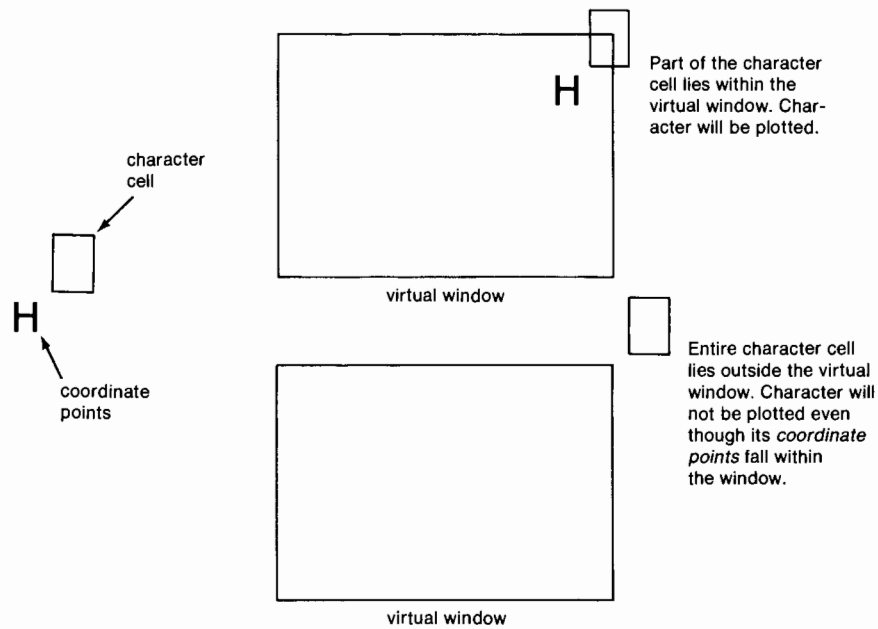


Figure 4-21. Displaying Character Definition Data Which is Outside the Character Cell

TEXT ORIGIN CONSIDERATIONS. If you are defining a font to use as graphics text, you should define your characters such that their baseline is the $y=0$ horizontal axis, (note in figure 4-20). The terminal determines the text origin (in the t^*mq command) based upon the $y=0$ axis.

MEMORY CONSIDERATIONS. The terminal can store up to 14 user defined fonts, depending upon the amount of available memory. The system font (stick characters) occupies 2-3K bytes of memory. The bold font of the *Graphics Editor/2700* and *Autoplot/2700* application software occupies 4-5K bytes.

How to Use Characters and Fonts

There are two ways to use characters:

- as plotting symbols — see Section 2. DRAWING FUNDAMENTALS, "Symbols".
- as graphics text — see Section 2. DRAWING FUNDAMENTALS, "Graphics Text".

To select a user defined font for labels or text, issue the command:

$\text{t}^*n\langle\text{font}\rangle\text{f}$

NOTE: The font must exist.

Creating a Picture

This font will be used whenever graphics text mode (`⋄`) or text label (`⋄`) is implemented. User defined fonts are subject to the same graphics text attributes as the system font.

Example. Display the “H” you created in the previous example.

Step 1. Select font 8.

```
⋄*n8F
```

Step 2. Position the pen.

```
⋄*pa 100 100 Z
```

Step 3. Write the label.

```
⋄*lHHHHHHHH CR
```

Defining Fonts for Extended Roman Characters

User defined fonts are logically paired (1,2 3,4) to allow for the definition of extended roman characters. Therefore, if font 3 is the active font and an extended roman character is typed, the corresponding character in the high font will be used according to the table below. Note that if the high font was activated by the `⋄*nF` command, typing a standard character will not cause the low font to be used.

The pairing of fonts is optional. Users who do not need to define extended roman characters may make full use of the user defined fonts.

Table 4-4. Extended Roman Character Set Codes

	30	40	50	60	70	80	90	100	110	120
0		´			ó	À	Ö			
1		˘	•	§	ú	í	Ü			
2		ˆ			à	Ø	É			
3		¨	ç		è	Æ	ï			
4		˜	Ñ	â	ò	á	ß			
5			ñ	ê	ù	í				
6			ı	ó	ä	ø				
7		£	ç	û	ë	æ				
8		˘	Œ	á	ö	Ä				
9		˘	£	é	ù	ì				

User Defined Font Library Commands

These commands are used to maintain font libraries. (See Section 5. PICTURE FILE COMMANDS.)

- `␣*nd` — Delete font.
- `␣*ne` — Delete all user defined fonts.



Picture File Commands _____ 5

INTRODUCTION

This section discusses the commands which let you examine and manipulate the contents of the picture file. Topics include:

- Transparent Mode
- Object Visibility Fence
- Library Commands
- Picture File Protection
- Copying and Examining the Picture File
- Picture File Inquiry Commands

Table 5-1. Picture File Commands

<code>⌘*hd</code>	Delete vector list
<code>⌘*he</code>	Delete vector list and associated objects
<code>⌘*hl</code>	Delete vector list library
<code>⌘*hn</code>	Rename vector list
<code>⌘*ho</code>	Replace vector list of an object
<code>⌘*gd</code>	Delete active object attribute list
<code>⌘*gl</code>	Delete all object attributes lists
<code>⌘*gn</code>	Rename object attribute list
<code>⌘*go</code>	Replace vector list reference
<code>⌘*ed</code>	Delete view
<code>⌘*el</code>	Delete all inactive views
<code>⌘*nd</code>	Delete user defined font
<code>⌘*ne</code>	Delete all user defined fonts

Table 5-1. Picture File Commands (Cont.)

E^*w1	Inquire picture file
E^*wr	Graphics hard reset
E^*wt	Set transparent mode
E^*wv	Set object visibility fence
E^*ww	Set picture protection
E^*wy	Set graphics save mode
$\text{E}^*w^$	Inquire picture file

TRANSPARENT MODE

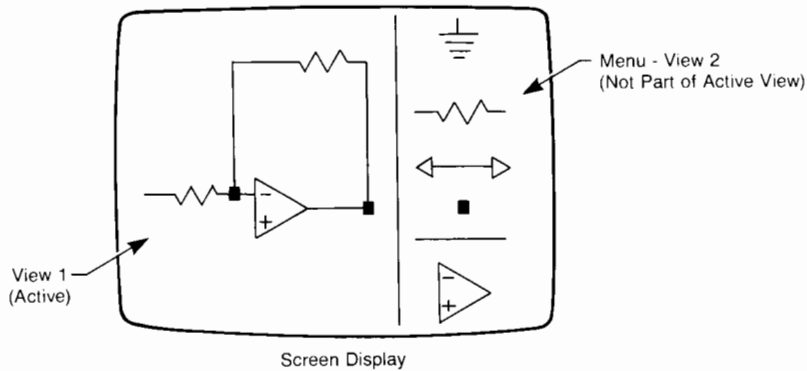
At power on, the terminal has an open displayed vector list ready to accept primitives. When the terminal is in "transparent mode", however, primitives which are not preceded by an E^*ha command are displayed on the screen, but are not saved in a displayed vector list. Primitives entered while the terminal is in transparent mode will be lost if the view is redrawn, zoomed, or panned.

Set Transparent Mode: E^*wt

Exit Transparent Mode: E^*w0t

Any primitives which are entered after transparent mode is exited may be saved in a vector list.

Figure 5-1 demonstrates the usefulness of transparent mode.



1. Vectors for menu are entered when terminal is set for transparent mode. (Vector memory is saved) E^*w1T
2. The redraw mode is set for multiple views. E^*e1M
3. View 2 is activated.
4. Transparent mode is exited. E^*w0T

Figure 5-1. Using Transparent Mode

OBJECT VISIBILITY FENCE

You can set a “visibility fence” such that objects with a visibility lower than or equal to the fence are not displayed on the screen.

Set Visibility Fence: `⌘*w<fence>v`

where `<fence>` is an integer in the range `(-32768, . . . 32767)`

LIBRARY COMMANDS

The terminal can maintain “libraries” of the five picture file elements; vector lists, object attributes, views, user defined fonts, and palettes. The following paragraphs describe commands which can be used to maintain such libraries.

NOTE: Many of these commands will cause the picture to be redrawn if auto-redraw mode is enabled. In general, any command which might change the appearance of the picture will cause a redraw.

Vector List Library Commands

Delete Vector List: `⌘*h<<vector list name>>d`

This command deletes the vector list specified by `<vector list name>` from the vector list library. Any set of object attributes which contained a pointer to this vector list will now have a nil pointer. If auto redraw is on, the picture is redrawn.

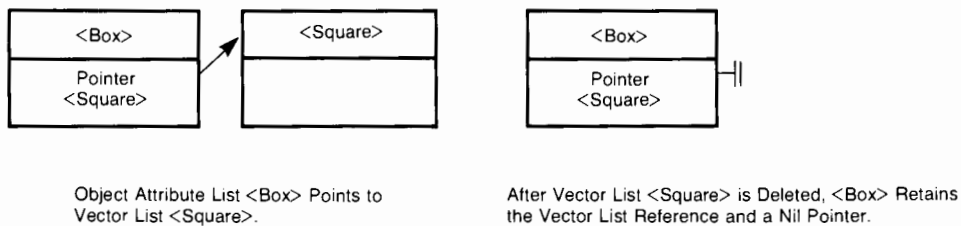


Figure 5-2. Delete Vector List (`⌘*hd`)

Delete Vector List and Associated Objects: `⌘*h<<vector list name>>e`

This command deletes the vector list specified by `<vector list name>`, and also deletes any object attribute lists which were assigned to this vector list. **NOTE:** Object attribute lists will be deleted even if the vector list is non-existent. If auto-redraw is enabled, the picture is redrawn after the execution of this command.

Picture File Commands

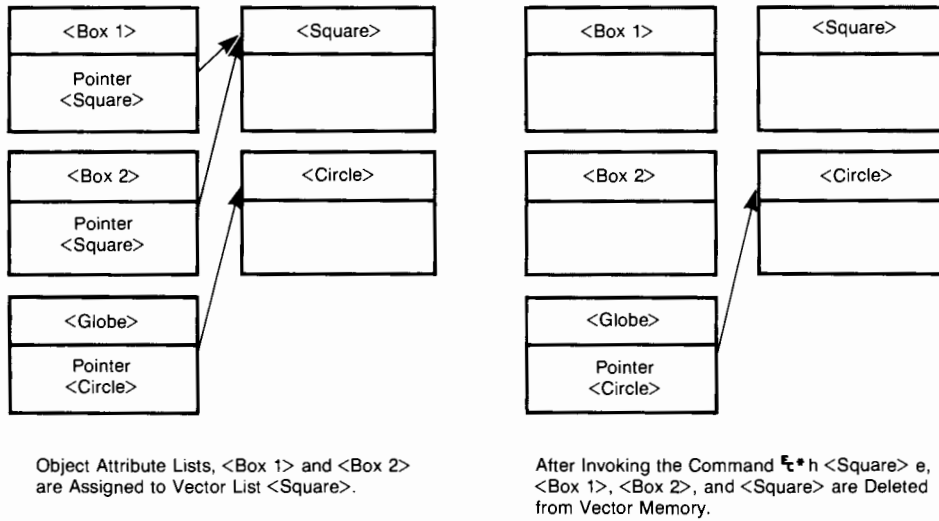


Figure 5-3. Delete Vector List and Associated Objects ($\epsilon * h e$)

Delete Vector List Library: $\epsilon * h 1$

All vector lists (with the exception of the current open vector list) are deleted. All object attribute sets which were assigned to these vector lists now have nil pointers. If auto redraw is on, the picture is redrawn.

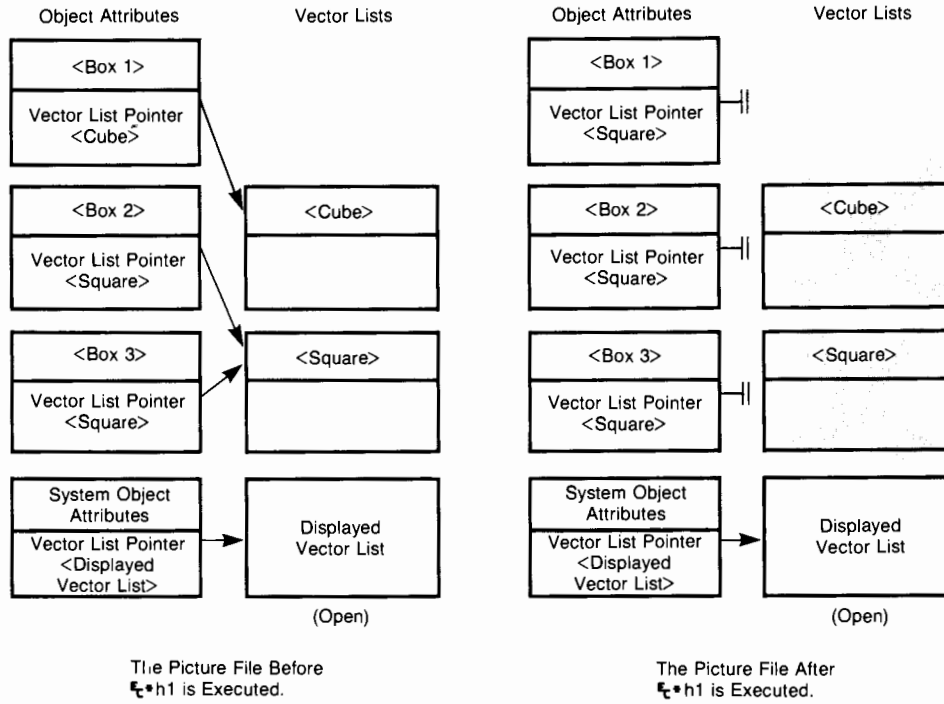
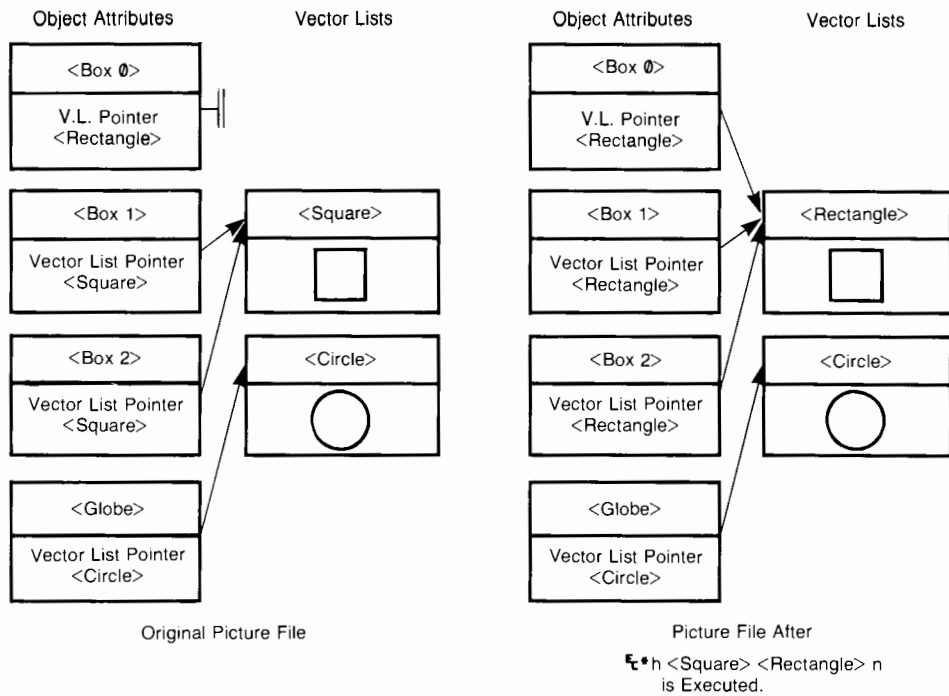


Figure 5-4. Delete Vector List Library (`!h1`)

Rename Vector List: `!h<<new vector list name>><<old vector list name>>n`

This command replaces or renames a vector list with `<new vector list name>`. All object attributes sets with pointers to `<old vector list name>` now have pointers to `<new vector list name>`. If auto redraw is enabled, the picture is redrawn.

Picture File Commands

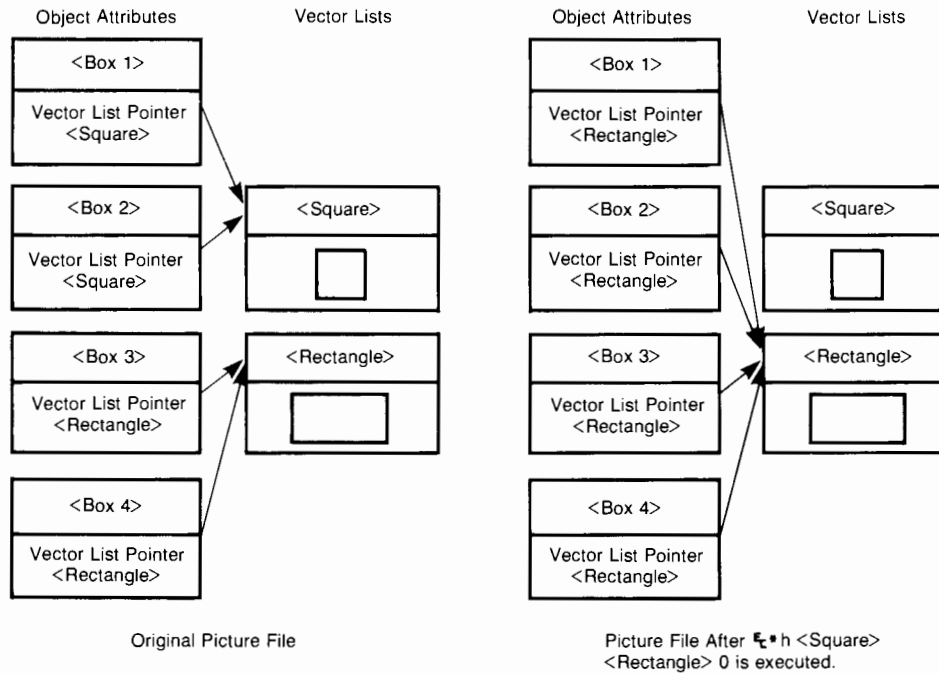


Note: Vector List <Square> = Vector List <Rectangle>

Figure 5-5. Rename Vector-List (⌘*hn)

Replace Vector List of an Object: ⌘*h<<new vector list name>><<old vector list name>>o

All objects created with the vector list specified by <old vector list name> are recreated with the vector list specified by <new vector list name>. If auto-redraw is on, the picture is redrawn.

Figure 5-6. Replace Vector List of an Object (F^*h)

Object Library Commands

Delete Object Attributes of Active Object: F^*gd

NOTE: You cannot delete the system set of object attributes. If auto-redraw is on, the picture is redrawn.

Delete All Sets of Object Attributes: F^*g1

NOTE: You cannot delete the system set of object attributes. If auto-redraw is on, the picture is redrawn at the end of the sequence. (The screen is cleared if there are no primitives in the displayed vector list, if the redraw mode is set correctly to clear the screen.)

Rename Object Attributes: $\text{F}^*g\langle\langle\text{new object attribute name}\rangle\rangle n$

The active set of object attributes is renamed. All subsequent F^*g commands affect the same set of object attributes (with its new name).

NOTE: This command does not affect the system object.

Example: Rename object attribute list "square" to "rectangle".

`ƒ*g<square>a<rectangle>n`

Replace Vector List Reference: `ƒ*g<<vector list name>>o`

The active set of object attributes are applied to a new vector list. If auto redraw is on, the picture is redrawn.

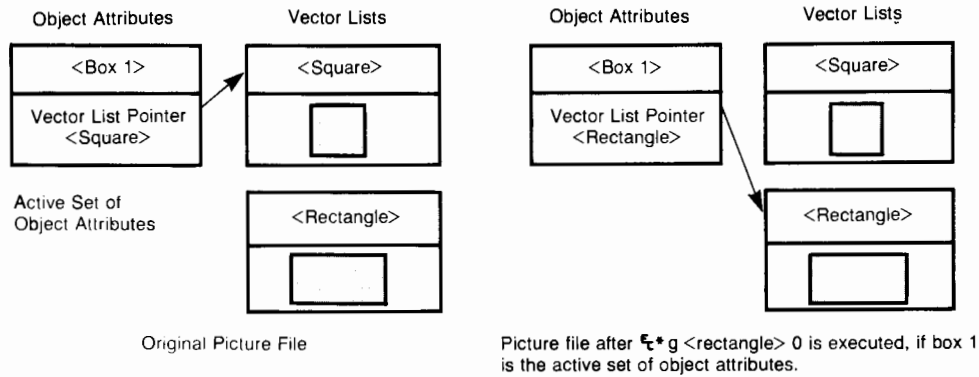


Figure 5-7. Replace Vector List Reference (`ƒ*g o`)

View Library Commands

Delete View: `ƒ*e<view number>d`

where <view number> is an integer in the range (0, . . . , 255)

NOTE: The view specified may not be the active view.

Delete All Inactive Views: `ƒ*e1`

User Defined Character And Font Library Commands

Delete Font: `ƒ*nd`

where is an integer in the range (2, . . . , 16)

If references an undefined font, Font 1 (the system font) will be used, and the "undefined font" error is set.

Delete All User Defined Fonts: `ƒ*ne`

Color Palette Library Commands

Delete Palette: `␣*v<palette number>d`

where <palette number> is an integer in the range (0, . . . ,255)

NOTE: <palette number> can not be '0' (the system palette) or the active palette.

Delete All Palettes: `␣*ve`

Deletes all palettes except the system palette '0', and resets the active palette to the system palette.

PROTECTING THE PICTURE FILE

Setting picture protect mode ensures that elements in the picture file will not be overwritten by new like named elements read into memory from the keyboard, datacomm, file system, et cetera. If a picture is protected, naming conflicts are resolved in favor of the resident element, and the duplicate element is lost.

NOTE: This command does not apply to views and palettes.

Set Picture File Protection: `␣*w<mode>w`

where <mode> = 0 (unprotected)
1 (protected)

COPYING/EXAMINING THE PICTURE FILE

You can copy the contents of the picture file to disc, printer, datacomm, or screen using the COPY FILE :PICTURE command. (See the Section 7. DEVICE CONTROL of the *HP 2700 Alphanumeric Reference Manual* for an explanation of the COPY command.)

Choosing Selected Elements

The following command selects the elements to be copied to the destination devices when the COPY FILE :PICTURE command is executed:

`␣*w<save mask>y`

where <save mask> is an integer in the range (-32768, . . . , 32767)

Picture File Commands

Bits set to '1' in the save mask correspond to elements of the picture file as follows:

<i>bit</i>	<i>meaning</i>
0	All Color Palettes
1	Active Color Palette
2	System Text Fonts
3	User-defined Text Fonts
4	Current Text Font
5	All Views
6	Active View
7	All Named Vector Lists
8	All Named Objects
9	Active Object/Vector List
10	System Object/Displayed Vector List

NOTES:

- In order to examine the picture file in escape sequence form on the display, enable Display Functions Mode before executing the COPY FILE :PICTURE command.
- The default picture save mode is "1474" or (10111000010_B).

How To Interpret The Picture File

Example: Create the following picture in the terminal:

```
Open Hidden Vector List          ㄿ*hA
Define Primary Pen                ㄿ*m4X

Plot Polygon Area Fill           ㄿ*psa 100,100 300,150, 350,200
                                   200,300 50, 150T
Reset Pen Position/Add Label     ㄿ*pa 200,200Z
                                   ㄿ*1POLYGON%
Close Vector List                 ㄿ*h<VL1>C

Define Object                     ㄿ*g<OBJ1>a<VL1>C
Define Attributes                 ㄿ*g 200,200x1,1r2,2s25,25T

Define View/Set Background Pen    ㄿ*e12i0,0 600,600w10,10
                                   500,500v3B

Select Default Elements to be Copied
to the Screen                     ㄿ*w1474Y

Copy Picture to Screen            COPY ALL :PICTURE TO :DISPLAY
```

With Display Functions mode enabled, you will see the following output:

```

ε*ν0m0p0i1a1i0a1b2i1a3i0a0b1c4i1a5i0a1b6i1a7i0.4a0.1333b0c8i0.9333a0.3333b9i0.6a
ε
ι1b0.0666c10i0.9333a0.7333b0c11i0.3333a0.3333b1c12i0.6a0.1333b13i0a1b0.5334c14iε
ι10.8a0.8b0.8c15iLε*e0a12iq3b10 10 500 500v0 0 600 600Wε*w0Tε*hAε*m15x0y0i2a1b2gε
ι255 1ch255 255 255 255 255 255 255 255d1s1t0Uε*n42 1Uε*pa0 0Zε*hBε*m4Xε*psε
aι100 100 300 150 350 200 200 300 50 150Tε*m1q1 1Mε*n1 0Rε*m0Dε*n0w1f7xε
ιZε*1POLYGONε
ε*h<VL1>Cε*g<OBJ1>a<VL1>cb25 25t2.8286 2.8286r2 2s200 200xabε*m4x0yε
ι0i2a1b2g255 1ch255 255 255 255 255 255 255 255d1s1t0Uε*n42 1Uε*pa0 0Zε*hbε
ιZε*e1r1Aε
ι

```

NOTES:

- ε*ν0m...L Set pen values and load palette 0.
- ε*e0a...600W View specifications.
- ε*w0T Turn off transparent mode.
- ε*ha...1U Vector list header.
- ε*pa...POLYGONε Vector list primitives.
- ε*h<VL1>C Close vector list.
- ε*g<OBJ1>a. . .200x Object attributes.
- . . .ab Reactivate system object attributes and set default values.
- ε*m4X. . .1U Vector list header of new displayed vector list.
- ε*pa. . .Z Initialize pen position of displayed vector list.
- ε*e1r1A Turn on auto redraw mode; redraw view.
- Picture file records are delineated with ε ι if COPY is used, (not if TRANSFER is used).
- Coordinate pairs are always separated by a space or letter — not an end of record character.

Reading The Picture File From A Host Computer

Appendix A. APPLICATION NOTES contains a BASIC program which reads the picture file, and stores the data in a datacomm file.

PICTURE FILE DEFAULTS

The following command performs a hard graphics reset: `␣*wr` This escape sequence sets the graphics defaults as follows:

FUNCTION	VALUE	COMMAND
Raster memory	clear	da,db
Graphics display	on	dc,dd
Alpha display	on	de,df
Zoom factor	1,1	di
Zoom center	255,185	dj
Graphics cursor	off, short	dk,dl
Rubberband line	off	dm,dn
Graphics cursor address		
virtual	0,0	do,dp
display	0,0	
definition	virtual	du
Alpha cursor	on,1	dq,dr
Graphics text mode	off	ds,dt
Display visibility	on(15)	dv
System write mask	on(15)	dw
Active view	0(default)	ei
viewport limits	0,0,511,389	ev
window limits	0,0,511,389	ew
highlighting	off	eh
Viewport bckgrnd color	0	eb
Redraw mask	2	em
Auto-redraw	on	ea
Active object	system	ga
System object attributes		
visibility	1	gv,gb
highlighting	off	gh,gb
detectability	1	gp,gb
translation	0,0	gt,gb
transform center	0,0	gx
rotation	0	gr,gb
scale	1,1	gs,gb
object write mask	15	gw,gb
Drawing Mode	2(Jam1)	ma,mr
Line type	1(solid)	mb,mr
User line pattern	255,1(solid)	mc
User fill pattern	255,255,255,255	
	255,255,255,255	md
Area type	2(user)	mg
Area highlighting	off	mh
Pick ID	0	mi
Relocatable origin	0,0	mj,mk,ml,mr
Text size	1	mm,mr
	7,10	ns

FUNCTION	VALUE	COMMAND
Text angle	1	mn,mr
	0	nr
Text slant	0	mo,mp,mr
Text justification	1	mq,mr
Symbol Size	1.0000	ms
Symbol Mode	1(off)	mt
Symbol Units	0(display)	mu
Primary pen	15	mx
Secondary pen	0	my
Text font	1(standard)	nf
User symbol	' * '	nu
Text spacing	0	nw
Pen condition	down	pa,pb,mr
Pen position	0,0	pc
Plotting command data type	ASCII absolute	pf,pg,ph,pi,pj,pk pl,pm
Raster dump configuration		
graphics video	on	rd,ri
horizontal window address	0	rm,ri
vertical window address	0	rn,ri
plane window address	0	ro,ri
horizontal window dim.	9999	rp,ri
vertical window dim.	9999	rq,ri
plane window dim.	9999	rr,ri
horizontal size	9999	rs,ri
vertical size	9999	rt,ri
planes	9999	ru,ri
x offset	0	rx,ri
y offset	0	ry,ri
z offset	0	rz,ri
Raster hardcopy formats		
black and white halftone mode	off	er
halftone index	0	ui
auto-halftone-tracking	2(high contrast)	ut
pattern map	as per auto values	up
Initial graphics color palette	system	vl
current palette	0(default)	vp
current pen	15	vi
color method	0(RGB)	vm
Object vis. fence	0	wv
Picture protection	1(enabled)	ww
Graphics save mode	1474	wy

PICTURE FILE INQUIRY COMMANDS

The following commands allow you to request status information about the picture file. They are explained in detail in Section 8. INQUIRY COMMANDS AND STATUS REQUESTS.

`^c#wi, ^c#w^`

Raster Manipulations _____ 6

INTRODUCTION

This section will show how you can directly manipulate pixel values in raster memory. Before you continue reading this section, you should review the discussion entitled "Raster Color Graphics" in Section 1. GRAPHICS CONCEPTS.

Raster Memory

A raster buffer stores the image on the screen as a binary array. Each pixel (picture element) is represented by a 4 bit binary number. Therefore, each pixel may have a value of (0-15). The value of a pixel is used to address the look-up table in the color mapper, which associates the value of the pixel with a color from the active palette. The terminal has a primary raster and an optional auxiliary raster. The primary and auxiliary rasters hold 512×512 pixels (although the screen can display only 512×390 pixels at one time).

Since each pixel is 4 bits, you can imagine each raster buffer as 4 planes of 512×512 bits. This model (shown in figure 6-1) will be used to illustrate many of the concepts discussed in this section.

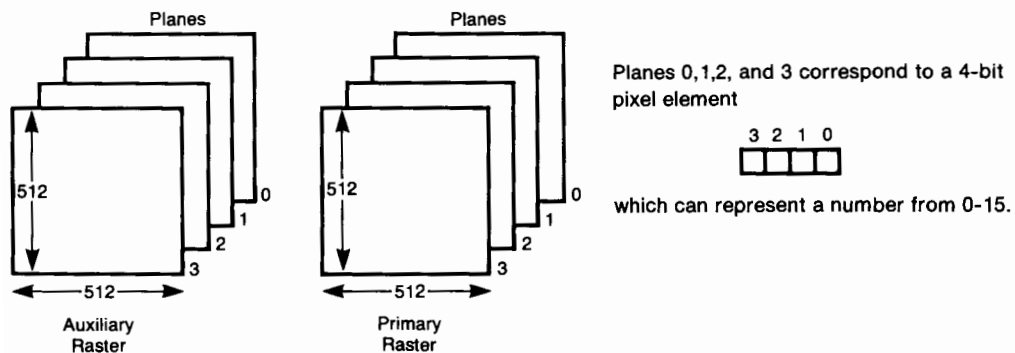


Figure 6-1. Raster Memory

Raster Memory Operations

The following raster operations will be explained in this section:

- Setting all the raster pixels to one value in effect, setting them to a color of the active palette
- Directing the flow of graphic data from the picture file to the raster by specifying:
 - the raster(s) (primary or auxiliary) which may receive the data
 - the individual raster planes which may receive the data
 - the raster planes which may store a particular object
 - the vector drawing mode
- Controlling the flow of raster data to the screen (or other output device) by setting a read mask
- Raster data transfers
 - from raster memory to printer, disc, or datacomm (raster dumps)
 - from disc or datacomm to a selected portion of the raster (raster loads)
 - from raster memory to the host (via selective reads by the host)

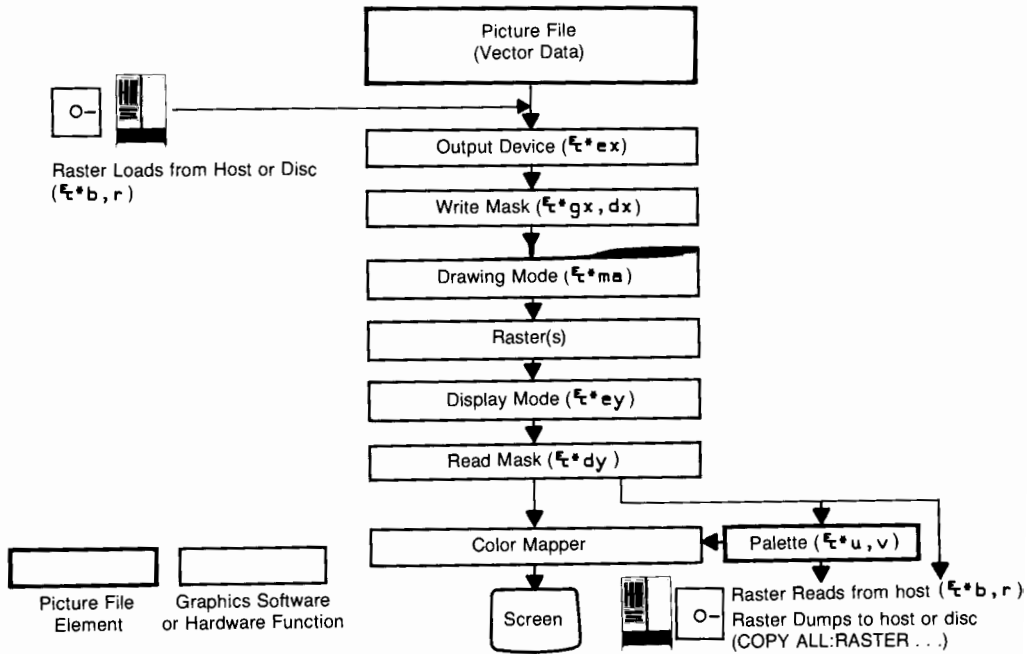


Figure 6-2. Raster Operations

Table 6-1. Raster Manipulation Escape Sequences

<i>Display Commands</i>		<code>␣*ra</code>	Prepare for raster dump
		<code>␣*rb</code>	End transfer
<code>␣*da,db</code>	Set raster buffer to pixel value	<code>␣*rc</code>	Erase screen
<code>␣*dv</code>	Set visibility mask	<code>␣*rd</code>	Turn on graphics video
<code>␣*dw</code>	Set display write mask	<code>␣*re</code>	Read raster status
<code>␣*gw</code>	Set object write mask	<code>␣*rf</code>	Set transfer format
<code>␣*ma</code>	Set drawing mode	<code>␣*ri</code>	Set all parameters to default values
<code>␣*ex</code>	Select output devices	<code>␣*rj</code>	Return raster size status
<code>␣*ey</code>	Set display mode	<code>␣*rk</code>	Return model number
<i>Raster Transfer Commands</i>		<code>␣*rm</code>	Set horizontal window address
		<code>␣*rn</code>	Set vertical window address
<code>␣*bp</code>	Read binary block into datacomm	<code>␣*ro</code>	Set level window address
<code>␣*bq</code>	Read binary block into datacomm; set up next plane	<code>␣*rp</code>	Set horizontal window dimension
<code>␣*br</code>	Read binary block into datacomm; set up next line	<code>␣*rq</code>	Set vertical window dimension
<code>␣*bu</code>	Write into raster memory	<code>␣*rr</code>	Set window plane dimension
<code>␣*bv</code>	Write into raster memory; set up next plane	<code>␣*rs</code>	Specify source x size
<code>␣*bw</code>	Write into raster memory; set up next line	<code>␣*rt</code>	Specify source y size
<code>␣*bx</code>	Set temporary X offset	<code>␣*ru</code>	Specify source z size
<code>␣*by</code>	Set temporary Y offset	<code>␣*rx</code>	Set X origin
<code>␣*bz</code>	Set temporary Z offset	<code>␣*ry</code>	Set Y origin
		<code>␣*rz</code>	Set Z origin

SETTING RASTER MEMORY

You can set all the pixels in raster memory to a particular value (corresponding to a color of the active palette) using either of two escape sequences:

`␣*d<pen number>a`

`␣*d<pen number>b`

where <pen number> is an integer in the range (0, . . . , +32767)

The low four bits of the <pen number> determine the pixel value. If a bit=1, all the bits in the associated raster plane are turned on. If a bit=0, all the bits in the associated raster plane are cleared.

Both commands perform the same function; only their defaults differ. The default for the `␣*da` command is pen 0 (normally black), and the default for the `␣*db` command is pen 15

Raster Manipulations

(normally white). These default values will perform a clear or set function, and make the terminal compatible with other Hewlett-Packard black and white graphics terminals.

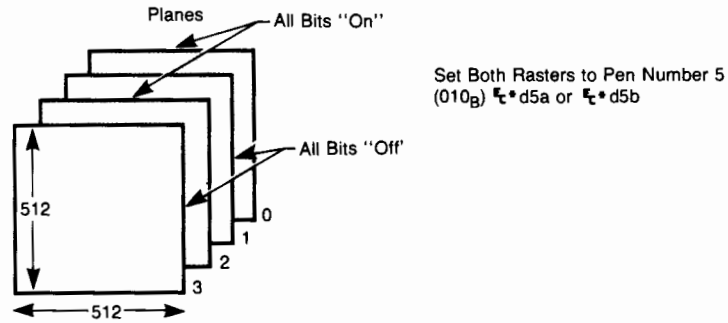


Figure 6-3. Setting the Raster Memory

NOTES:

- Performing these functions will erase any open vector list.
- Both of these commands are affected by the system write mask, described below.

WRITING TO RASTER

You can direct the flow of graphic information to raster memory by routing the data from the picture file to one or both rasters. The displayed result will depend on the display and object write masks and vector drawing mode in use.

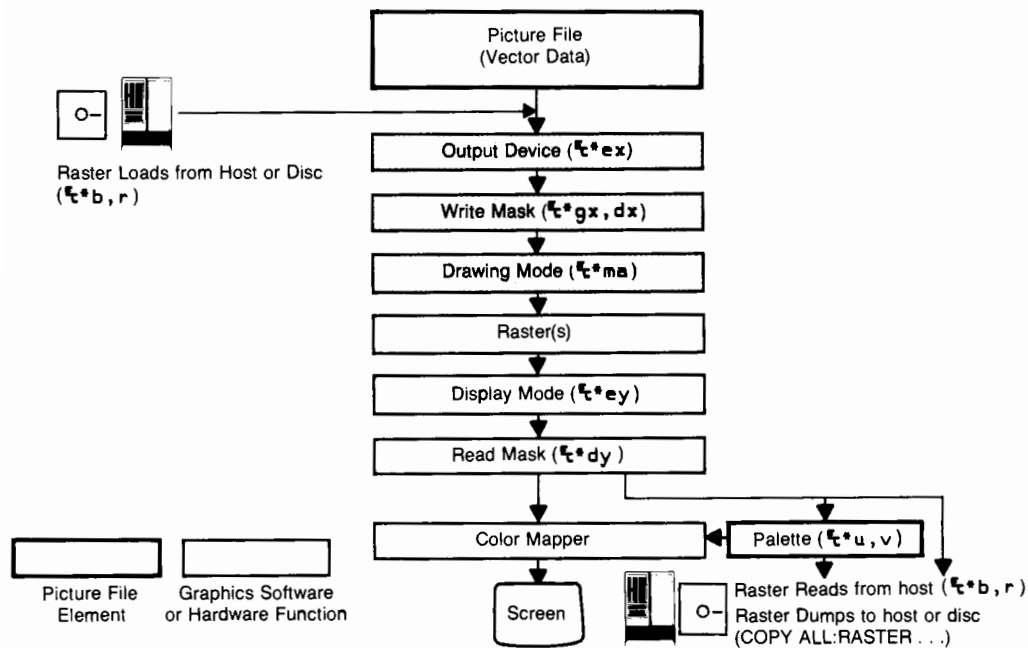


Figure 6-4. Writing to Raster

Specifying Output Devices

The terminal's output devices are the primary and auxiliary raster buffers. You can direct the flow of graphic information from the picture file to either device (or to both) using the command:

$$t*e<mask>X$$

where $\langle\text{mask}\rangle$ is an integer in the range $(-32768, \dots, +32767)$

The low two bits determine the output device according to the following criteria:

- If bit 0=1, the primary raster is enabled.
- If bit 1=1, the auxiliary raster is enabled.
- If bit 0 and bit 1=1, both rasters are enabled; i.e., any redraw will affect both rasters.
- If bit 0 and bit 1=0, the terminal is set for 8-bit mask mode. (See "System Write Mask" later in this section.)

Raster Manipulations

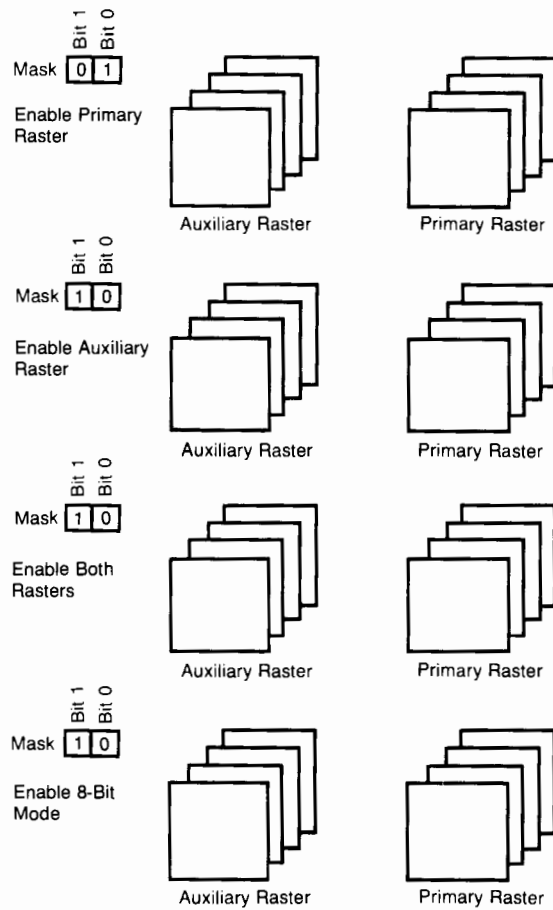


Figure 6-5. Specifying Output Devices ($\text{t} \cdot \text{e} \langle \text{mask} \rangle \text{x}$)

NOTES:

- 8-bit mask mode has meaning only in conjunction with the system write mask, explained later in this section.
- When double buffer mode is enabled, most $\text{t} \cdot \text{ex}$ and $\text{t} \cdot \text{ey}$ operations are illegal, and will have no effect.

System Write Mask

The system write mask determines which raster planes are available to store vector information. The system write mask is set by supplying a mask in the command:

$\text{r}*\text{d}\langle\text{mask}\rangle\text{w}$

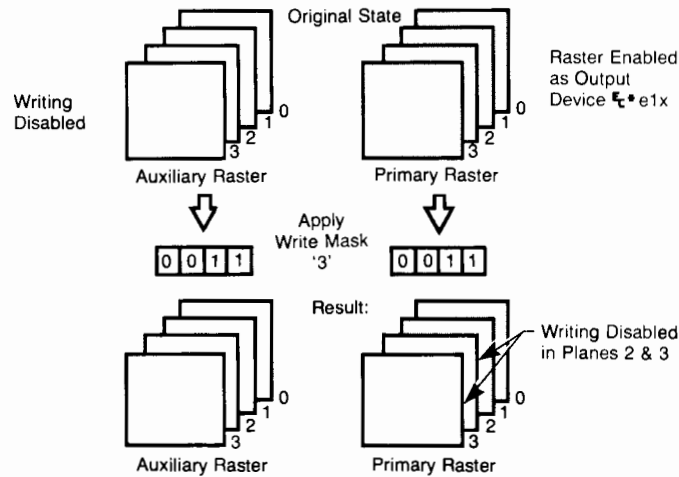
where $\langle\text{mask}\rangle$ is an integer in the range $(-32768, \dots, +32767)$

Unless 8-bit mask mode is specified, only the low four bits of this number are used to determine the mask. If both the auxiliary and primary rasters are enabled, the mask is the same for each raster. If 8-bit mask mode is specified, the mask for the rasters can be unique. The mask is ANDED with the logical output device(s) set in the $\text{r}*\text{e}1\text{x}$ command.

Example 1. If the output device is the primary raster ($\text{r}*\text{e}1\text{x}$), and the system write mask is '3', then:

output device	Aux.	Prim.	
	- - - -	X X X X	
write mask	0 0 1 1	0 0 1 1	
	- - - -	- - X X	
	X = writing enabled - = writing disabled		

Planes 0 and 1 of the primary raster are available to store graphic information.



To Observe this Result on Your Terminal; Follow These Steps:

1. Hard Reset $\text{r}*\text{wr}$
2. $\text{r}*\text{e}1\text{x}$ - Enable Primary Raster
3. $\text{r}*\text{d}b$ - Set Raster to Pen Number 15
4. Press G
CLEAR
5. $\text{r}*\text{d}3\text{w}$ - Set Write Mask
6. $\text{r}*\text{d}b$ (The Raster Should be Set to Pen Number 3)

Figure 6-6. Example 1

Raster Manipulations

Example 2. If both the primary and auxiliary rasters have been specified as output devices (ε*ε3x), and the system write mask is '5', then:

	Aux.	Prim.
output device	x x x x	x x x x
write mask	0 1 0 1	0 1 0 1

	- x - x	- x - x
	x = writing enabled	
	- = writing disabled	

Planes 0 and 2 of the primary raster and planes 0 and 2 of the auxiliary raster are available for storing vector information.

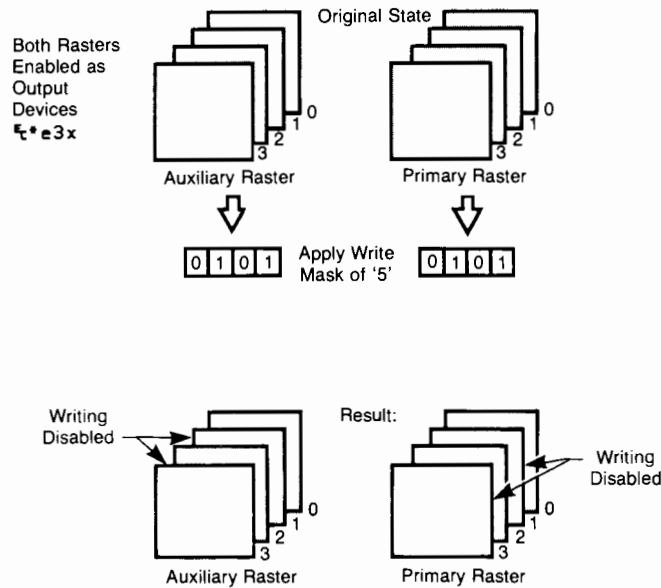


Figure 6-7. Example 2

Example 3. If 8-bit mask mode ($\text{r}*\text{e}0\text{x}$) is specified, the low eight bits of the mask in the $\text{r}*\text{d}\langle\text{mask}\rangle\text{w}$ command are used. Therefore, if the write mask is 128, then:

	Aux.	Prim.
output device	X X X X	X X X X
write mask	1 0 0 0	0 0 0 0
	X - - -	- - - -
	X = writing enabled	- = writing disabled

Only plane 3 of the auxiliary raster is available for storing graphic information.

NOTE: If the write mask were set to '255', all 8 raster planes would be enabled. The actual pen colors would depend upon the display mode, however, which only allows the simultaneous display of 16 colors. (See "Display Modes" in this section.)

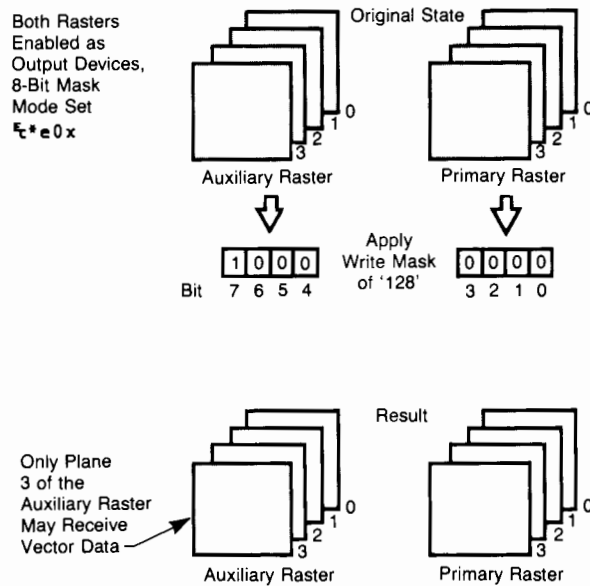


Figure 6-8. Example 3

Object Write Mask

You can also set write masks which apply only to individual objects. An object write mask is an object attribute, and is set using the command.

```
⌘*g<mask>W
```

where <mask> is an integer in the range (-32768, . . . , +32767); the low four bits (or low 8 bits in 8 bit mode) are used to determine the mask

NOTE: As with all object attribute commands, this command only affects the active object.

The object write mask is ANDED with (the logical output device AND system write mask).

Example 4. Specify both rasters as output devices, a system write mask of '10', and an object write mask for <apple> of '7'.

Step 1. Specify output devices.

```
⌘*e3X
```

Step 2. Set the system write mask.

```
⌘*d10W (1 0 1 0 B)
```

Step 3. Reactivate the object attributes for <apple>, and set the object write mask.

```
⌘*g<apple>a7W (0 1 1 1 B)
```

	Aux.	Prim.
output device	X X X X	X X X X
write mask	1 0 1 0	1 0 1 0
	<hr/>	
	X - X -	X - X -
object write mask	0 1 1 1	0 1 1 1
	<hr/>	
	- - X -	- - X -
	X = writing enabled	
	- = writing disabled	

Plane 2 of the primary raster and plane 2 of the auxiliary raster are available for storing <apple> vectors.

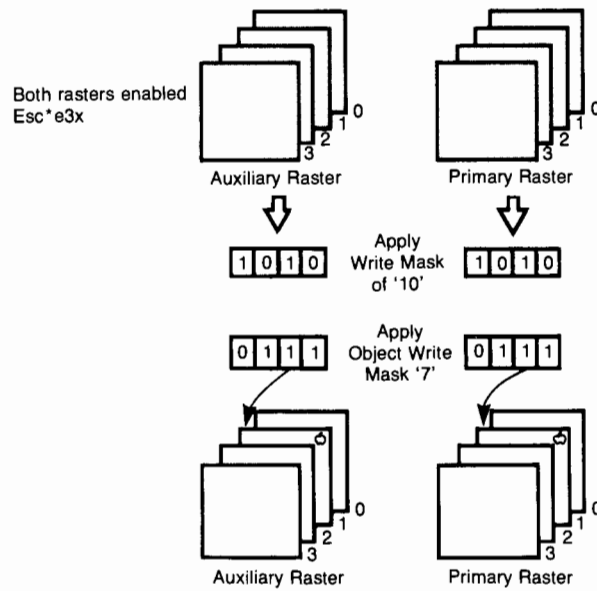


Figure 6-9. Object Write Masks (Example 4)

Vector Drawing Modes

The terminal has seven drawing modes which affect the way in which pixels are set when vectors are drawn in the raster.

The command to select a drawing mode is:

`Esc * m <mode> a`

where <mode> is an integer in the range (0, . . . , 7)

<mode>	meaning
0	Non-operative (NOP)
1	Clear1
2	Jam1 (Default)
3	Comp1
4	Jam2
5	OR
6	Comp2
7	Clear2

Raster Manipulations

The effects of the various drawing modes will be demonstrated using the following conditions:

- A portion of a raster line exists with pixel values:

3 3 3 3 0 0 0 0

- A dotted line will be drawn with line pattern 7 ($\tau * m7B$):

x - x - x - x -

- Primary pen = 0 0 0 1_B = 1 ($\tau * mx$)
- Secondary pen = 0 0 1 0_B = 2 ($\tau * my$)
- Background pen = 0 1 1 0_B = 6 ($\tau * eb$)

Mode 0. NOP (Picture Protect no change to raster)

Mode 1. CLEAR1 — If the overlaying bit= - No change
If the overlaying bit= X Pixel is drawn in the current viewport-
background color

CLEAR1 mode is used for object undraw commands: $\tau * gu$

3 3 3 3 0 0 0 0	Pixel value of raster
x - x - x - x -	Pattern drawn in CLEAR1 mode
<hr/>	
6 3 6 3 6 0 6 0	New pixel values

Mode 2. JAM1 — If the overlaying bit= - No change
If the overlaying bit= x Pixel=Primary pen

JAM1 is the default drawing mode.

3 3 3 3 0 0 0 0	Pixel value of raster
x - x - x - x -	Pattern drawn in JAM1 mode
<hr/>	
1 3 1 3 1 0 1 0	New pixel values

Mode 3. COMP1 — If the overlaying bit= - No change
If the overlaying bit= x NOT pixel

COMP1 performs a contrast function, however, the vectors are not guaranteed to contrast with the background color. (See Mode 6 COMP2.) If you redraw a line a second time in COMP1 mode, the original pixel values will be restored.

Raster Manipulations

When pixel value 0 = 0 0 0 0, is OR'ed with the
 primary pen = 1 = 0 0 0 1, the result is

$$\begin{array}{r} 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 1 \\ \hline 0\ 0\ 0\ 1 = 1. \end{array}$$

3 3 3 3 0 0 0 0	Pixel value of raster
x - x - x - x -	Pattern drawn in OR mode
<hr/>	
3 3 3 3 1 0 1 0	New pixel values

Mode 6. COMP2 — If the overlaying bit= - No change
 If the overlaying bit= x Pixel is exclusively OR'ed (XOR'ed) with
 (primary pen XOR'ed with background pen)

COMP2 mode performs a contrast function in which the new line is guaranteed not to be the background color.

When the primary pen 1 = 0 0 0 1 is XOR'ed with the
 background pen 6 = 0 1 1 0, the result is

$$\begin{array}{r} 0\ 1\ 1\ 1 = 7 \text{ which is XOR'ed with} \\ 0\ 0\ 1\ 1, \text{ and the result is} \\ \hline 0\ 1\ 0\ 0 = 4. \end{array}$$

When '7' is XOR'ed with the pixel value '0', the result is '7'.

NOTE: Exclusive OR can be thought of as binary addition without carry.

Therefore,

3 3 3 3 0 0 0 0	Pixel value of raster
x - x - x - x -	Pattern drawn in COMP2 mode
<hr/>	
4 3 4 3 7 0 7 0	New Pixel Values

Redrawing the line a second time in COMP2 mode results in the restoration of the original raster pixel values.

(From above . . .)

Primary Pen XOR'ed with Background Pen = (7) = 0 1 1 1	
XOR'ed with new pixel (4) = 0 1 0 0	
	<hr/>
	0 0 1 1 = 3
and, (7) = 0 1 1 1	
XOR'ed with new pixel (7) = 0 1 1 1	
	<hr/>
	0 0 0 0 B = 0

Therefore,

4 3 4 3 7 0 7 0	Pixel Value of Raster
x - x - x - x -	Pattern redrawn in COMP2 mode
3 3 3 3 0 0 0 0	Result

Mode 7. **CLEAR2** — If the overlaying bit= - No change
 If the overlaying bit= x pixel AND'ed with NOT primary pen

CLEAR2 mode has the effect of clearing the bits which correspond to the primary pen.

primary pen = 1	=	0 0 0 1	
NOT primary pen = 14	=	1 1 1 0	
AND pixel = 3	=	0 0 1 1	
		0 0 1 0	= 2
NOT primary pen = 14	=	1 1 1 0	
AND pixel = 0	=	0 0 0 0	
		0 0 0 0	= 0

3 3 3 3 0 0 0 0	Pixel value of raster
x - x - x - x -	Pattern drawn in CLEAR2 mode
2 3 2 3 0 0 0 0	New Pixel Values

RASTER VISIBILITY

Just as you were able to restrict the flow of graphic information to the raster, you may restrict the flow of graphic information read from the raster to the screen or other device (printer, disc, host). Raster visibility is determined by two conditions:

- the setting of the system read mask, and
- the setting of the display mode

NOTE: The system read mask and display mode do not alter the content of raster memory — only the way in which you view it.

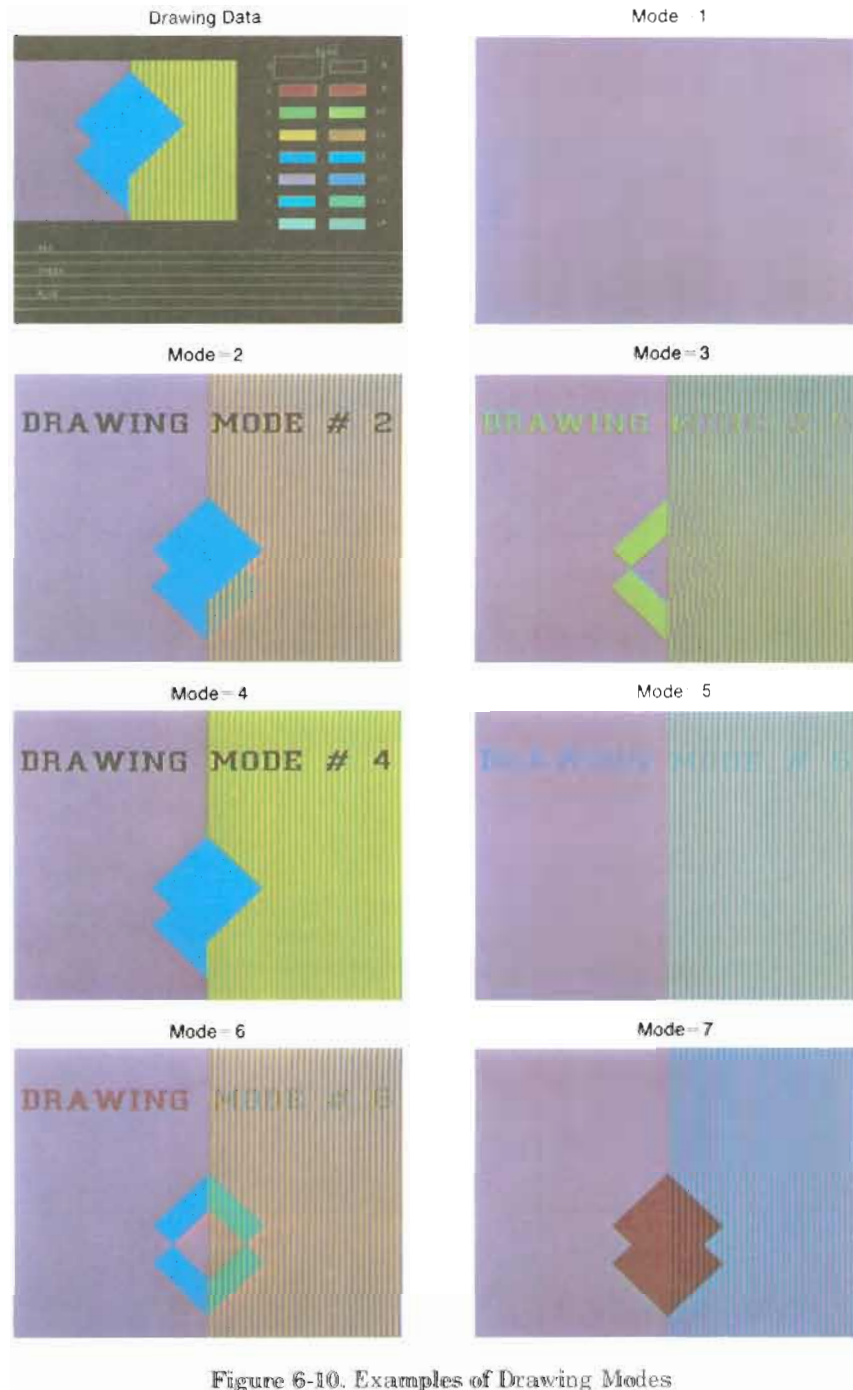


Figure 6-10. Examples of Drawing Modes

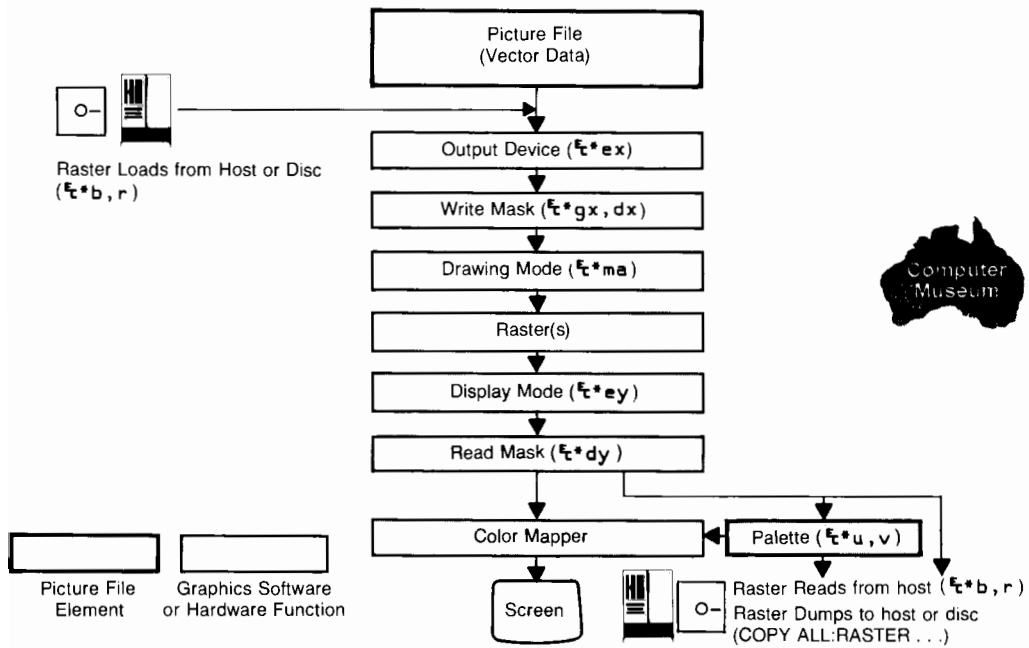


Figure 6-11. Raster Visibility

Turning On Graphics Video

`r*d`

This command performs the same function as the `r*dc` command. It ensures that raster data will be mapped to the screen according to the display modes, set below.

System Read Mask

The system read mask is set by supplying a mask in the command:

`r*d<mask>v`

where `<mask>` is an integer in the range `(-32738, ..., -32767)`

Depending upon the setting of the display mode, (explained below) either the low 4-bits or the low 8-bits determine the mask.

Display Modes

Display modes are set using the command:

$$\text{t*e<display mode>y}$$

where <display mode> is an integer in the range (-32768, ..., 32767)

The low 3 bits determine the display mode by the following criteria:

If bit 0 = bit 1 = 0 8-bit mask mode
 If bit 0 = 1 Display primary raster
 If bit 1 = 1 Display auxiliary raster
 If bit 2 = 1 Foreground/background mode

NOTE: You must have an auxiliary raster to activate 8-bit mask mode, display auxiliary raster, and foreground/background mode.

The effect of each display mode will be illustrated with a sample display mask.

Display Mode

0 0 0 Set 8-bit mask mode

The pixel value of the primary and auxiliary rasters is AND'ed with the 8-bit visibility mask. Then, both rasters are OR'ed.

4 pixels in the auxiliary raster: 7 7 7 7
 4 pixels in the primary raster: 3 0 3 0
 8-bit Visibility Mask (197): 1 1 0 0 0 1 0 1

Apply bits 0-3 of the mask to the primary raster, and bits 4-7 to the auxiliary raster (8-bit mask mode).

$\begin{array}{r} 7 = 0 1 1 1 \\ \text{AND } 12 = 1 1 0 0 \\ \hline 0 1 0 0 = 4 \end{array}$	$\begin{array}{r} 3 = 0 0 1 1 \\ \text{AND } 5 = 0 1 0 1 \\ \hline 0 0 0 1 = 1 \end{array}$
--	---

Therefore,

4 pixels from the auxiliary raster: 4 4 4 4
 4 pixels from the primary raster: 1 0 1 0

OR the two rasters together, and we get:

$$\begin{array}{r}
 4 = 0\ 1\ 0\ 0 \text{ OR'ed with} \\
 1 = 0\ 0\ 0\ 1 \\
 \hline
 0\ 1\ 0\ 1 = 5
 \end{array}
 \qquad
 \begin{array}{r}
 4 = 0\ 1\ 0\ 0 \text{ OR'ed with} \\
 0 = 0\ 0\ 0\ 0 \\
 \hline
 0\ 1\ 0\ 0 = 4
 \end{array}$$

The display shows: 5 4 5 4

Display Mode
0 0 1 Display Primary Raster Only

The primary raster is AND'ed with the low 4 bits of the system read mask.

Display Mode
0 1 0 Display Auxiliary Raster Only

The auxiliary raster is AND'ed with the low 4 bits of the system read mask.

Display Mode
0 1 1 OR Mode

The primary raster is OR'ed with the auxiliary raster.

Display Mode
1 0 0 Foreground/Background, 8-bit mask mode

This mode allows a picture to be drawn and left in the background buffer as other pictures are drawn over it in the foreground buffer.

The pixel value of the primary and auxiliary rasters is AND'ed with an 8-bit read mask. Then, if the value of a pixel in the primary raster is not zero, it is displayed. Otherwise, the pixel of the auxiliary memory is displayed.

$$\begin{array}{l}
 4 \text{ pixels in the auxiliary raster: } \quad 7\ 7\ 7\ 7 \\
 4 \text{ pixels in the primary raster: } \quad 3\ 0\ 3\ 0 \\
 \text{Read Mask (147): } \quad 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1
 \end{array}$$

Apply bits 0-3 of the mask to the primary raster, and bits 4-7 to the auxiliary raster (8-bit mask mode).

$$\begin{array}{r}
 7 = 0\ 1\ 1\ 1 \\
 \text{AND } 9 = 1\ 0\ 0\ 1 \\
 \hline
 0\ 0\ 0\ 1 = 1
 \end{array}
 \qquad
 \begin{array}{r}
 3 = 0\ 0\ 1\ 1 \\
 \text{AND } 3 = 0\ 0\ 1\ 1 \\
 \hline
 0\ 0\ 1\ 1 = 3
 \end{array}$$

Raster Manipulations

Therefore,

4 pixels from the auxiliary raster: 1 1 1 1
4 pixels from the primary raster: 3 0 3 0
If primary pixel <> 0 display primary pixel
If primary pixel = 0 display auxiliary pixel

1 1 1 1	auxiliary raster
3 0 3 0	primary raster
<hr/>	
3 1 3 1	display

Display Mode

! 1 ! 0 ! 1 ! Display Primary Raster Only

The primary raster is AND'ed with the low 4 bits of the system read mask. Note that although foreground/background mode is specified, it has no effect since the auxiliary raster is not visible.

Display Mode

! 1 ! 1 ! 0 ! Display Auxiliary Raster Only

The auxiliary raster is AND'ed with the low 4 bits of the system read mask. Note that although foreground/background mode is specified, it has no effect since the auxiliary raster is not visible.

Display Mode

! 1 ! 1 ! 1 ! Foreground/Background, 4-bit mask mode

The pixel value of the primary and auxiliary rasters is AND'ed with a 4-bit read mask. Then, if the value of a pixel in the primary raster is not zero, it is displayed otherwise, the pixel of the auxiliary memory is displayed.

4 pixels in the auxiliary raster: 7 7 7 7
4 pixels in the primary raster: 3 0 3 0
Visibility Mask (15): 1 1 1 1

Apply the mask to the primary and auxiliary raster.

7 = 0 1 1 1	3 = 0 0 1 1
AND 15 = 1 1 1 1	AND 15 = 1 1 1 1
<hr/>	<hr/>
0 1 1 1 = 7	0 0 1 1 = 3

Therefore,

4 pixels in the auxiliary raster: 7 7 7 7
4 pixels in the primary raster: 3 0 3 0

If primary pixel \neq 0 display primary pixel
 If primary pixel = 0 display auxiliary pixel

7 7 7 7	auxiliary raster
3 0 3 0	primary raster
<hr/>	
3 7 3 7	display

APPLICATIONS

See Appendix A. APPLICATIONS NOTES for examples using read and write masks.

RASTER DATA TRANSFERS

Binary data from raster memory can be directly transferred to printer, disc, or datacomm. In addition, a disc or host computer can transfer binary data directly to raster memory. This discussion will cover the following transfers:

- Raster Dumps — transfer of the entire raster memory to printer, disc, or datacomm
- Raster Loads — transfer of raster data from disc or datacomm to a *selected portion* of raster memory. NOTE: Raster loads are compatible with raster dumps. A dump to a disc file can be reloaded to recreate the raster.
- Raster Reads — transfer of *selected data* from raster to a host computer.

Raster Manipulations

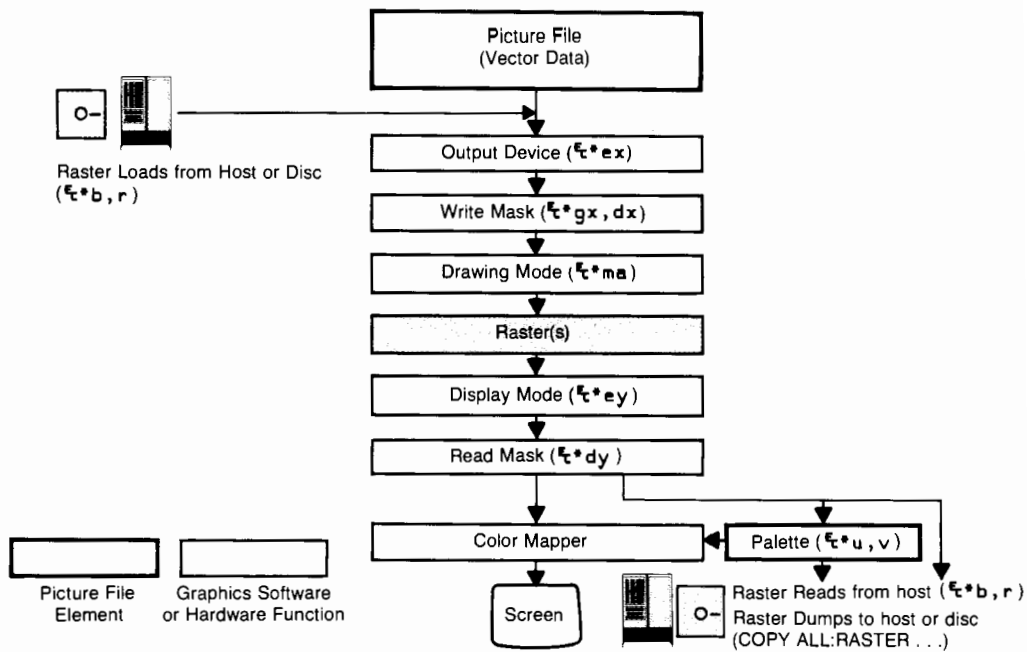


Figure 6-12. Raster Data Transfers

The following paragraphs describe the format in which raster data transfers occur. Section 7. GRAPHICS INPUT/OUTPUT OPERATIONS describes how to perform raster dumps to datacomm, disc, and printer and raster loads to the terminal.

Raster Dumps

The transfer of binary data from raster to a device (printer, disc, or datacomm) is referred to as a *raster dump*. Raster dumps are initiated using the device control command:

```
COPY ALL :RASTER
```

The data is dumped in the following format:

- dump dimensions
- printer specifications (optional)
- data type and format
- start record
- data transfer
- termination record

RASTER DUMP DIMENSIONS. The dimensions of the picture to be dumped are sent in the escape sequences:

```

ε*r<number of data bytes>S
ε*r<number of data lines>T
ε*r<number of data planes>U

```

The terminal sends the escape sequence:

```

ε*r512s390t4U (color dump)
ε*r512s390t1U (black and white dump)

```

PRINTER SPECIFICATIONS. Device specifications are selected in the Printer Configuration Menus described in Section 7. These specifications are sent in the escape sequences:

- auto-centering specification

```

ε*rH Disable auto-centering
or
ε*rG Enable auto-centering

```

- printer scaling specification

```
ε*r<x scale>,<y scale>v
```

DATA TYPE AND FORMAT. Raster dumps can be either black and white, halftone, or color, depending upon the setting of the Printer Configuration Menu. They can also be sent in a horizontal or vertical format. These specifications are sent in the commands:

```
ε*r<mode>F
```

where <mode> is determined by the low 3 bits as follows:

<i>bit</i>	<i>meaning</i>
0	Bit off = Horizontal Format; Bit on = Vertical Format
1	Not used
2	Bit off = Color; Bit on = Black and White

NOTE: Dumps to datacomm and disc files are always COLOR.

START RECORD. The start record consists of the initialize transfer command:

```
ε*rA
```

and the color palette (ε*v commands) if the dump is color.

DATA TRANSFER

Color Raster Dumps. The data is dumped beginning with the starting address of raster memory (column 0, row 0, plane 0). After a row is dumped, the column address is reset to 0, the plane address is incremented by one, and the row address remains the same. After all of the planes of a row are dumped, the column and plane addresses are reset to 0, and the row address is incremented by one.

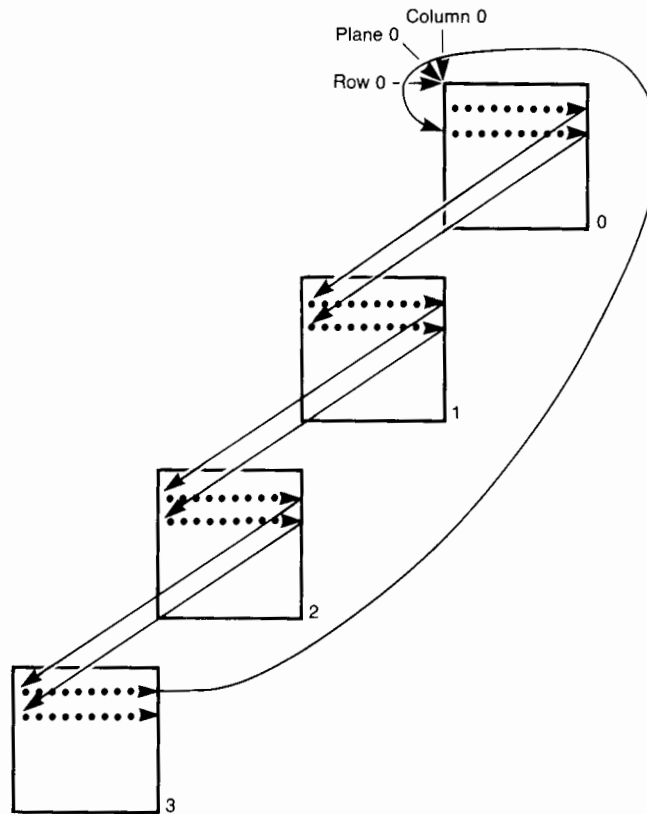


Figure 6-13. Color Raster Dump

B&W or Halftone Dumps. In order to produce black and white or halftone raster dumps, the terminal OR's the raster planes to form a single plane on/off bit pattern. The data is dumped beginning with the starting address of raster memory (column 0, row 0). After a row is dumped, the column address is reset to zero, and the row is incremented by one.

Transfer Commands. The data is transferred in an escape sequence with the following format:

$$\text{\textbackslash} * b \langle x \text{ offset} \rangle x \langle \text{number of data bytes} \rangle \begin{matrix} W \\ V \end{matrix} \langle \text{data block} \rangle$$

Each portion of this command is described below:

$$\text{\textbackslash} * b \langle x \text{ offset} \rangle x$$

This data compaction command specifies a temporary offset for the current row of data. This x offset corresponds to the number of bits starting current raster line which may be transferred as zeros. For dumps, the offset is always a multiple of eight.

Following the x offset is a command which specifies how many *bytes* of data will be dumped from the current raster row. This information can be contained in one of two escape sequences:

$$\text{\textbackslash} * b \langle \text{number of data bytes} \rangle V \langle \text{data block} \rangle$$

where $\langle \text{number of data bytes} \rangle = (0, \dots, 255)$

After this transfer, the column address is set to the x origin; the plane address is incremented by one; and the row address remains the same.

$$\text{\textbackslash} * b \langle \text{number of data bytes} \rangle W \langle \text{data block} \rangle$$

where $\langle \text{number of data bytes} \rangle = (0, \dots, 255)$

After this transfer, the row address is incremented by one, the plane address is set to the z origin, and the column address is set to the x origin.

After an $\text{\textbackslash} * b(V, W)$ sequence, the actual binary data is sent as a block. The next transfer begins at the incremented raster memory address.

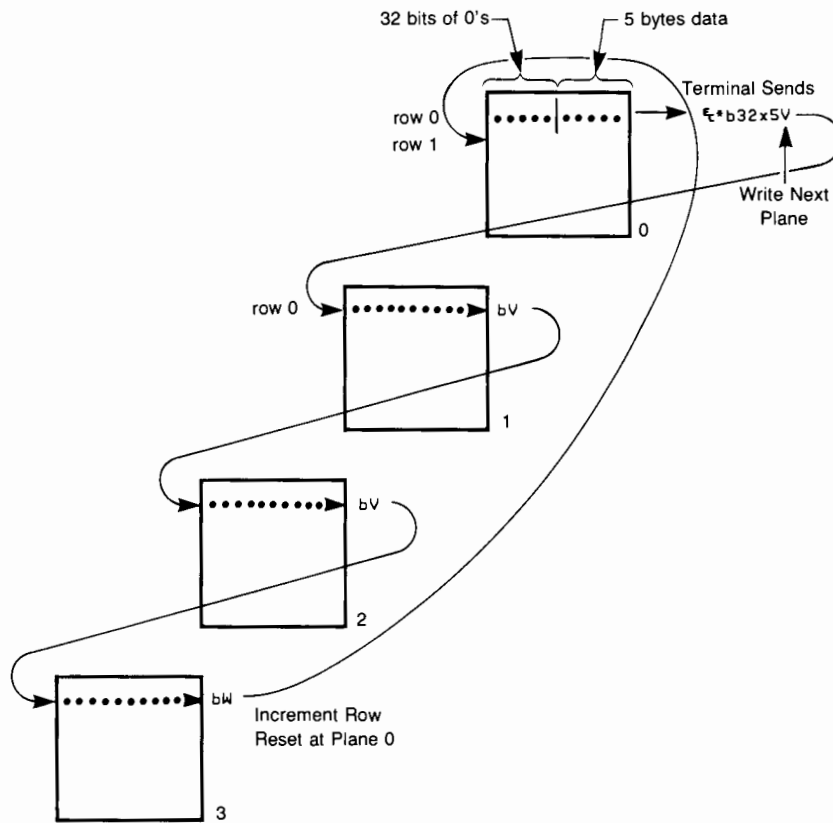


Figure 6-14. Raster Dump/Data Compaction

NOTE: The row origin of a dump defaults to the uppermost visible scan line of the display. The row origin can be redefined via the Raster Configuration Menu, (see Section 7. GRAPHICS INPUT/OUTPUT OPERATIONS).

TERMINATION RECORD. The transfer is terminated by the command:

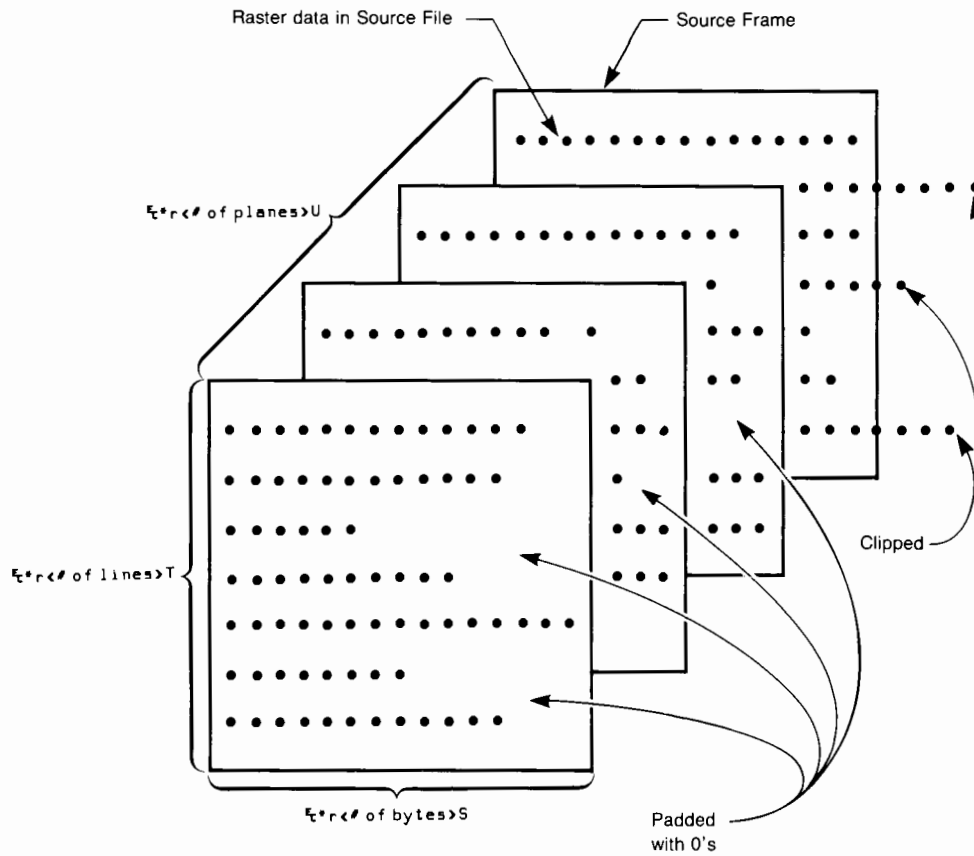
$\text{t} * rB$

Raster Loads

A *raster load* refers to the transfer of raster data from a source file on a host computer or disc to raster memory. A raster load consists of:

- Source File Dimensioning Commands
- Raster Offset and Format Commands
- Start Record
- Raster Data
- Termination Record

DIMENSIONING THE SOURCE FILE. The terminal can receive up to $9999 \times 9999 \times 9999$ bits of data. You must first quantify the amount of data sent to the terminal by using `*r(S,T,U)` commands. These commands are normally the first line of the source file. The `*rS` command specifies how many bytes of data will be sent in each line, the `*rT` command specifies many lines of data per plane will be sent, and the `*rU` command specifies how many planes will be sent. The data dimensioned by these commands is called the *source frame*. If the source frame is larger than the raster or smaller than the source file, the extra data is discarded. If the source frame is smaller than the raster but larger than the source file, the extra bits are padded to zeros.



The data in the source file is dimensioned by the $\epsilon^*r(S, T, U)$ commands. Only the data within the source frame will be recognized by the raster. Short lines are padded with zeros.

Figure 6-15. From Source File to Source Frame

PARTITIONING THE SOURCE FRAME (WINDOW FRAME). You may also choose to display only a portion of the source frame. The $\epsilon^*r(M, N, D, P, Q, R)$ commands specify the *window frame* within the source frame. The $\epsilon^*r(M, N, D)$ commands specify the upper left-hand corner of the window frame relative to the source frame.

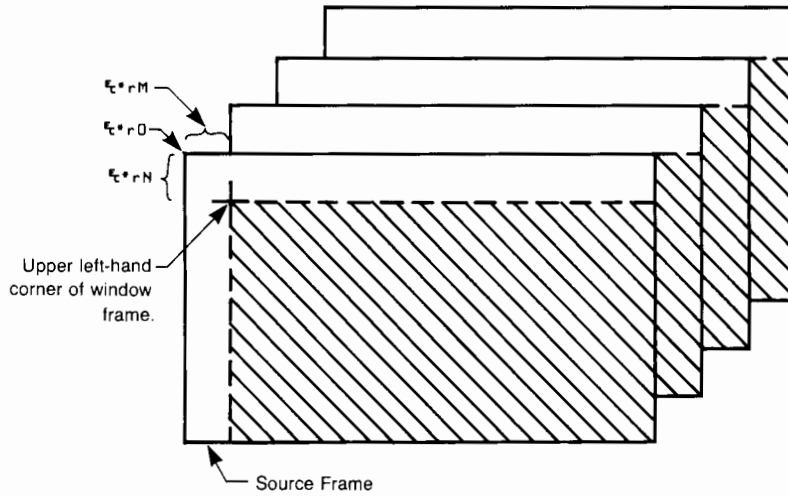


Figure 6-16. From Source Frame To Window Frame (Upper Lefthand Corner)

The $\epsilon^*r(P, Q, R)$ commands specify length, width and height of the window frame.

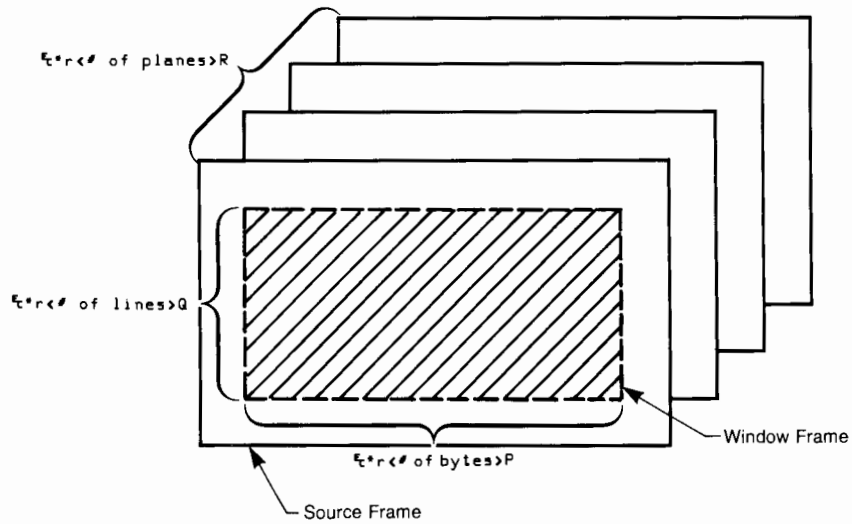


Figure 6-17. From Source Frame to Window Frame (Truncating the Data)

PARTITIONING THE RASTER. You can specify the portion of raster memory to receive the incoming data from the window frame. The point of raster memory to which the window frame is mapped is determined by the $\text{r}(X, Y, Z)$ sequences which specify the X, Y, and Z offsets (the upper left-hand corner) of the *window frame* relative to the raster memory.

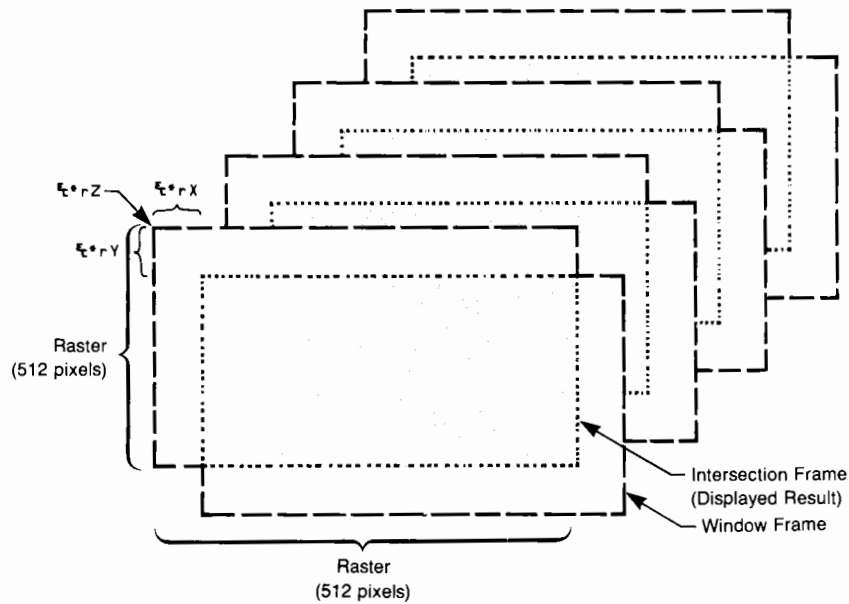


Figure 6-18. From Window Frame to Intersection Frame

The intersection of the raster and the window frame is called the Intersection Frame. The Intersection Frame is the resultant raster picture.

RASTER LOAD FORMATS. The default origin of a raster load is the upper left corner of the raster. The window frame is mapped to the raster in a horizontal format: left to right, top to bottom.

You can also reset the origin to the lower left corner of the raster, and load the raster in a vertical format: bottom to top, left to right.

Set Raster Load Format: $\text{r}\langle\text{format}\rangle\text{f}$

where	<format>	meaning
	0	horizontal (default)
	1	vertical

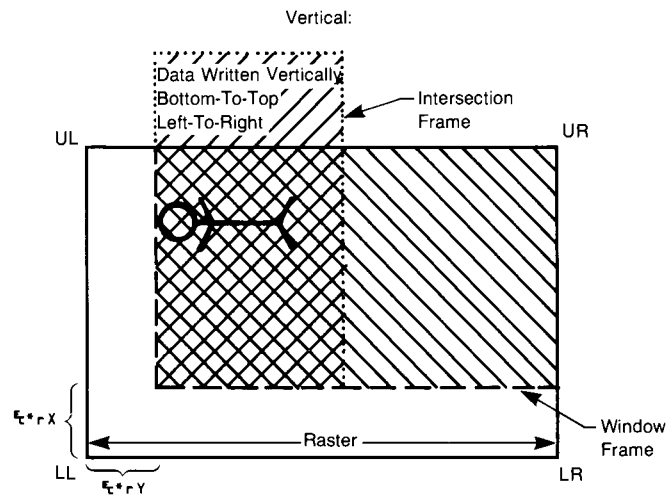
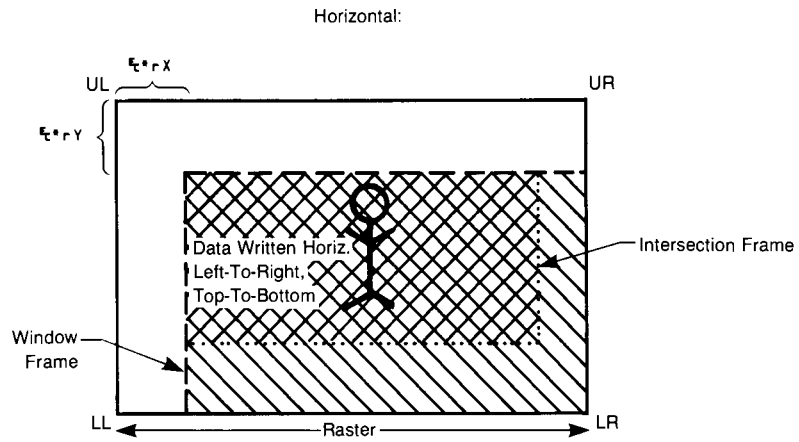
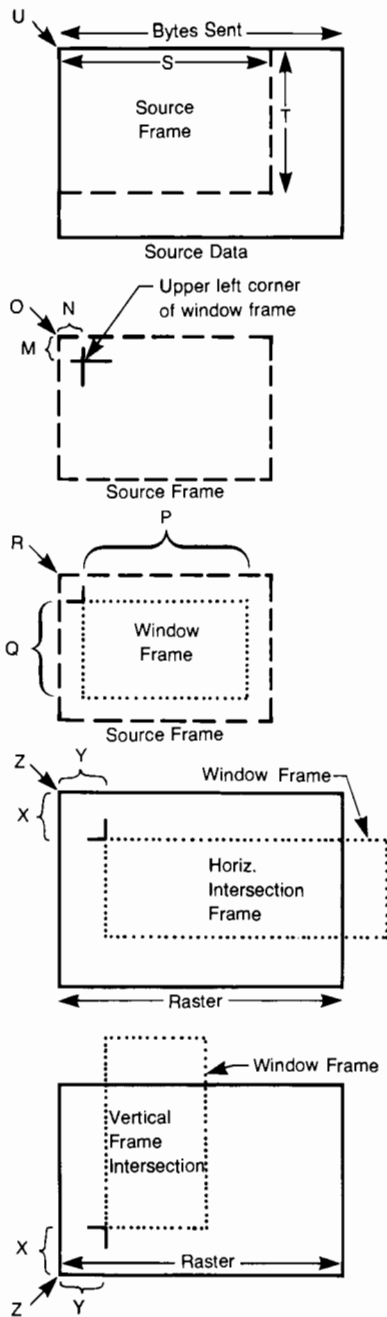


Figure 6-19. Raster Load Formats

Raster Manipulations



1. Raster data in Source File is dimensioned for the X, Y, and Z directions using the $\text{E}^*r(S, T, U)$ commands to define a Source Frame.

2. Using $\text{E}^*r(M, N, O)$ commands, the terminal is set to ignore "O" raster planes, "N" raster lines, and "M" raster bytes of data at the beginning of the transfer.

3. Using $\text{E}^*r(P, Q, R)$ commands, the terminal prepares to receive "R" raster planes, "Q" raster lines/plane, and "P" raster bytes/line.

4. An X, Y, Z offset is set in raster memory using $\text{E}^*r(X, Y, Z)$ commands.

5. The contents of the window frame are written into the receive frame (forming the I-frame) in either a horizontal or vertical format. (Set with $\text{E}^*rF.$)

Figure 6-20. Raster Load (Dimensioning and Offset Review)

START RECORD. If the raster load data was formerly raster dump data, the start record will contain the $\epsilon*v$ sequences described for raster dumps. The $\epsilon*v$ sequences will affect the *active* palette.

All raster loads must be initiated by the command:

$\epsilon*rA$

When this command is received, the terminal prepares to receive binary data. The "pen" position (starting point for writing raster lines) is initialized according to the X, Y, and Z offsets specified in the $\epsilon*r(X, Y, Z)$ commands. The starting point defaults to the upper left-hand corner of the screen if the format is horizontal, or to the lower left-hand corner of the screen if the format is vertical.

DATA TRANSFER. Data is written to the raster in the following format:

	x		U
$\epsilon*b$ <offset>	y	<# of data bytes>	V <data block>
	z		W

The terminal recognizes column, row, and plane temporary offsets for data loaded into raster memory. These offsets are specified with the following data compaction commands:

Set temporary column offset: $\epsilon*b<x\ bits>X$

where <x bits> = (0, ..., 9999)

This command specifies the number of bits to the right of the x origin on a raster line which may be set to zero.

Set temporary row offset: $\epsilon*b<dy\ bits>Y$

where <dy bits> = (0, ..., 9999)

This command specifies the number of raster lines which may be skipped relative to the current pen position. This is a relative specification, therefore, multiple row offset commands have a cumulative effect.

Example. $\epsilon*b\ 1y\ 2y\ 10w = \epsilon*b\ 3y\ 10w$

Set temporary plane offset: $\epsilon*b<dz\ bits>Z$

where <dz bits> = (0, ..., 9999)

This command specifies the number of raster planes which may be skipped relative to the current pen position. This is a relative specification, therefore, multiple plane offset commands have a cumulative effect.

Raster Manipulations

Row 0	0 0 0 0 0 0 0 0	x offset = 8
1	0 0 0 0 0 0 1 1	x offset = 6
2	1 1 1 1 1 1 1 1	x offset = 0
3	1 1 1 1 1 1 1 1	x offset = 0
4	0 0 0 0 0 0 1 1	x offset = 6
5	0 0 0 0 0 0 0 0	x offset = 8

Figure 6-21. Data Compaction

The $\epsilon * b(U, V, W)$ sequences are the same as those described for raster dumps.

$\epsilon * b \langle \text{number of data bytes} \rangle U \langle \text{data block} \rangle$

where $\langle \text{number of data bytes} \rangle = (0, \dots, 255)$

After this transfer, the column address is incremented to the next column; the row and plane addresses are unchanged. (NOTE: This command can be used multiple times if the raster data length is greater than 255 bytes.)

$\epsilon * b \langle \text{number of data bytes} \rangle V \langle \text{data block} \rangle$

where $\langle \text{number of data bytes} \rangle = (0, \dots, 255)$

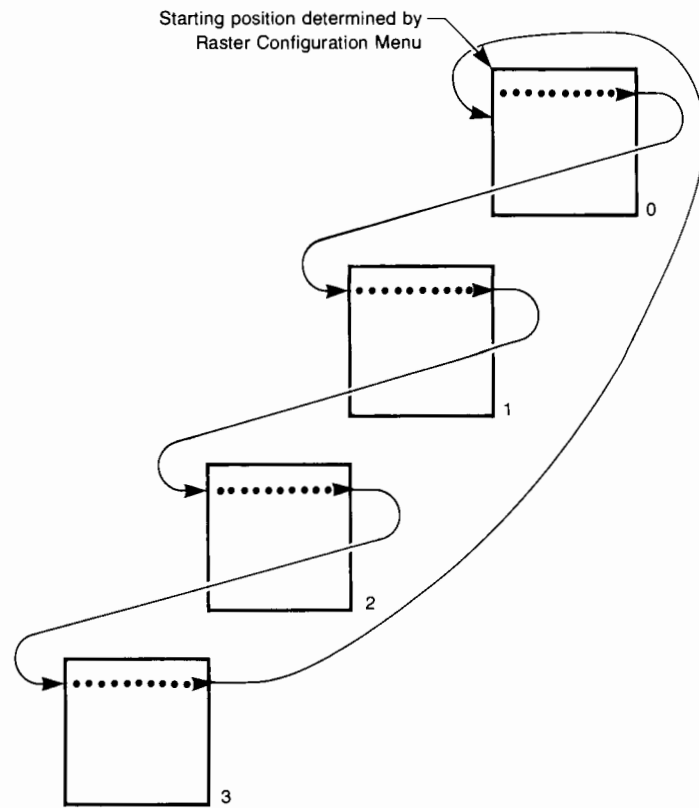
After this transfer, the column address is set to the x origin; the plane address is incremented by one; and the row address remains the same.

$\epsilon * b \langle \text{number of data bytes} \rangle W \langle \text{data block} \rangle$

where $\langle \text{number of data bytes} \rangle = (0, \dots, 255)$

After this transfer, the row address is incremented by one, the plane address is set to the z origin, and the column address is set to the x origin.

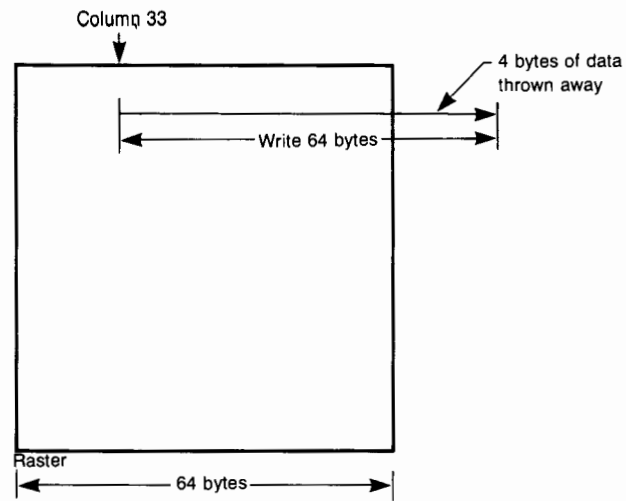
After an $\epsilon * b(U, V, W)$ sequence, the actual binary data is sent as a block. The next transfer begins at the incremented raster memory address.



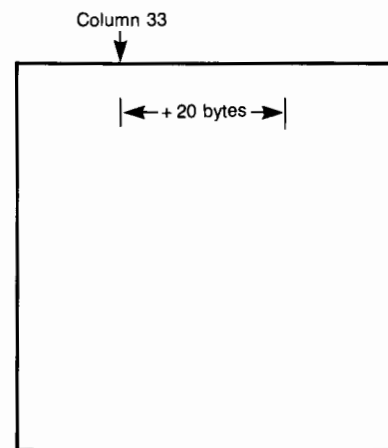
- ξ^*rA - Initialize Transfer
- ξ^*b10V - Load 10 bytes of data in line 0, plane 0
- ξ^*b10V - Load 10 bytes of data in line 0, plane 1
- ξ^*b10V - Load 10 bytes of data in line 0, plane 2
- ξ^*b10W - Load 10 bytes of data in line 0, plane 3
increment y address

Figure 6-22. Raster Load Format

Raster Manipulations



1. Send $t \cdot b32 \times 64W$
 $32x = 4 \text{ bytes}$



2. Send $t \cdot b32 \times 20W$

Figure 6-23. Raster Loads/Data Compaction

TERMINATION RECORD. The termination record consists of the `*rB` command, which resets the raster window dimensions and offsets to their default values (until the next `*rA`).

RASTER LOADS FROM DATACOMM

A remote computer can send image data to the terminal using a format similar to the one described above. Before the binary data for each record is sent, an ENQ/ACK handshake must be performed to allow time for the terminal to go into binary mode.

1. Send `*b64W`
2. When the terminal sends back `^`, send the 64 bytes.

The handshake is similar to that used when writing binary data to disc. (See Section 7. DEVICE CONTROL, of the *HP 2700 Alphanumeric Reference Manual*.) If no data is sent, (`*b0W` for a blank line), the ENQ/ACK handshake need not be performed.

NOTE: You can use the `*rE` (raster status) command, to check if ENQ/ACK handshaking is enabled. (See "Raster Status".) If the handshaking is not enabled, the host is responsible for supplying a delay.

HOW RASTER LOADS ARE AFFECTED BY DRAWING MODES. Raster loads are dependent upon the vector drawing mode in effect. Draw modes affect raster loads differently from picture file writes, since the raster load operation is unaware of the viewport background color.

A sample pattern consisting of 8 bits of a raster plane is used to cover all cases.

Drawing Mode 0. NOP

If overlaying bit = 0 No change
 If overlaying bit = 1 No change

raster plane bit pattern	0 0 0 0 1 1 1 1
incoming raster dump data	0 0 1 1 0 0 1 1
	0 0 0 0 1 1 1 1

Drawing Mode 1. CLEAR1 (CLEAR)

If overlaying bit = 0 No change
 If overlaying bit = 1 Turn off bit

raster plane bit pattern	0 0 0 0 1 1 1 1
incoming raster dump data	0 0 1 1 0 0 1 1
	0 0 0 0 1 1 0 0

Raster Manipulations

Drawing Mode 2. JAM1 (JAM) default

If overlaying bit = 0 Turn off bit
If overlaying bit = 1 Turn on bit

raster plane bit pattern	0 0 0 0 1 1 1 1
incoming raster dump data	0 0 1 1 0 0 1 1
	<hr/>
	0 0 1 1 0 0 1 1

Drawing Mode 3. COMP1 (COMP)

If overlaying bit = 0 No change
If overlaying bit = 1 NOT raster plane bit

raster plane bit pattern	0 0 0 0 1 1 1 1
incoming raster dump data	0 0 1 1 0 0 1 1
	<hr/>
	0 0 1 1 1 1 0 0

Drawing Mode 4. JAM2 (JAM)

Drawing Mode 5. OR (Set)

If overlaying bit = 0 No change
If overlaying bit = 1 Turn bit on

raster plane bit pattern	0 0 0 0 1 1 1 1
incoming raster dump data	0 0 1 1 0 0 1 1
	<hr/>
	0 0 1 1 1 1 1 1

Drawing Mode 6. COMP2 (COMP)

Drawing Mode 7. CLEAR2 (CLEAR)

Raster Reads

Data can be selectively read from the raster by the host using the following commands:

Specify starting point for raster read: $\text{r}^{\text{r}}\text{x}^{\text{c}}\text{y}^{\text{p}}\text{zA}$

Read binary block into datacomm: $\text{r}^{\text{r}}\text{b}^{\text{c}}\text{p}$

$\langle \text{number of data bytes} \rangle = (0, \dots, 255)$

This command reads a binary block. The $\langle \text{number of data bytes} \rangle$ of data requested are sent back upon termination of the sequence. After this transfer, the column address is incremented to the next column, the row and plane address are unchanged.

Read binary block into datacomm, set up next line: $\text{\textbackslash}*\text{b}\langle\text{number of data bytes}\rangle\text{q}$

$\langle\text{number of data bytes}\rangle = (0, \dots, 255)$

This command reads a binary block. The $\langle\text{number of data bytes}\rangle$ of data requested are sent back upon termination of the sequence. After this transfer, the plane address is incremented to the next plane, and the column address is set to the x origin.

Read binary block into datacomm, set up next line: $\text{\textbackslash}*\text{b}\langle\text{number of data bytes}\rangle\text{r}$

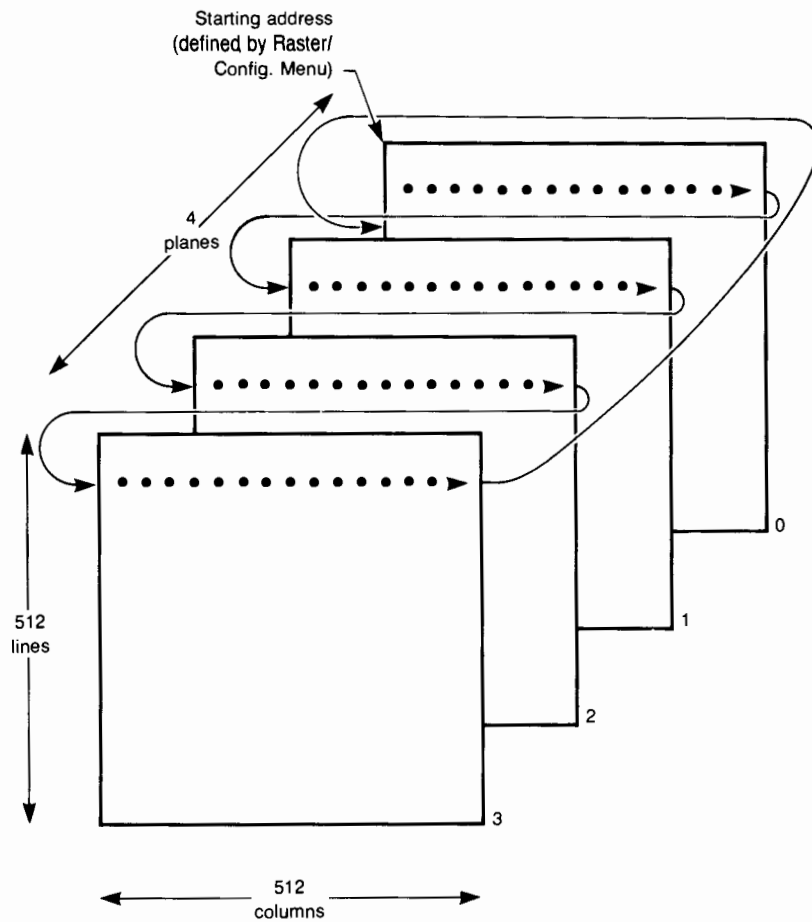
$\langle\text{number of data bytes}\rangle = (0, \dots, 255)$

This command reads a binary block. The $\langle\text{number of data bytes}\rangle$ of data requested are sent back upon termination of the sequence. After this transfer, the row address is incremented to the next line, the plane address is set to the z origin, and the column address is set to the x origin.

Example. The following example illustrates the raster read operation.

$\text{\textbackslash}*\text{r}60\text{x}50\text{y}1\text{zA}$	Select starting read point 60 pixels from the left of the raster, 50 pixels from the top, and one plane in.
$\text{\textbackslash}*\text{b}10\text{R}$	Read 10 bytes of binary data. Host sends DC1 Terminal transfers 10 bytes of data without a terminator.
$\text{\textbackslash}*\text{b}10\text{R}$	Read 10 bytes of data with same X and Z offset from next lower scan line. Host sends DC1 The terminal transfers 10 bytes with no terminator, et cetera.

Raster Manipulations



- ϵ^*rA - Initialize Transfer
- ϵ^*b10q - Read 10 bytes from line 0, plane 0, & send to host
- ϵ^*b10q - Read 10 bytes from line 0, plane 1, & send to host
- ϵ^*b10q - Read 10 bytes from line 0, plane 2, & send to host
- ϵ^*b10r - Read 10 bytes from line 0, plane 3, & send to host, increment y address
- ϵ^*b10q - Read 10 bytes from line 1, plane 0, & send to host
- ϵ^*b10q - Read 10 bytes from line 1, plane 1, & send to host
- ϵ^*b10q - Read 10 bytes from line 1, plane 2, & send to host
- ϵ^*b10r - Read 10 bytes from line 1, plane 3, & send to host, increment y address

Figure 6-24. Raster Reads

Raster Defaults

Raster defaults are set with the command: `␣*ri`

Graphics video	on
Screen format	horizontal
Horizontal window address	0
Vertical window address	0
Level window address	0
Level window dimension	9999
Vertical window dimension	9999
Level window dimension	9999
Horizontal size	9999
Vertical size	9999
Levels	9999
X offset	0
Y offset	0
Z offset	0

Raster Status

The following commands can be used to obtain raster information.

Read Raster Status: `␣*rE`

Upon receipt of <DC1>, the terminal returns one byte of status as follows:

RESET.	Bit 0 set to 1, the terminal has been reset.
READY.	Bit 1 set to 1, the terminal is ready for raster data. (always '1')
REPEAT.	Bit 2 set to 1, repeat the transfer. (always '0')
ENQ/ACK.	Bit 3 set to 1, ENQ/ACK handshake is enabled.
ERROR.	Bit 4 set to 1, error has occurred.
UNUSED.	Bit 5 set to 0.
UNUSED.	Bit 6 set to 1.
UNUSED.	Bit 7 set to 0.

`␣*r<mode>J` — Return raster size status.

<mode> = 0,1

This inquiry returns an ASCII string with the terminal's raster size. If mode is 0 or empty, the returned string is '512,390'. If the mode is 1, the string is '512,390,4', where 512 = X dimension, 390 = Y dimension, and 4 = Z dimension.

Return Model Number: `␣*rK`

The terminal returns the string "2703A".

Graphics Input/Output Operations — 7

INTRODUCTION

This section discusses the control and operation of the various graphics input/output devices which may be connected to the terminal.

Topics include:

- Graphics Hardcopy Devices: plotters and printers recognized by the terminal
- Raster Dumps to Printers, Discs, and Datacomm
- Plotting from the Picture File
- Graphics Input Devices: thumbwheels, GID key, and the graphics tablet
- External Monitors and Cameras
- Graphics I/O Device Configuration Menus

Refer also to Section 9. USING YOUR TERMINAL WITH PLOTTERS AND PRINTERS, of the *HP 2700 User's Manual* for basic instructions on connecting and operating plotters and printers. See Section 7. DEVICE CONTROL, of the *HP 2700 Alphanumeric Reference Manual* for information on escape sequence control of alphanumeric printers, the HP-IB menu, and special shared printer configurations.

GRAPHICS HARDCOPY DEVICES

The terminal recognizes two types of hardcopy devices: "external" and "shared". External devices are connected to the terminal through the terminal's RS-232 interface. Shared devices use the Shared Peripheral Interface. External devices are addressed via the device control keys as EX PRNTR or EX PLOT. Shared devices which can self-identify are accessed as PLOTTER or PRINTER, (or PLOTTER <#>/PRINTER <#> in the case of multiple plotters/printers). Shared devices which cannot self-identify are addressed by their HP-IB address.

RASTER DUMPS

You can transfer the contents of raster memory to a raster printer, disc file, or datacomm file using the keystroke command:

DEVICE CONTROL

<i>Operation</i>	<i>Source</i>	<i>Destination</i>
COPY	FILE	RASTER
TRANSFER	ALL	EX PRNTR PRINTER <#> HP-IB# <#> <filename>;<volumename> DATACOM

or by using the appropriate ϵ, c command.

The portion of memory to be dumped is determined by the Raster Configuration Menu. (See figure 7-12 and table 7-5.)

Refer also to Section 6. RASTER MANIPULATIONS, for a complete description of raster memory and raster data transfer operations.

Raster Dump To Printer

The transfer of raster data to a printer is controlled by the Printer Configuration Menus. (See figures 7-8, 7-9 and table 7-2.) These menus allows you to specify:

- Picture Orientation (Horizontal/Vertical) (See figure 7-1.)
- 8-Color, 125-Color, Half-tone, or Black and White Pictures (See figure 7-2.)
- Picture Size
- Data Compaction
- Formfeeds

See "Graphics I/O Device Configuration Menus" in this section for details.

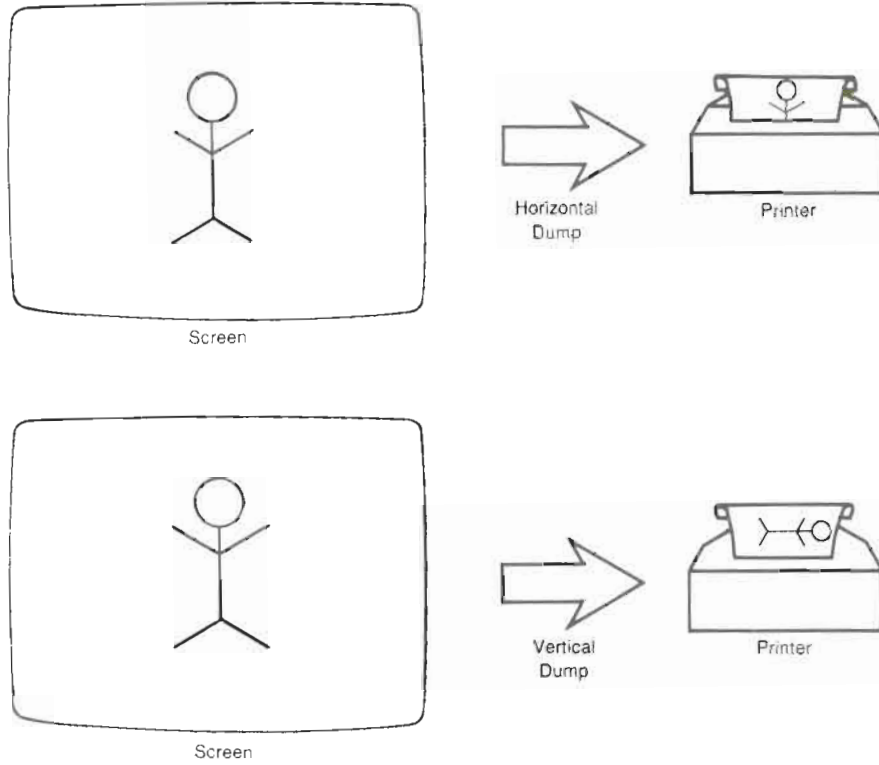


Figure 7-1. Raster Dump Orientations

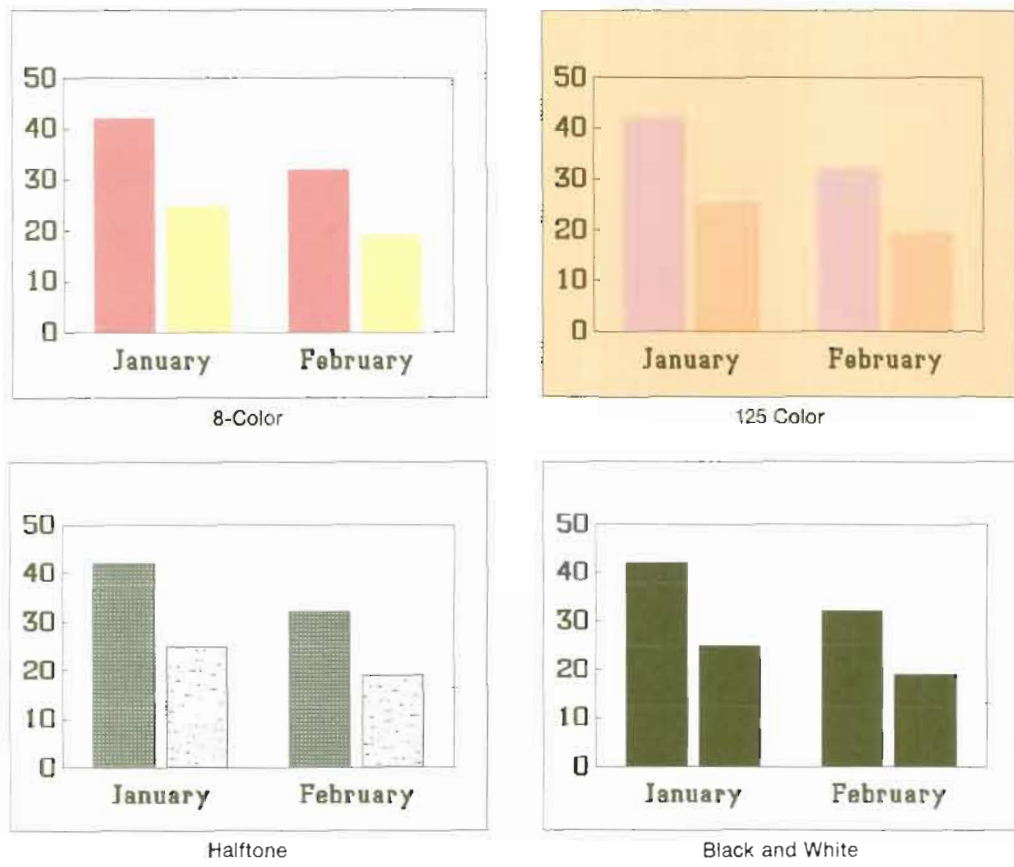


Figure 7-2. Raster Dump Types

Raster Dumps To Disc Or Datacomm Files

Raster dumps to a disc or datacomm file are always color dumps sent in a horizontal format. To perform a raster dump to disc, use the command:

```
COPY ALL :RASTER TO <filename>;<volumename>
```

or use the equivalent \uparrow, c sequence.

You can also dump the data to disc or datacomm in a different format (i.e. Halftone/Vertical) by following these steps:

Step 1. Access the "External Printer Configuration Menu", and set the parameters, i.e. "HTONE", "VERTICAL".

Step 2. Execute the COPY :RASTER command, specifying EX PRNTR as the first destination device before your disc or datacomm filename. (You need not have an external printer connected to the terminal.)

```
*COPY ALL :RASTER TO :EX PRNTR, <filename>;<volumename>
```

The parameters set in the printer configuration menu will apply to all subsequent destinations.

PLOTTING FROM THE PICTURE FILE

The terminal uses information stored in the picture file to generate a picture in HPGL, the command language recognized by most Hewlett Packard plotters. Only the data contained in the active viewport is plotted.

Before you plot the picture, check the settings in the Plotter Configuration Menu. (See figure 7-10 and table 7-3 in this section.)

To initiate the plot use the command:

```

DEVICE CONTROL  PLOT,  EX PLOT
                  HP-1B# <#>
                  PLOTTER <#>

```

or use an equivalent \uparrow, c command.

To stop a plot, press or .

NOTES:

- You cannot resume a plot which was stopped.
- After pressing the key to stop the plot, you may notice a delay before the terminal returns to normal operation.

- If the correct command sequence is executed, but the plot is unsuccessful, an error message is displayed in the command message window. If the plot was initiated by an escape sequence, the error can be inquired with the REPORT STATUS OF COMMAND command. (See Section 9. STATUS of the *HP 2700 Alphanumeric Reference Manual*.)

Color Raster Dumps

The terminal has alternate drivers to support two non-HP raster devices:

- PrintaColor GP1024
- RAMTEK 4100

These printers can be used to obtain color hardcopy of raster memory.

Both devices connect to the terminal through the RS-232 interface and are controlled by the External Printer Configuration Menu. They require the following settings in the menu:

Option This field selects the alternate driver used to send data in a recognizable format to the printer. The selections are:

0 — HP raster dump sequences (default)

2 — PrintaColor GP1024 raster dump sequences

3 — PrintaColor GP1024 (inverted black and white) raster dump sequences

4 — RAMTEK 4100 raster dump sequences

5 — RAMTEK 4100 (inverted black and white) raster dump sequences

NOTE: Values 3 and 5 print white dots on the screen as black dots.

Expansion Width (Height)

The color printers can reproduce up to 125 colors if these fields are set to '2' or greater. If the expansion fields are set to '1', only 8 colors will be reproduced.

Content A default setting of HTONE in this field lets the printers produce 125 different colors. When the expansion of fields are set to 1.0 or less, a content selection of COLOR may be used. This limits colors to 8 and produces crisper small prints.

Exp Mode This field should be set to its default value 'INT'.

NOTES:

- The settings of the `Auto-Centering` and `Data Compaction` fields have no effect.
- The `Formfeed` field settings produce the normal effects.
- Color alpha is not supported.

GRAPHICS INPUT DEVICES**Thumbwheels and Gid Key**

The keyboard graphics input device consists of the thumbwheels (used to control the graphics cursor) and the GID key.

Computer resident programs can interact with device through the following two escape sequences:

Read graphics cursor position: `␣*53^`

The terminal returns the current graphics cursor position in the current cursor definition units (virtual or display).

`<x>,<y> = +nnnnn,+nnnnn`

Wait for key; read graphics cursor position: `␣*54^`

The terminal returns the current graphics cursor position in the current cursor definition units, as well as the decimal representation of the ASCII character which terminated the wait.

`<x>,<y>,<keycode> = +nnnnn,+nnnnn,nnn`

If the key value returned is 128, then the GID key has been pressed.

NOTE: The sensitivity of the thumbwheels is controlled by the Keyboard Configuration Menu, discussed in "Graphics I/O Device Configuration Menus" in this section. (See figure 7-11 and table 7-4.)

Graphics Tablet

DESCRIPTION. The graphics tablet (HP part no. 13273T) consists of a flat drawing surface and a stylus connected to the tablet by a cable. When the stylus is moved about on the tablet surface, its movements are tracked by the graphics cursor on the display. The stylus tip contains a switch which behaves like the GID key.

INSTALLATION

CAUTION

Set the terminal power switch to OFF before connecting or disconnecting the tablet to prevent damage to the terminal.

The tablet is installed by connecting the free end of the tablet cable to the Tablet Interface at the rear of the terminal. (See figure 7-3.) When the terminal is powered up, the tablet is ready for use.

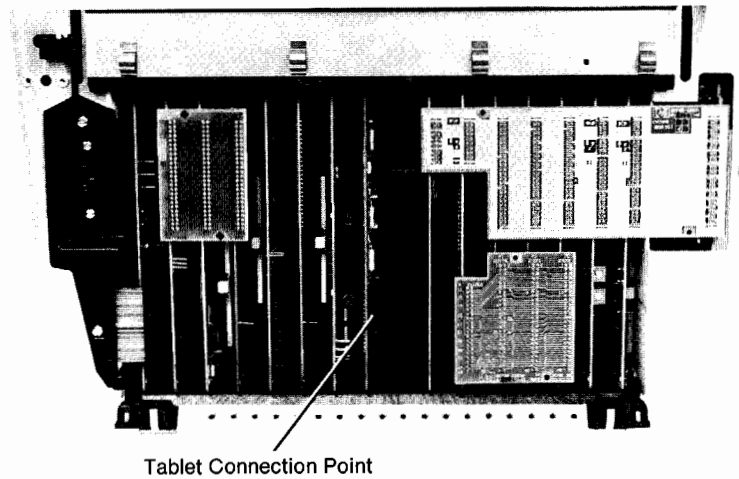


Figure 7-3. Tablet Connection

OPERATION. When the tablet is operational, its surface is sensitive to nearness of the stylus tip. When the tip is brought very close to the tablet surface, the cursor on the terminal display moves to a corresponding point on the display screen.

CONTROLS. The tablet has three pushbutton controls which are used to select the tablet operating conditions:

- Rightmost Control — selects absolute or relative mode.
- Center Control — centers the graphics cursor on the display screen.
- Leftmost Control — selects lefthand/righthand operation.

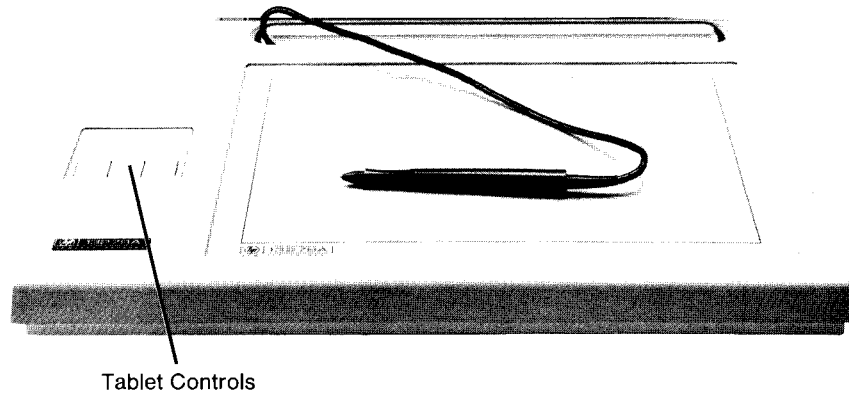


Figure 7-4. Graphics Tablet Controls

ABSOLUTE/RELATIVE MODE. The graphics tablet operates in two modes; absolute and relative. In absolute mode, the graphics cursor on the display tracks the movement of the stylus.

In relative (or "stroke emulation") mode, the terminal senses only the direction and length of the stylus movement. The absolute location of the stylus is ignored, as illustrated in figure 7-5. It is possible, while in relative mode, to move the cursor entirely off the active viewport and back on again. The terminal continues to keep track of the cursor location.

NOTE: The sensitivity of the stylus in relative mode is affected by the settings for thumb-wheel sensitivity in the Keyboard Configuration Menu. (See figure 7-11 and table 7-4.)

LEFTHAND/RIGHTHAND OPERATION. The tablet can be rotated 180 degrees to reorient it for the lefthanded user. The origin (now at the upper right corner) is relocated to the lower left corner by pressing the innermost control button. The tablet controls are not affected.

USING THE TABLET IN GRAPHICS OPERATIONS. When it is defined for relative mode, the tablet can be used to perform graphics manipulations such as zooming, panning, windowing, and color modification.

TABLET ESCAPE SEQUENCES. The tablet can interact with an application program with the same escape sequences used by the terminal GID device:

- Read graphics cursor position: $\text{␣}*\text{␣}3\text{␣}$
- Wait for key; read graphics cursor position: $\text{␣}*\text{␣}4\text{␣}$

(See "Thumbwheels and GID Key" in this section for details.)

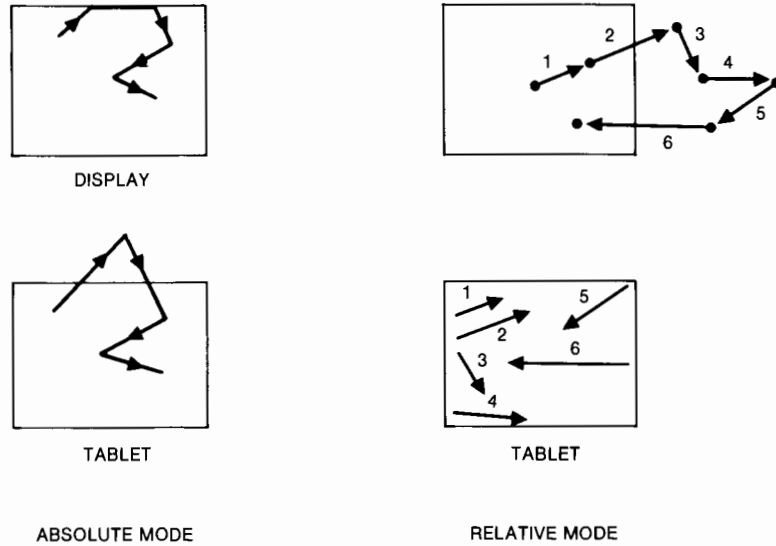


Figure 7-5. Cursor Movement in Absolute and Relative Mode

EXTERNAL MONITORS AND CAMERAS

The terminal can display an external image of its graphics memory using the external video interface (EVI). The raster data can be routed to the color mapper on the EVI, and the image can be displayed on devices compatible with the RGB output signal.

The External Video Interface is the leftmost member of the graphics board set. Both graphics topplanes extend one slot to the left for this addition.

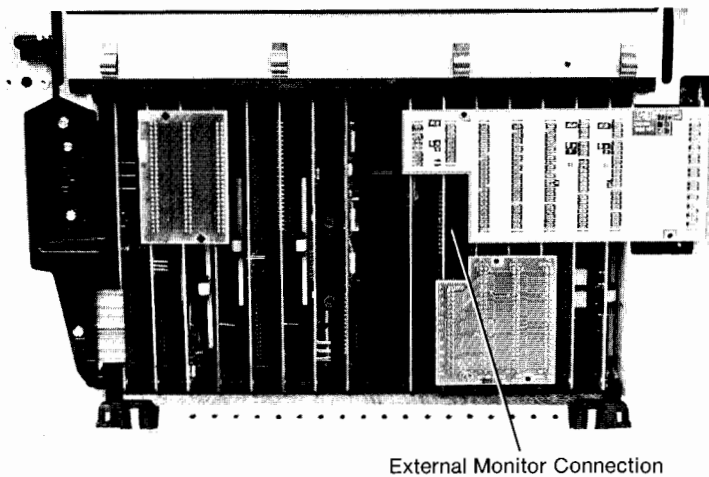


Figure 7-6. External Monitor (Camera) Connection

You can program the terminal such that the external monitor receives different data from the terminal screen.

Step 1. Select one or other of the rasters as the output device for the monitor:

⌘*2Y

Step 2. Set the `Ext Monitor` field in the Terminal Configuration Menu to "YES". (See figure 7-13 and table 7-6 in this section.)

Step 3. Select the remaining raster buffer as the output device for the screen:

⌘*1Y

In this case, data written to the auxiliary raster will appear on the monitor, and data written to primary raster will appear on the display. Note that if a picture is displayed on the monitor and the `Ext Monitor` field is subsequently set to "NO", the picture will remain on the monitor, but will not be updated. If the a new color palette is subsequently activated, the EVI color mapper will not be updated.

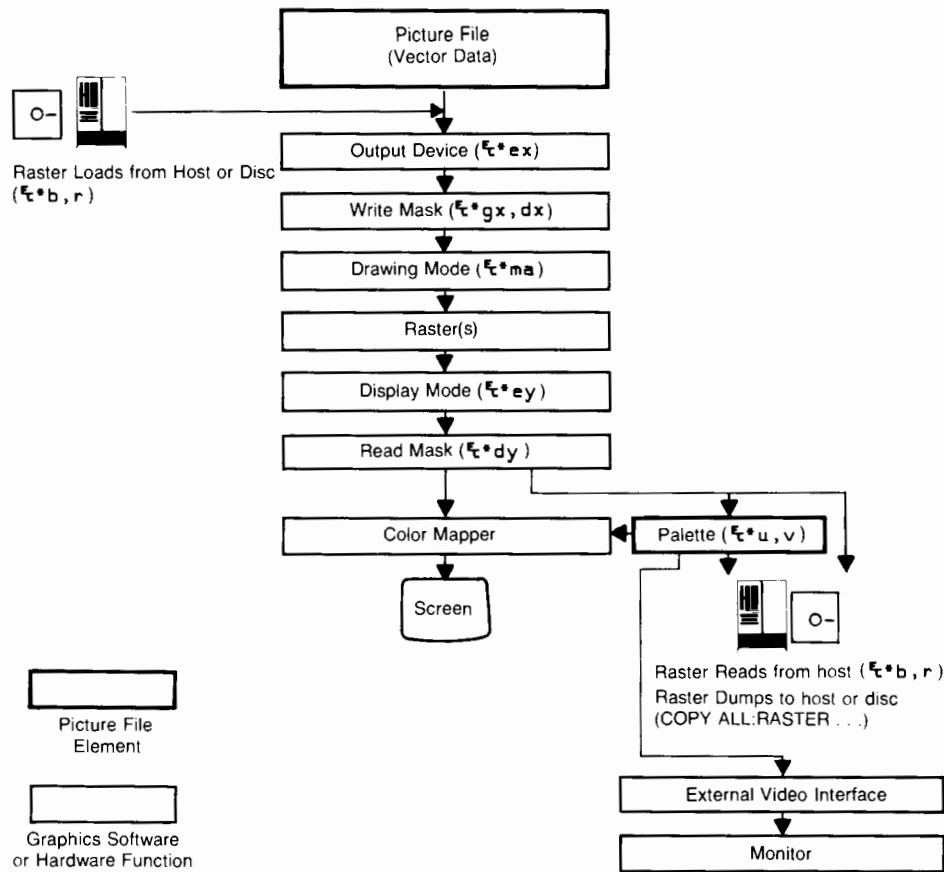


Figure 7-7. External Monitor Operation

GRAPHICS I/O DEVICE CONFIGURATION MENUS

The terminal has special configuration menus through which you can exercise more control over I/O devices. These include:

- Printer Configuration Menu
- Plotter Configuration Menu
- Keyboard Configuration Menu (for the GID devices)
- Raster Configuration Menu
- Terminal Configuration Menu (for External Monitors)

Device menus can be filled using keystrokes or using escape sequences. This subsection first discusses the process by which the menus are filled. The various parameters are explained under "Menu Fields".

Keyboard Operation Of Configuration Menus

Configuration menus can be displayed on the screen, and their content can be altered using function keys. Configuration values may also be stored in non-volatile memory where they are saved even when the terminal is turned off. The process of changing, activating, and storing configuration values is discussed in the following paragraphs.

HOW TO DISPLAY THE MENUS. To display the configuration menus, use the following keystroke sequence:

, config keys

The following function key choices will be made available to you:

Each function key, when pressed, causes a particular configuration menu to appear on the screen.

HOW TO MAKE MENU SELECTIONS. When you display a configuration menu, the function keys are redefined to assist you in selecting the various parameters within the menu.

These keys are explained in table 7-1.

Table 7-1. Configuration Menu Function Keys

SAVE CONF IG	After you have set the menu fields to the desired values, you may activate them and save them in non-volatile memory by using this key.
NEXT CHOICE PREVIOUS CHOICE	These keys allow you to select parameters that are restricted to a system-defined list of selections. (Such a field is underlined.) Use either the NEXT CHOICE or PREVIOUS CHOICE function key to cycle through the list of selections until the desired one is displayed.
DEFAULT VALUES	Use this key to set all the fields in the menu to their default values.
POWER ON VALUES	Use this key to set all the fields in the menu to the values currently stored in non-volatile memory.
ACTIVE VALUE	Use this key to set all the fields in the menu to the current active values.

DSPYFNC	Pressing this key alternately enables and disables Display Functions mode within a configuration menu.
config keys	Pressing this key removes the menu from the screen WITHOUT activating any of the selections and without saving the selections in non-volatile memory. Normal operation is restored, and the configuration set of the function key labels is displayed.
TEMPSAVE	Pressing this key activates the fields of the menu on the screen, but does not save them in non-volatile memory.

To change a selection on a menu, perform the following steps:

Step 1. Place the cursor at the character position to be changed. This can be done using the `TAB` key or the cursor-positioning keys. The `TAB` key moves the cursor to the next selection field each time the key is pressed.

Step 2. If the field is underlined, use the `NEXT CHAR` or `PREV CHAR` function keys to make your selection.

Step 3. If the field is not underlined, enter the desired value from the keyboard.


NOTE: If you should make an illegal entry that causes an error message, the keyboard will lock up, and any subsequent keystrokes will only cause the terminal bell to beep. You must press the `RETURN` key to unlock the keyboard and resume operation.

Step 4. Activate the altered values using the `SAVE CONFIG` and `TEMPSAVE` function keys. If you wish to return to normal terminal operation without activating the changed values, use the `config keys` function key.

ACTIVATING AND SAVING CONFIGURATION MENU FIELDS. In order to activate the values in the menu fields, you must press either the `TEMPSAVE` or `SAVE CONFIG` function keys. Pressing the `TEMPSAVE` key activates the values in the menu fields (but does not store them in non-volatile memory), and restores normal terminal operation. Pressing the `SAVE CONFIG` key not only activates the menu fields, but it stores those values in non-volatile memory. These values will be saved when the terminal is powered down.

NOTES ON CONFIGURATION VALUES STORED IN NON-VOLATILE MEMORY. The last set of configured values saved by the user is the set stored in non-volatile memory. If none has been stored by the user, the default set is stored. When a menu is called to the display screen, the values currently in use are displayed. When the terminal is turned on or hard reset, the set of configuration values stored in non-volatile memory becomes the active set.

Printer and Plotter Configuration Menus

To access these menus, press , `config keys`, `device config`. You should see the following display:

```
EXTERNAL PLOTTER
EXTERNAL PRINTER
HP-IB                (if the Shared Peripheral Interface is installed)

PRINTER1            (if a self-identifying shared printer is connected)

PLOTTER1            (if a shared plotter is connected; see "Plotter Configuration Menus"
                    in this section.)

PRINTER2            (if more than one shared printer is connected to the terminal)
```

Position the cursor under the label of the device you wish to address, and press `show menu`.

A menu of one of the following formats will be displayed on the screen.

External printers and non-identifying shared printers are controlled by the "External Printer" configuration menu. Self-identifying shared printers are controlled by the "Printer[N]" configuration menus.

All plotter configuration menus have the same form. External plotters and non-identifying shared plotters are controlled by the "External Plotter" configuration menu. Self-identifying shared plotters are controlled by the "Plotter[N]" configuration menus.

Keyboard Configuration Menu

The keyboard configuration menu contains settings which control the sensitivity of the vertical and horizontal thumbwheels as well as the graphics tablet when it is defined for relative mode. (See table 8-4.)

The Raster Configuration Menu

This menu can be used to send selective portions of data from the raster to a printer during `COPY ALL :RASTER` operations. (See table 7-5.)

Terminal Configuration Menu

The Terminal Configuration Menu contains the field "Ext Monitor" which determines whether graphics visibility updates are routed to the External Video Interface. (See table 7-6 and "External Monitors and Cameras" in this section.)

```

PRINTER[N] CONFIGURATION
Raster Orientation  HORIZ      Content  HTONE      Option  0
Expansion Width    1.000      Height  1.000      Exp Mode INT
Auto-Centering    NONE       Data Compact NO       Formfeed YES
    
```

Figure 7-8. PrinterN Configuration Menu

```

EXTERNAL or HP-IB[N] PRINTER CONFIGURATION
Raster Orientation  HORIZ      Content  HTONE      Option  0
Expansion Width    1.000      Height  1.000      Exp Mode INT
Auto-Centering    NONE       Data Compact NO       Formfeed YES
Printer Nulls     0

SAVE  NEXT  PREVIOUS  DEFAULT  POWER ON  ACTIVE  TEMP  config
CONFIG CHOICE CHOICE  VALUES  VALUES  SAVE   keys
    
```

Figure 7-9. External or HP-IB[N] Printer Configuration Menu

```

PLOTTER CONFIGURATION
Absolute P1 ( 520, 380), P2 ( 10520, 8000)      Use Absolute NO
Preserve Aspect Ratio YES                      Chart Advance NONE
Number of Pens 4 Pen Speed 0 Areafill Delta 0      Option Mask 0
    
```

Figure 7-10. Plotter Configuration Menu

```

Keyboard
Bell Tone  HIGH D  Bell Volume MEDIUM  Bell Duration MEDIUM
Repeat Rate SLOW  Delay Rate SLOW

Thumbwheel Sensitivity: Horizontal 4
                             Vertical 4

SAVE  NEXT  PREVIOUS  DEFAULT  POWER ON  ACTIVE  TEMPSAVE  config
CONFIG CHOICE CHOICE  VALUES  VALUES  DSPYFNC  keys
    
```

Figure 7-11. The Keyboard Configuration Menu

```

                    RASTER CONFIGURATION
Horiz Limit 511      Vert Limit 389      Option 0

```

Figure 7-12. Raster Configuration Menu

```

                    TERMINAL CONFIGURATION
                                Frame Rate 60
                                Datacom/Printer Port1/Port2
RETURN Def R      RETURN-ENTER NO
Language    USASCII
REMOTE OFF      BLOCK OFF      MODIFY OFF      Auto LF OFF
Local Echo OFF      CAPS Lock OFF      Start Col 1      ASCII 8 Bits OFF
XmitFncn(A) NO      SPOW(B) NO      InhEolWrp(C) NO  Line/Page(D) LINE
InhHndShk(G) NO      Inh DC2(H) NO      InvertWrp(M) NO      InhDcTst(W) NO
FldSeparator F      BlkTerminator E
ESC Abort NO      DiscBuffSize*1K      Alph Dsp Mem*16K      Ext Monitor NO

```

* Press SAVE CONFIG twice to invoke this field. WARNING: Hard reset will occur.

Figure 7-13. The Terminal Configuration Menu

Escape Sequence Operation

Configuration values for graphics input/output devices can also be set with escape sequences. The user specifies the device class, device index, a set of working values, the menu field to be altered, and new value with the command:

```

          a
 $\text{Esc}^*q\langle\text{device id}\rangle c\langle\text{index}\rangle i\langle\text{working values}\rangle d\langle\text{field id}\rangle f\langle\text{value}\rangle V$ 
          p

```

NOTES:

- Configurations set with escape sequences are not saved in non-volatile memory unless the SAVE CONFIG function key is executed.
- If the configuration menus have been locked with the Esc^*q1L command, attempts to change values will generate an "illegal operation" graphics error.

SPECIFY DEVICE CLASS

$\text{\textasciitilde}q\langle\text{device id}\rangle c$

where	<id>	Device Class
	1	PRINTERx
	2	PLOTTERx
	3	GID
	4	RASTER
	5	EXTERNAL MONITOR

This command selects the device class for subsequent $\text{\textasciitilde}q$ sequences. The device class remains selected until another class is selected. The device class is specified as an integer <device id>, as shown.

SPECIFY DEVICE INDEX

$\text{\textasciitilde}q\langle\text{device index}\rangle i$

An index must be selected for all devices. Device classes which contain one device, such as RASTER, always have a device index of '1'. A device index of '0' for the PRINTER or PLOTTER class, specifies the external printer/plotter. A device index of '1' or greater for hardcopy devices refers to a self-identifying shared printer or plotter. The device index for shared devices can be determined by a "Show Devices" listing.

NOTE: This index remains in effect until a subsequent index is selected.

SET WORKING PARAMETERS. This command is optional. If working parameters are not set, they will default to the active values.

Set active values: $\text{\textasciitilde}qa$

Set default values: $\text{\textasciitilde}qd$

Set power on values: $\text{\textasciitilde}qp$

SELECT MENU FIELD AND NEW VALUE

$\text{\textasciitilde}q\langle\text{field id}\rangle f\langle\text{value}\rangle v$

Each field of the configuration menus is identified by a field id. The field id's and their values are listed under "Menu Fields".


Configuration Menu Fields

The following tables list the menu fields which control graphics I/O operations. Only those fields which can be altered using F^*q sequences are listed. Values for escape sequence operation are listed on the left. Values for keyboard operation are listed on the right.



Table 7-2. Printer Configuration Menu Fields

Raster Orientation Field ID: 1 Values: 0 = HORIZ 1 = VERT	This field specifies the format of a raster dump transfer. Values: HORIZ = The dump begins in the upper left corner of the screen, and continues left to right, top to bottom. VERT = The dump begins in the lower left corner of the screen, and continues bottom to top, left to right. Default: HORIZ
Content Field ID: 2 Values: 0 = COLOR 1 = ORed pixel black & white 2 = HTONE 3 = ALPHA	This field specifies the type of data to be transferred. Values: COLOR = The raster is dumped according to specifications described in Section 6. RASTER MANIPULATIONS, "Raster Data Transfers". (See also "Color Raster Dumps" in this section.) ORed = The four bits of each pixel are logically OR'ed. The result is a black and white dump. HTONE = This mode is used to dump color pictures to a black and white printer. Before the transfer begins, the color area fills are redrawn in a predefined hatched pattern. The patterns are automatically set to reflect the luminacities of each pen in the active color palette.

You can preview the patterns before initiating the transfer by pressing , `redraw modes`, `HALFTONE`. If different patterns are desired, you can enter the Color Menu and alter the colors until the required pattern is found. Customized patterns may also be specified using `^t*u` sequences. (See "Halftoning" in Section 3. COLOR.)

ALPHA = This mode is set when the printer is not capable of handling a raster dump. An attempt to dump graphics with this mode aborts the dump and returns an error message.

Default: `HTONE`

Expansion Width (Height)
 Field ID: 3
 Values:
`<x factor>,<y factor>`
`(0.0000,..., 4.000)`

This field specifies the width and height of the graphics hardcopy. The factors are set separately to allow for independent scaling. "Factor" is defined as the number of dots desired on the hardcopy device divided by the number of pixels copied from the raster.

EXAMPLE: If a scale factor of 1.0 is used for a raster dump to the 2631G graphics printer, the resulting picture is considerably smaller than the paper width. To fill the paper, use the following formula:

$$\text{Scale Required} = \frac{\text{width 2631G (number of dots)}}{\text{raster width (pixels)}}$$

$$976/512 = 1.906$$

Note that the same width and height factor should be entered to maintain the picture's aspect ratio.

Values:
`(0.0000,..., 4.0000)`

Default: `1.000`

Exp Mode Field ID: 4 Values: 0 = internal expansion 1 = external expansion	<p>The expansion field provides control over where picture expansion takes place. If the printer cannot execute the expansion specified in the <code>Expansion Width (Height)</code> fields above, the <code>INT</code> option should be used.</p> <p>Values:</p> <p><code>EXT</code> = expansion is handled by the printer.</p> <p><code>INT</code> = The data is expanded by the terminal.</p> <p>Default: <code>INT</code></p>
Auto-Centering Field ID: 5 Values: 0 = NO 1 = YES 2 = NONE	<p>This field controls picture centering on printers which have this feature.</p> <p>Values:</p> <p><code>NO</code> = Disable printer auto centering</p> <p><code>YES</code> = Enable printer auto centering</p> <p><code>SAME</code> = No auto-centering command is sent to the printer; it is left in the same state.</p> <p>Default: <code>NONE</code></p>
Option Field ID: 6 Values: (0, ..., 255)	<p>Entering a non-zero number in this field selects a non-standard HP data format for raster dumps as defined by an alternate driver. If an alternate driver does not exist, an error message is displayed. (See also "Color Raster Dumps" in this section.)</p> <p>Default: <code>0</code></p>
Formfeed Field ID: 7 Values: 0 = NO 1 = YES	<p>This field controls the termination formfeed at the end of a copy or transfer to a printer.</p> <p>Values: <code>YES, NO</code></p> <p>Default: <code>YES</code></p>

Data Compact Field ID: 8 Values: 0 = NO 1 = YES	This field controls the terminal's raster dump data compaction feature. With compaction enabled, leading and trailing "0" values are stripped from the dump and specified by an $\text{E}*\text{bx}$ sequence. (See "Raster Data Transfers" in Section 6. RASTER MANIPULATIONS.)
	Values: YES = enable data compaction NO = disable data compaction. (Should be used if the printer does not understand $\text{E}*\text{bx}$ commands.)
	Default: NO

Table 7-3. Plotter Configuration Menu Fields

Absolute P1 (P2) Field ID: 1 Values: <xll><yll><xur><yur> (-32768, ..., 32767)	P1 and P2 are coordinate pairs specifying the lower left and upper right plotting limits. These limits are only used if the Use Absolute field (below) is set to "YES". Values: (xll, yll, xur, yur) are integers in the range (-32768, ..., +32767)
	Default: P1 = 520,380 P2 = 10520,8000
Use Absolute Field ID: 2 Values: 0 = NO 1 = YES	Values: YES = the limits specified in the Absolute P1 (P2) field will be used. NO = the limits specified by the plotter control keys will be used.
	Default: NO

<p>Preserve Aspect Ratio Field ID: 3 Values: 0 = NO 1 = YES</p>	<p>Values: YES = the active graphics view will be mapped into the largest area possible (justified to the lower left corner of the platen within the active plotter limits) which does not distort the aspect ratio of the view. NO = the lower left and upper right corners of the active graphics view will be mapped directly into P1,P2. Default: YES</p>
--	--

<p>Chart Advance Field ID: 4 Values: 0 = NONE 1 = FULL 2 = HALF</p>	<p>Values: NONE = no chart advance instruction will be given upon normal termination of the plot. FULL = the full chart advance instruction is sent to the plotter. HALF = the half chart advance instruction is sent to the plotter. NOTE: If this field is set to "FULL" or "HALF", the plotter is asked whether it is a chart advance plotter with properly loaded paper. Default: NONE</p>
---	---

<p>Number of Pens Field ID: 5 Values: (1, ..., 99)</p>	<p>Enter the number of pens for your plotter. The following algorithm will be used to determine which of the terminal's "pens" will be mapped to which plotter pen. ("Terminal Pen #" MOD "# of Plotter Pens") = "Plotter Pen Used" If the result is 0, then "Plotter Pen Used" = "# of Plotter Pens". If the result is greater than the number of plotter pens, the pen will not be changed.</p>
--	---

EXAMPLE: If your plotter has four pens, vectors plotted in terminal pen 15 will be plotted in pen 3 on the plotter.

$$(15 \text{ MOD } 4) = 3$$

Default: 4

Pen Speed
Field ID: 6
Values:
(0, ..., 99)

This field specifies the speed of the plotting pen in centimeters/second. If the value is invalid for the plotter, it is ignored.

Values:
(0, ..., 99)

NOTE: If you are plotting on paper, enter "0". If you are plotting on transparencies, enter "10". If you are plotting area fills on transparencies enter "4".

Default: 0

Areafill Delta
Field ID: 7
Values:
(0, ..., 99)

This field specifies the spacing of pen strokes for an area fill in plotter units.

Values:
(0, ..., 99)

NOTE: If you are using thick pen tips, enter 20. If you are using thin pen tips, enter 10.

Default: 0

Option Mask
Field ID: 8
Values:
(0, ..., 255)

The option mask is an 8-bit mask whose bits specify certain options, as follows:

<i>bit</i>	<i>meaning</i>
0	determines whether graphics pen 0 is plotted. If the bit = 0, the vector is not plotted. If the bit = 1, the vector is plotted.

NOTE: Because the background pen is normally drawn in the terminal's pen 0, setting this field to 1 may produce unusual results.

Default: 0

Table 7-4. GID Fields

<p>X resolution Field ID: 1 Values: (1, ..., 250)</p>	<p>This field controls the sensitivity of the horizontal thumbwheel and the graphics tablet when it is defined for relative mode.</p> <p>Values: 1 (most sensitive) — 250 (least sensitive)</p> <p>Default: 4</p>
<p>Y resolution Field ID: 2 Values: (1, ..., 250)</p>	<p>This field controls the sensitivity of the vertical thumbwheel and the graphics tablet when it is defined for relative mode.</p> <p>Values: 1 (most sensitive) — 250 (least sensitive)</p> <p>Default: 4</p>

Table 7-5. Raster Configuration Menu Fields

<p>Horiz Limit (Vert Limit) Field ID: 1 Values: (0, ..., 511)</p>	<p>This field determines the portion of raster memory to be dumped to a printer, disc or datacomm file, or sent to the External Video Interface.</p> <p>Default: 511, 389</p>
<p>Option Field ID: 2 Values: (0, ..., 255)</p>	<p>Entering a non-zero value in this field selects a non-standard HP data format for raster loads as defined by an alternate driver. If an alternate driver does not exist, an error message is displayed.</p> <p>Default: 0</p>

Table 7-6. Terminal Configuration Menu

Ext Monitor Field ID: 1 Values: 0 = NO 1 = YES	This field determines whether graphics visibility updates are routed to the External Video Monitor. (See "External Monitors and Cameras" in this section for details.) Values: YES, NO Default: NO
---	--

GRAPHICS DEVICE CONFIGURATION INQUIRY

The graphics device configuration inquiry command returns information about the active device's current values.

`^t*q<mask>^`

See Section 8. INQUIRY COMMANDS AND STATUS REQUESTS, for details.

Inquiry Commands and Status Requests

GRAPHICS INQUIRY COMMANDS

Included in some graphics command families are graphics inquiries. These inquiries can be used by a program on a host computer to obtain information on the various settings controlled by the command family. This information includes pens, palettes, objects, and views. In addition, a general graphics status request, `g*s` can be used to check the status of most graphics parameters. Table 8-1 lists the various graphics inquiry and status commands.

Table 8-1. Graphics Inquiry and Status Commands

<code>g*e<mask>, <view#>^</code>	Inquire view
<code>g*g<mask>^</code>	Inquire object
<code>g*h<mask>^</code>	Inquire vector list
<code>g*i<mask>^</code>	Inquire pick
<code>g*q<mask>^</code>	Inquire device
<code>g*s<mask>^</code>	Inquire graphics status
<code>g*u<mask>, <pattern pen>^</code>	Inquire hardcopy formats
<code>g*v<mask>, <pen#>, <palette#>^</code>	Inquire palette
<code>g*w<mask>^</code>	Inquire view

You can select specific items for inquiry by setting a bit mask within the inquiry command. Each binary bit position in the mask is used to select a different item. For example, if the mask has a value of "1", the first inquiry item will be selected. A value of "3" will select the first and second items. A value of "6" will select the second and third items.

Mask Value	Inquiry Items Returned
0	0 0 0 0 0 Nothing
1	0 0 0 0 1 1st item
2	0 0 0 1 0 2nd item
3	0 0 0 1 1 1st and 3rd items
4	0 0 1 0 0 3rd item
5	0 0 1 0 1 1st and 3rd items
.	.
.	.
.	.

The mask values can range between 32767 and -32768. Negative values are evaluated as two's complement numbers. Some inquiries have additional parameters such as pen or palette numbers. Refer to the specific inquiry for information on these additional parameters.

When a graphics inquiry is made, the terminal will respond with one or more bytes of information followed by a block terminator. All information is returned in ASCII format, separated by commas. Coordinates are returned in a fixed format, consisting of a sign and five digits. Leading zeros are used as required to provide a fixed number of digits (i.e. +00100, -01234). This allows you to use simple input statements without the need to mask or shift bits. The format of returned data is listed for each command in table 8-2.

If the DC1 handshake protocol is enabled, the status block is not actually sent until receipt of a DC1 character. If the DC1 character is used, only one status request can be enabled while the terminal is waiting for a DC1. When the DC1 is received, the last status block requested will be sent.

The terminal configuration determines the terminating characters sent following the status block (%/ or %). Graphic inquiry and status requests turn on an echo suppress mode in the terminal. This prevents information echoed back from the computer from being displayed on the screen. Once a status block has been sent, characters received by the terminal will not be displayed until one of the following control characters is received: @, %, %, %, %, %, %, %, or %. With the exception of % and % the terminating control code itself will be executed.

The terminal expects the status information to be echoed and uses the terminating control character to turn off the suppress echo mode. If the computer does not echo the status back, a suitable control character must be returned to the terminal to turn off the echo suppress mode.

Information on data communications and terminal configuration is given in the *HP 2700 Alphanumeric Reference Manual*.

Inquire View (`%*e<mask>,<view>^`)

This command returns view information. If no view is specified, information on the active view will be returned. The command will be ignored if the number of parameters is 0 or more than 2 or if `<view>` is not in the range 0-255. If the `<mask>` is 0 then a null string will be returned. If the specified view does not exist and `<mask>` `<>` 1, dummy values are returned.

Inquire Implied Object (`%*g<mask>^`)

This command returns information on the implied object. The command will be ignored if the number of parameters is not 1. If the `<mask>` is 0 then a null string will be returned. If the implied object does not exist and `<mask>` `<>` 1, dummy values are returned.

Inquire Vector List (`%*h<mask>^`)

This command returns vector list information. The vector list is specified using a separate command, `%*h<<vector list>>s`.

Inquire Picked Object (E*i<mask>^)

This command returns information on the currently picked object. If an object is not picked (pick=0), then returned items 1 and 2 may not be valid, and items 3-5 are not valid.

Inquire Graphics Device Configuration (E*q<mask>^)

This command returns configuration information for the current graphics device. Refer to Section 7, Input/Output Operations, for additional information.

Inquire Raster Hardcopy Format (E*u<mask>, <pen>^)

This command returns raster hardcopy format information. If a pen is not selected, pen#15 will be assumed. If the <pen #> is not in the range 0-15, the command will be ignored. If the number of parameters is not 1 or 2, the command will also be ignored.

Inquire Graphics Color (E*v<mask>, <pen>, <palette>^)

This command returns information on pens and palettes. If a palette number is not included, the currently selected palette is assumed. If a pen number is not included, then pen #15 is assumed. If the <mask> = 0, then a null string is returned. If the palette does not exist and mask bits 4 or 5 = 1, then dummy values are returned. If the number of parameters is not in the range 1-3 the command is ignored.

Inquire Picture File (E*w<mask>^)

This command returns picture file information.

Table 8-2. Graphics Inquiry Data Formats

```

E*e<mask>, <view#>^          Inquire view
  <mask>  ={-32768, , -1, 1, , 32767}
  <view#> ={0, , 127}, {128, , 255}

```

bit	format	meaning
----	-----	-----
0	n	view exits (0/1)
1	n	view is active (0/1)
2	n	view is highlighted
3	+nnnnn	highlight pen (valid only if highlighted)
4	+nnnnn	background color
5	+nnnnn,+nnnnn,+nnnnn,+nnnnn	viewport
6	+nnnnn,+nnnnn,+nnnnn,+nnnnn	window
7	+nnn.nnnn,+nnn.nnnn	zoom factor
8	+nnnnn,+nnnnn	zoom center
9	+nnnnn	active view number

Table 8-2. Graphics Inquiry Data Formats

Ⓔ*g<mask>^		Inquire object	
{-32768,,-1,1,,32767}			
bit	format	meaning	
---	-----	-----	
0	n	object attribute list name exists (0/1)	
1	n	object is highlighted (0/1)	
2	+nnnnn	pick priority	
3	+nnnnn	visibility	
4	+nnnnn	write mask	
5	+nnnnn,+nnnnn	transformation center	
6	+nnn.nnnn,+nnn.nnnn	scale	
7	+nnn.nnnn,+nnn.nnnn	rotation (run/rise format)	
8	+nnnnn,+nnnnn	translation	
9	:12 Characters	object attribute list name	
10	:12 Characters	vector list name	
Ⓔ*h<mask>^		Inquire vector list	
{-32768,,1,1,,32767}			
bit	format	meaning	
---	-----	-----	
0	n	vector list name exists and is not null (0/1)	
1	n	vector list is referenced by object attributes (0/1)	
Ⓔ*i<mask>^		Inquire pick	
{-32768,,32767}			
bit	format	meaning	
---	-----	-----	
0	n	valid pick item: pick success (0/1)	
1	+nnnnn,+nnnnn	pick aperture coordinates, x & y	
2	+nnnnn,+nnnnn	pick aperture size in device coordinates, x & y	
3	+nnnnn	pick ID	
4	12 Characters	object attribute list name	
5	12 Characters	vector list name	
Ⓔ*q<mask>^		Graphics device configuration inquiry	
{-32768,,-1,1,,32767}			
	Bit	Format	Meaning
	---	-----	-----
	0	n	Device configuration menu is locked (unlock with "Ⓔg0L")
	1	n	Device type exists
	2	n	Ordinal device index exists
	3	n	Current field type:
			0 Non-existent
			1 Selection field
			2 Integer
			3 Decimal
			4 Integer array
			5 Decimal array
			6 Bounded string

Table 8-2. Graphics Inquiry Data Formats (Continued)

4	*	Current field value:
Type	Format	Meaning
----	-----	-----
0		Non existent
1	+nnnnn	Selection field
2	+nnnnn	Integer
3	+nnnnn.nnnn	Decimal
4	n,+nnnnn,+nnnnn,....	n*integer array
5	n,+nnnnn.nnnn, +nnnnn.nnnn,....	n*decimal array
6	Character string	bounded string of length<=80 chars.

^t*s<parameter>^ Graphics Status (see description later
in this section)

^t*u<mask>,<pen>^ Raster hardcopy formats inquiry
Default: pen #15

<mask> {-32768,,-1,1,,32767}
<pen> {0,,15}

Bit	Format	Meaning
---	-----	-----
0	+nnnnn	Auto-half-tone-tracking
1	+nnnnn	Last pen selected by ^t*ui command
2	+nnnnn,+nnnnn, +nnnnn,+nnnnn, +nnnnn,+nnnnn, +nnnnn,+nnnnn	Half-tone pattern for pen requested in this ^t*u^ sequence

^t*v<mask>,<pen>,<palette#>^ Inquire palette

<mask>{-32768,,-1,1,,32767}
<pen>={0,,15}
<palette#>={0,,127},{128,255}

bit	format	meaning
---	-----	-----
0	n	palette exits (0/1)
1	n	palette is current palette (0/1)
2	n	method (0/1)
3	+nnnnn	selected palette ID
4	n.nnnn,n.nnnn,n.nnnn	selected pen RGB
5	n.nnnn,n.nnnn,n.nnnn	selected pen HSL

^t*w<mask>^ Inquire picture

<mask>{-32768,,-1,1,,32767}

bit	format	meaning
---	-----	-----
0	+nnnnn	number of objects in picture file
1	+nnnnn	number of vector lists in picture file
2	+nnnnn	number of views in picture file
3	+nnnnn	number of palettes in picture file
4	+nnnnn	number of system graphics fonts
5	+nnnnn	number of user defined graphics fonts

Sample Program

The following program illustrates how to use the graphics inquiry commands to verify color assignment.

```

COLRSTAT
 10 DIM A$(80)
 20 PRINT '27"*v0m1p1a1b1c1I";
 30 PRINT '27"*v0m2p0a0b1c1I";
 40 REM Step Palette (P1)
 50 FOR P1=0 TO 2
 60   PRINT LIN(2);"Palette ";P1;LIN(1)
 70   REM Step Pen (P2)
 80   FOR P2=0 TO 3
 90     PRINT '27"*v16,";P2,P1;"^";
100     LINPUT A$
110     PRINT "Pen ";P2,A$
120   NEXT P2
130 NEXT P1
>run
COLRSTAT

```

Palette 0

Pen 0	0.0000,0.0000,0.0000
Pen 1	1.0000,1.0000,1.0000
Pen 2	0.0000,1.0000,0.0000
Pen 3	1.0000,1.0000,0.0000

Palette 1

Pen 0	0.0000,0.0000,0.0000
Pen 1	1.0000,1.0000,1.0000
Pen 2	0.0000,1.0000,0.0000
Pen 3	1.0000,1.0000,0.0000

Palette 2

Pen 0	0.0000,0.0000,0.0000
Pen 1	0.0000,0.0000,1.0000
Pen 2	0.0000,1.0000,0.0000
Pen 3	1.0000,1.0000,0.0000

GRAPHICS STATUS

In addition to graphics inquiries you can request general graphics status information. All graphics status requests are initiated by sending an `␣ * ␣` followed by a single parameter (0 through 31) followed by a `^`. The single parameter selects the particular status block desired. If an invalid parameter is used, the terminal will simply return its I.D. (see Read Device I.D., parameter = 1).

Graphics Status Request: `Esc * s <parameter> ^`

where: `Esc * s` is the graphics status escape sequence.

`<parameter>` is 0-31 and selects one of 32 blocks of graphics status information.

The graphics status blocks that can be requested are listed in table 8-3 together with the format of the terminal's response. Detailed descriptions of each of the status requests are contained in the following paragraphs.

Table 8-3. Graphics Status Requests

ESC CODE	FUNCTION
<code>Esc*s<parameter>^</code>	Read graphics status
<code><parameter> = 0</code>	Read error code <code><error code></code> <code>+nnnnn</code> bit error description --- 0 wrong parameter type 1 wrong number of parameters 2 parameter out of range 3 illegal parameter 4 undefined character 5 undefined font 6 undefined operand 7 illegal operand 8 arithmetic overflow 9 memory not available 10 graphics hardware error
1	Read device ID <code><device ID></code> 2703A
2	Read pen position <code><X>,<Y>,<pen></code> <code>+nnnnn,+nnnnn,n</code> <code><pen> = 0, pen up, next coordinate is a move</code> <code>1, pen down, next coordinate is a draw</code>
3	Read graphics cursor position <code><X>,<Y></code> <code>+nnnnn,+nnnnn</code>
4	Read graphics cursor position and wait for key <code><X>,<Y>,<keycode></code> <code>+nnnnn,+nnnnn,nnn</code>
5	Read raster size <code><xmin>,<ymin>,<xmax>,<ymax>,<xres>,<yres></code> <code>+00000,+00000,+00511,+00389,00002.,00002.</code>

Table 8-3. Graphics Status Requests (Continued)

```

<parameter> = 6 Read graphics capabilities
<b1>,<b2>,...,<b15>,<b16>
3,16,2,16,1,0,0,1,2,3,2,2,0,0,0
<b1> Clear display
0 no clear
1 paper advance
2 full screen clear
3 partial clear by area
<b2> Number of pens
<b3> Color capability
0 black or white
1 grey levels
2 color
<b4> Colorlevel capability
<b5> Area shading
0 no
1 yes
<b6>-<b7> not used
<b8> Dynamic modification
0 no
1 yes
<b9> Graphics character size
0 fixed
1 integer multiples of base size
2 any size
<b10> Graphics character angles
0 fixed
1 multiples of 90 degrees
2 multiples of 45 degrees
3 any angle
<b11> Graphics character slant
0 fixed
1 45 degrees
2 any angle from -45 to 45 degrees
<b12> Line patterns
0 none
1 predefined only
2 user defined and predefined
<b13-b16> not used

7 Read graphics text status
<xsize>,<ysize>,<lorg>,<angle>,<slant>
+nnnnn,+nnnnn,nn,+nnn.nnnn,+nnn.nnnn
<lorg>
the text origin
<angle>,<slant>
in degrees

8 Read zoom status
<zoom size>,<zoom on/off>
nnn.nnnn,1
<zoom size>
the zoom factor
<zoom on/off>
0 = zoom off
1 = zoom on
    
```

Table 8-3. Graphics Status Requests (Continued)

<parameter>	<p>9 Read relocatable origin <X coordinate>,<Y coordinate> +nnnnn,+nnnnn</p> <p>10 Read reset status <reset>,<b1>,<b2>,...,<b6>,<b7> n,0,0,0,0,0,0 <b1>-<b7> not used <reset> 0=no hard reset since last check 1=terminal hard reset since last check</p> <p>11 Read area shading capability <capability>,<xpattern>,<ypattern> 2,8,8 <capability> Area fill restrictions 1 rectangular only 2 closed polygon only <x pattern><y pattern> the size in display units, x & y, of the area shading pattern</p> <p>12 Read graphics modification capabilities <erase capability>,<complement capability> 1,1 <erase capability> Selective erase 0 no 1 yes <complement capability> Complement mode 0 no 1 yes</p> <p>13 Read secondary cursor information <type>,<dx>,<dy>,<units>,<validity> n,+nnnnn,+nnnnn,n,n <type> Cursor type 0 short crosshair 1 full screen crosshair 2 box <dx><dy> the current box cursor size, in current units <units> Cursor definition units 0 display 1 vector list <validity> Cursor validity 0 cursor position and size info is not valid at this time in these units 1 cursor info is valid</p> <p>14 Read graphics auto-redraw flag <flag> n 0=off 1=on</p> <p>15 Read graphics redraw mode <mode> +nnnnn</p>
-------------	--

Table 8-3. Graphics Status Requests (Continued)

<parameter>	=	16	Read graphics output device <device> +nnnnn
		17	Read display output device <display mode> +nnnnn
		18	Read transparent mode flag <flag> n 0=off 1=on
		19	Read graphics save mode <mode> +nnnnn
		20	Read pick ID <ID> +nnnnn
		21	Read current graphics draw mode <mode> +nnnnn
		22	Read current graphics line type <line> +nnnnn
		23	Read current graphics area pattern <area> +nnnnn
		24	Return symbol size <size> +nnn.nnnn
		25	Return graphics symbol mode <mode> +nnnnn
		26	Return graphics symbol units <units> +nnnnn
		27	Return user defined symbol letter <usymb> +nnnnn
		28	Return user defined symbol font <ufont> +nnnnn
		29	Return primary pen # <pen> +nnnnn
		30	Return secondary pen # <pen> +nnnnn
		31	Read graphics free space <freespace> +nnnnnnnn

Read Device Error Code (Parameter=0)

This command tests to see if errors have occurred. The terminal will respond with a 16-bit, two's compliment integer. Each of the 16 bits represents a different error condition.

BIT	MEANING
0	wrong parameter type
1	wrong number of parameters
2	parameter out of range
3	illegal parameter
4	undefined character
5	undefined font
6	undefined operand
7	illegal operand
8	arithmetic overflow
9	memory not available
10	graphics hardware failure

Read Device I.D. (Parameter=1)

When you request a device I.D. the terminal responds with its Hewlett-Packard model number, 2703A.

Device I.D. Request: `␣ * ␣ 1 ^`

The terminal responds: `2703A <terminator>`

Read Current Pen Position (Parameter=2)

The pen position and status are returned as a string of ASCII characters.

Read Pen Position: `␣ * ␣ 2 ^`

The terminal will return the current x,y coordinates of the pen together with the pen state (0=up, 1=down). The coordinates returned are current units.

For example, assume that the pen is at 360, 80, the pen is up, and the terminal is set for the DC1 handshake, with CR as the terminator:

The computer sends: `␣ * ␣ 2 ^ <terminator> DC1`

The terminal responds: `<X>, <Y>, <Pen>, <terminator>`

Read Graphics Cursor Position (Parameter=3)

The graphics cursor position (in current cursor units) is returned as a string of ASCII characters.



Read Graphics Cursor: $\text{t} * \text{s} 3 \wedge$

The terminal responds: $\langle X \rangle, \langle Y \rangle \langle \text{terminator} \rangle$

where: $\langle X \rangle$ - X coordinate
 $\langle Y \rangle$ - Y coordinate

The position bytes are ordered as in the read pen request.

Read Graphics Cursor Position with Wait (Parameter=4)

This request allows the user to position the cursor, then strike a key to return the position. The ASCII decimal code for the key struck is also returned (not the actual character). The code is returned as three digits. For example, striking an uppercase "A" would return "065". Only ASCII character keys will generate a response (i.e. , , etc. are ignored). The only exception is the graphics input (GID) key or tablet stylus. If either of these two input devices is pressed (activated), the ASCII digits "128" will be returned. The graphics cursor is turned on, if not already on. If an escape sequence is received by the terminal after it has received the READ CURSOR with WAIT command and before a key is struck, the READ CURSOR command will be aborted. The new sequence will be executed instead.

Read Graphics Cursor with Wait: $\text{t} * \text{s} 4 \wedge$

The terminal responds: $\langle X \rangle, \langle Y \rangle, \langle \text{key code} \rangle \langle \text{terminator} \rangle$

where: $\langle X \rangle$ - X coordinate
 $\langle Y \rangle$ - Y coordinate
 $\langle \text{key code} \rangle$ - Decimal value of key struck

The position bytes are ordered as in the read pen request.

Read Display Size (Parameter=5)

This request returns the number of displayable units in the X and Y axes. It also returns the number of units per millimeter in the display. This request allows you to scale data for use on graphic devices with varying display areas.

Read Display Size: $\text{t} * \text{s} 5 \wedge$

The terminal responds: $\langle LLX \rangle, \langle LLY \rangle, \langle URX \rangle, \langle URY \rangle, \langle MMX \rangle, \langle MMY \rangle \langle \text{terminator} \rangle$

where: $\langle LLX \rangle, \langle URX \rangle$ - Lower left and upper right x coordinates
 $\langle LLY \rangle, \langle URY \rangle$ - Lower left and upper right y coordinates
 $\langle MMX \rangle, \langle MMY \rangle$ - Units per millimeter in the x and y axes (five digits and a decimal point)

The terminal will always return a fixed response. The lower left corner has coordinates of 0,0. The upper right corner has coordinates of 511, 389. There are approximately 3 units per millimeter in each axis.

Terminal response: +00000,+00000,+00511,+00389, 00003.,00003. <terminator>

Read Device Capabilities (Parameter=6)

The device capabilities request returns a list of graphic and plotting features available in the terminal. This allows you to use one program for a variety of graphic devices. Not all of the features listed are available in the terminal. The absence of a feature is indicated by a 0. If a feature is present, it may be necessary to send an additional request to determine the exact capabilities present. Where multiple response values are possible, the terminal's standard response is shaded.

Device Capability: $\epsilon * \text{ } 6 \wedge$

The terminal responds: <b1>,<b2>,<b3>,<b4>,<b5>,<b6>,<b7>,<b8>,<b9>,<b10>,<b11>,<b12>,<b13>,<b14>,<b15>,<b16><terminator>

where:

- <b1> = Clear Display
 - 0 = no clear
 - 1 = paper advance
 - 2 = clear (total erase)
 - 3 = partial clear by area
- <b2> = Number of Pens (16)
- <b3> = Color Capability
 - 0 = black and white
 - 1 = grey levels
 - 2 = color
- <b4> = Number of color levels
- <b5> = Area Shading
 - 0 = no
 - 1 = yes (see Read Area Shading Capability)
- <b6>,<b7> = Not Used (0,0)
- <b8> = Dynamic Modification
 - 0 = no
 - 1 = yes (see Read Modification Capability)
- <b9> = Graphics Character Size
 - 0 = fixed
 - 1 = integer multiples of the basic cell size
 - 2 = any size

where:

<b10> = Graphics Character Angles
0 = fixed
1 = multiples of 90 degrees
2 = multiples of 45 degrees
3 = any angle (-45 to +45 degrees)

<b11> = Graphics Character Slant
0 = fixed
1 = 45 degrees
2 = any angle (-45 to +45 degrees)

<b12> = Dot-Dash Line Patterns
0 = none
1 = predefined only
2 = user defined and predefined

<b13>-<b16> = Not Used (0,0,0,0)

Read Graphics Text Status (Parameter=7)

The terminal returns the current text size, orientation, slant, and type of justification. Refer to section 2, Drawing Fundamentals, for a description of graphics text.

Read Graphics Text: $\text{F} * \text{s} 7 \wedge$

The terminal returns: **<X size>**,**<Y size>**,**<origin>**, **<angle>**,**<slant>****<terminator>**

where: **<X size>** = dimension of the character cell
<Y size> = Y dimension of the character cell
<origin> = Relative position of text to cursor (see the text origin command)
<angle> = Text angle in degrees
<slant> = Text slant

Read Zoom Status (Parameter=8)

This request returns the terminal's zoom setting.

Read Zoom Status: $\text{F} * \text{s} 8 \wedge$

The terminal returns: **<zoom size>**,**<zoom on/off>****<terminator>**

where: **<zoom size>** = Zoom setting
<zoom on/off> = 0 for off, 1 for on

Read Relocatable Origin (Parameter=9)

The position of the relocatable origin is returned as x and y coordinates.

Read Relocatable Origin: $\text{t} * \text{s} 9 \wedge$

The terminal returns: $\langle X \text{ coordinate} \rangle, \langle Y \text{ coordinate} \rangle \langle \text{terminator} \rangle$



Read Reset Status (Parameter=10)

You can determine whether or not the terminal has executed a full reset (or Power On) since the last time reset status was checked. This will tell you whether or not you need to reestablish terminal settings or images before resuming terminal functions. An additional seven bytes are returned but are not used. The state of $\langle \text{reset} \rangle$ flag is set to "0" following this request.

Read Reset Status: $\text{t} * \text{s} 10 \wedge$

The terminal returns: $\langle \text{reset} \rangle, \langle b1 \rangle, \langle b2 \rangle, \langle b3 \rangle, \langle b4 \rangle, \langle b5 \rangle, \langle b6 \rangle, \langle b7 \rangle \langle \text{terminator} \rangle$

where: $\langle \text{reset} \rangle = 0$ (no reset since last check)
 $\langle \text{reset} \rangle = 1$ (terminal has been reset)
 $\langle b1 \rangle - \langle b7 \rangle =$ Not used

Read Area Shading Capability (Parameter=11)

The area shading capability of the terminal can be read. These are fixed.

Read Area Shading: $\text{t} * \text{s} 11 \wedge$

The terminal will always respond: $2, 8, 8 \langle \text{terminator} \rangle$

The "2" indicates that the area shaded must be a closed polygon. (A "1" would have meant only rectangular area shading was allowed.) The first "8" indicates that the shading pattern is 8 units wide. The second "8" indicates that the pattern is 8 units high.

Read Graphics Modification Capabilities (Parameter=12)

You can read the terminal's dynamic graphics capabilities. This is the ability to the terminal to change selected portions of the display (clearing and complimenting).

Read Graphics Modification: $\text{t} * \text{s} 12 \wedge$

The terminal will always respond: $1, 1 \langle \text{terminator} \rangle$

These two bytes indicate that the terminal has selective erase and compliment capabilities.

Read Secondary Cursor Status (Parameter = 13)

This command returns additional cursor information.

Read Secondary Cursor Status: `^ * s 13 ^`

The terminal returns: `<type>`, `<xsize>`, `<ysize>`, `<current units>`, and `<error state>`

where:

- `<type>` - 0 short cross hair
1 full screen cross hair
2 box
- `<x size>`, `<y size>` - cursor size in current units
- `<current units>` - 0 display
1 virtual
- `<error>` - 0 no error
1 cursor data not valid

Status Request (Parameters = 14 - 30)

Refer to table 8-3.

Read Graphics Freespace (Parameter = 31)

Returns the number of bytes of graphics memory remaining.

Any Other Parameter

Any other parameter which has not been assigned causes the terminal I.D. to be returned.

`2703A <terminator>`

Sample Program

The following program illustrates how to request information from the picture file. The graphics cursor is used to select the desired object on the terminal screen. When an object is selected, the program returns information about the object.

PICTLOOK

```

10 DIM X$(20),Y$(122),Z$(65),O$(12)
20 PRINT "Position the graphics cursor and pick an object."
30 PRINT "Type the letter x to stop"
40 PRINT '27"*d0u10 10X";
50 PRINT '27"*s4^";
60 LINPUT X$
70 IF X$(15)="120" THEN GOTO 340
80 PRINT '27"*1P";
90 PRINT '27"*163^";
100 LINPUT Z$
110 IF Z$(1,1)="1" THEN GOTO 150
120 PRINT '7"Unsuccessful pick, try again"
130 PRINT
140 GOTO 50
150 O$=Z$(39)
160 PRINT '27"*g("&O$+">A";
170 PRINT '27"*g2047^";
180 LINPUT Y$
190 PRINT "The object name is ";Y$(95,106)
200 PRINT "The vector list name is ";Y$(110,121)
210 PRINT
220 IF Y$(3,3)="0" THEN GOTO 240
230 PRINT "The object is highlighted."
240 PRINT "The detectability is ";Y$(5,10)
250 PRINT "Visibility is ";Y$(12,17)
260 PRINT "The write mask is ";Y$(19,24)
270 PRINT "The transformation center is ("&Y$(26,38);")"
280 PRINT "The scaling factor is ("&Y$(40,58);")"
290 PRINT "Rotation (run,rise) is ("&Y$(60,78);")"
300 PRINT "Translation is ("&Y$(80,92);")"
310 PRINT
320 PRINT
330 GOTO 50
340 END

```

RUN

```

PICTLOOK
Position the graphics cursor and pick an object.
Type the letter x to stop
Unsuccessful pick, try again

The object name is 07
The vector list name is V7

The detectability is +00001
Visibility is +00103
The write mask is +00015
The transformation center is (+00000,+00000)
The scaling factor is (+001.0000,+001.0000)
Rotation (run,rise) is (+001.0000,+000.0000)
Translation is (+00864,+00000)
>

```


Application Notes _____ A

INTRODUCTION

This appendix contains specialized programming examples, and lists the differences between the HP2647/48 and this terminal's implementation of graphics escape sequences.

ANIMATION

Each terminal is shipped with a demonstration disc which contains an animated example — a dragon flying over a pool of water. This appendix provides the programming for that demonstration.

The dragon demonstration requires that both the primary and auxiliary raster buffers be present in the terminal. The demonstration is accomplished in the following steps:

1. The auxiliary raster buffer is loaded with the background picture of sky, castle, and water. In this case, binary image data is loaded into the buffer.
2. The picture file elements (palette, vector lists containing the nine dragon positions, object attributes, and view) are downloaded into vector memory.
3. Output devices, display modes, and read/write masks are set. The dragon is drawn in the foreground planes (the primary raster). The animated effect is achieved by setting up a pseudo double buffer mode within the primary raster — only two planes are visible at a time. While one position of the dragon is displayed, a new position is drawn in hidden raster planes, etc.
4. Each position of the dragon is created as an object. The objects are drawn and cleared while alternating the read/write masks.

NOTE: The demo is initialized using the `COPY <file> :DISPLAY` command. Although the disc contains both ASCII and Binary information, the data is automatically routed to the proper destination (Picture File or Raster).

Application Notes

Step 1. Load the auxiliary raster buffer with the background image.

- a. Enable the auxiliary raster as the output device and set the display mode so that this raster is visible.

$\text{\*e2x2y

- b. Load the auxiliary raster with binary image data. (See the program listing for a breakdown of the raster transfer sequences.)

Step 2. Enable read/write masks.

- a. Select 8-bit write mask mode:

$\text{\*c1X

- b. Select 8-bit read mask mode:

$\text{\*c4Y

- c. Set up read mask:

$\text{\$}^*d252V = \begin{array}{cc} \text{aux.} & \text{prim.} \\ 1111 & 1100 \end{array}$

- d. Set up write mask:

$\text{\$}^*deW = \begin{array}{cc} \text{aux.} & \text{prim.} \\ 0000 & 0011 \end{array}$

- e. Clear hidden planes to pen 0:

$\text{\*dA

- f. Draw object

- g. Alternate read/write masks:

Read Mask: $\text{\$}^*d243V = 1111 \quad 0011$

Write Mask: $\text{\$}^*d12V = 0000 \quad 1100$

- h. Clear hidden planes to pen 0 and draw next object

- i. Goto c.

Step 3. Set up the palette.

The dragon may be drawn in 4 colors, because only two planes are used to display a position at a time. Also, the color palette must be created such that colors will not change when the read/write masks are alternated.

The four possible states of the two planes are:

```
00
01
10
11
```

Each pen chosen for the dragon must have the same value regardless of which write mask 3 (0011) or 12 (1100) is enabled. When the terminal is enabled with write mask 3, the possible pens for the dragon are:

```
0000
0001
0010
0011
```

When the terminal is enable with write mask 12, the possible pens for the dragon are:

```
0000
0100
1000
1100
```

Therefore, at any time, the colors for the dragon may be 0,1,2, and 3. This is achieved by selecting pens 0,5,10, and 15 to define the dragon.

```
0000 = 0
0101 = 5
1010 = 10
1111 = 15
```

Therefore,

Pen Number	Value with Mask=3	Value with Mask=12
0000	00 = 0	00 = 0
0101	01 = 1	01 = 1
1010	10 = 2	10 = 2
1111	11 = 3	11 = 3

Step 4. Download the vector lists (dragon positions).



Figure A-1. The Nine Dragon Positions

Table A-1. Listing For Dragon Demonstration

```

*e2x2Y

*r512s390t4U*rH*rA
.5334b0.5334c7i0a0c8i1a0.5334c9i0a0b10i0.2667b0.8c11i1a0b0c12i0.2667a0.2667b1c13i
0.8a0.8b0.8c14i1a1b1c15iL

*b0V
*b0V
*b0V
*b0W
.
.
.
*rB

*e1X

*v0m64pr0i.25b1i4i.5b2i8i1a0b3i12i1b1c15i
*v.25a.5b.75c5i.5a0b0c6i.5b.5c7i1a9i0a0b10i.25b.75c11i.75a.5c13i
.25b.75c11i.75a.5c13i.75b.75c14iL

*hA*m5X*psa2012 176 904 -872 696 -880 804 -940 -916 -2276 -516 -2292
-212 -2496 2392 -604 2700 -660 3220 -484 3684 -472 4232 -220 ta96 16 300
312 252 24 a412 16 572 176 592 -44 a712 -16 872 156 888 -44 a1076 -16
1232 184 1268 -20 a1400 28 1580 232 1620 8 a1808 132 1960 408 1968 188
a2112 136 2324 356 2316 140 a2480 120 2808 368 2744 56 a3068 136 3356
356 3248 92 a3572 36 3852 140 3768 -108 a4024 -136 4256 -20 4148 -236
a4468 -256 4696 -64 4648 -280 a4988 -260 5144 -84 5136 -264 a5428 -248
5632 -72 5604 -232Z*m10X*psa-64 12 -104 88 284 148 392 80 1160 72 1572
112 1932 280 2124 208 3264 240 4016 -68 4876 -192 6152 -136 6056 -52
6720 -108 5992 -344 6144 -192 5156 -324 4116 -344 3808 -304 3536 -328
3772 -416 3848 -704 3936 -688 4032 -840 4124 -736 4116 -676 4164 -784
4064 -944 3904 -936 3668 -568 3368 -672 3104 -540 2936 -340 3000 -216
2952 -132 2732 -276 2536 -544 2572 -1044 2772 -1696 3420 -2756 3104
-2724 2616 -2956 2400 -3288 2140 -3204 1212 -3284 840 -3472 660 -3220
-28 -2960 -428 -2944 -1064 -3084 328 -1764 1116 -1140 964 -1120 1244
-1024 1672 -400 1600 -252 1468 -156 584 -172 460 -224 336 -160 292 -196
52 -228 -24 -172 -204 -200 -268 -272 -384 -232 -320 -48 -288 -76 ta828
-920 704 -1544 a1096 -1296 912 -864 1372 -1104Z*m5X*pa848 -3436 836
-1840 1164 -1108 a2336 -3156 1420 -1980 1240 -1064 1868 -1868 3256 -2664Z
*m15X*pa-8 0 72 -8Z*h<d1>c
.
.
.
*m5X*pa-140 104 -240 336 -264 168 -428 116 -600 380 -632
164 -728 208 -940 412 -888 132 -1376 376 -1360 160sa-2272 -156 -2440
-512 -2428 -796 -2320 -988 -2144 -1080 -1600 -1012 -1260 -848 -1088 -632
-996 -420 -2092 192 Z*m10X*psa-88 160 -180 224 -388 172 -628 256 -1008
292 -1500 236 -1848 108 -1964 300 -1924 756 -1760 1220 -1472 1544 -1676
1588 -1892 1884 -1884 2136 -1844 2264 -2052 2404 -2148 2548 -2216 3136
-2156 3312 -2412 3284 -2780 3460 -3140 3776 -3348 4136 -3356 2992 -3196
1912 -3088 1484 -3120 1492 -2804 760 -2364 12 -2356 -220 -2436 -320
-2540 -328 -2556 -156 -2752 -268 -2780 -496 -2684 -408 -2692 -492 -2612
-408 -2560 -488 -2408 -436 -2272 -284 -2072 -176 -1864 -132 -1768 -236
-1752 -436 -1840 -660 -1936 -976 -2204 -1088 -2276 -1256 -2232 -1376
-2196 -1300 -2148 -1356 -2040 -1128 -1736 -1108 -1548 -880 -1412 -708
-1272 -616 -1104 -600 -1424 -940 -1472 -1088 -1752 -1244 -1744 -1172
-1964 -1332 -2008 -1484 -1876 -1404 -1864 -1528 -1772 -1396 -1600 -1352
-1564 -1296 -1252 -1132 -1012 -1092 -1004 -996 -900 -912 -492 -1040 -192
-1260 208 -1336 1020 -1232 940 -1368 1628 -1048 968 -1124 1008 -1212 528
-1244 104 -1192 -232 -1016 -396 -748 -540 -668 336 -816 320 -748 1104
-564 2040 -580 2832 -156 3496 56 3008 160 2644 376 2432 552 2136 536

```

Table A-1. Listing For Dragon Demonstration (Continued)

```

-140 -204 -156 -64 -144 104 -232 124 -252 280 -204 256 -88 80 -16 72 96
-92 156a572 488 484 -148 268 -676a1672 680 1080 64 372 -716 1288 -156
2396 516 a-1208 -632 260 -764a-632 -200 -836 -252 -952 -220a-364 -784
-216 -848 -296 -900a-196 -1024 -28 -1036 -112 -1108a104 -1204 256 -1148
224 -1228a464 -1236 584 -1172 564 -1236a784 -1228 936 -1164 888 -1240Z
*m15X*pa24 -16 -40 8Z*h<d9>c

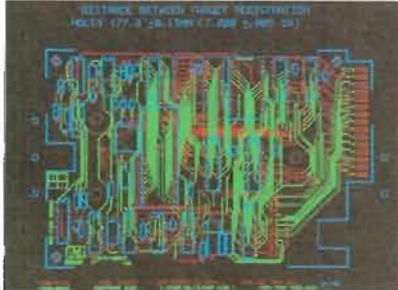
*e0a0m1x4Y*d252v3wA*gb<od>a<d1>c500,200t,.02,.02s1,0R*eR
*d243v12wA*g<d2>o495,200t*eR
*d252v3wA *g<d3>o490,200t*eR
*d243v12wA*g<d6>o315,200t*eR
.
.
.
*d252v3wA *g<d1>o180,200t*eR

*d243v12wA*g<d2>o163,203t.9397,-.342r*eR
*d252v3wA *g<d3>o148,212t.766,-.6428r*eR
*d243v12wA*g<d4>o137,225t.5,-.866r*eR
.
.
.
*d252v3wA *g<d7>o229,241t.1736,.9848r*eR
    
```

THE LAYERED LOOK

A picture with overlays (anatomy drawings, printed circuit boards, etc.) can be segmented into layers on the terminal corresponding to raster planes. Each layer can then be displayed separately by setting the 8-bit display mode and read masks. A function key could be dedicated to each layer:

- f1 = Fc*d1V
- f2 = Fc*d2V
- f3 = Fc*d4V
- f4 = Fc*d8V
- f5 = Fc*d3V
- f6 = Fc*d5V
- f7 = Fc*d7V
- f8 = Fc*d15V



Note, however, that only four colors will be displayed since each pixel can be one of 4 states

- 0000 - 0
- 0010 - 1
- 0100 - 2
- 1000 - 3
- 0000 - 0
- 0010 - 1
- 0100 - 2
- 1000 - 3

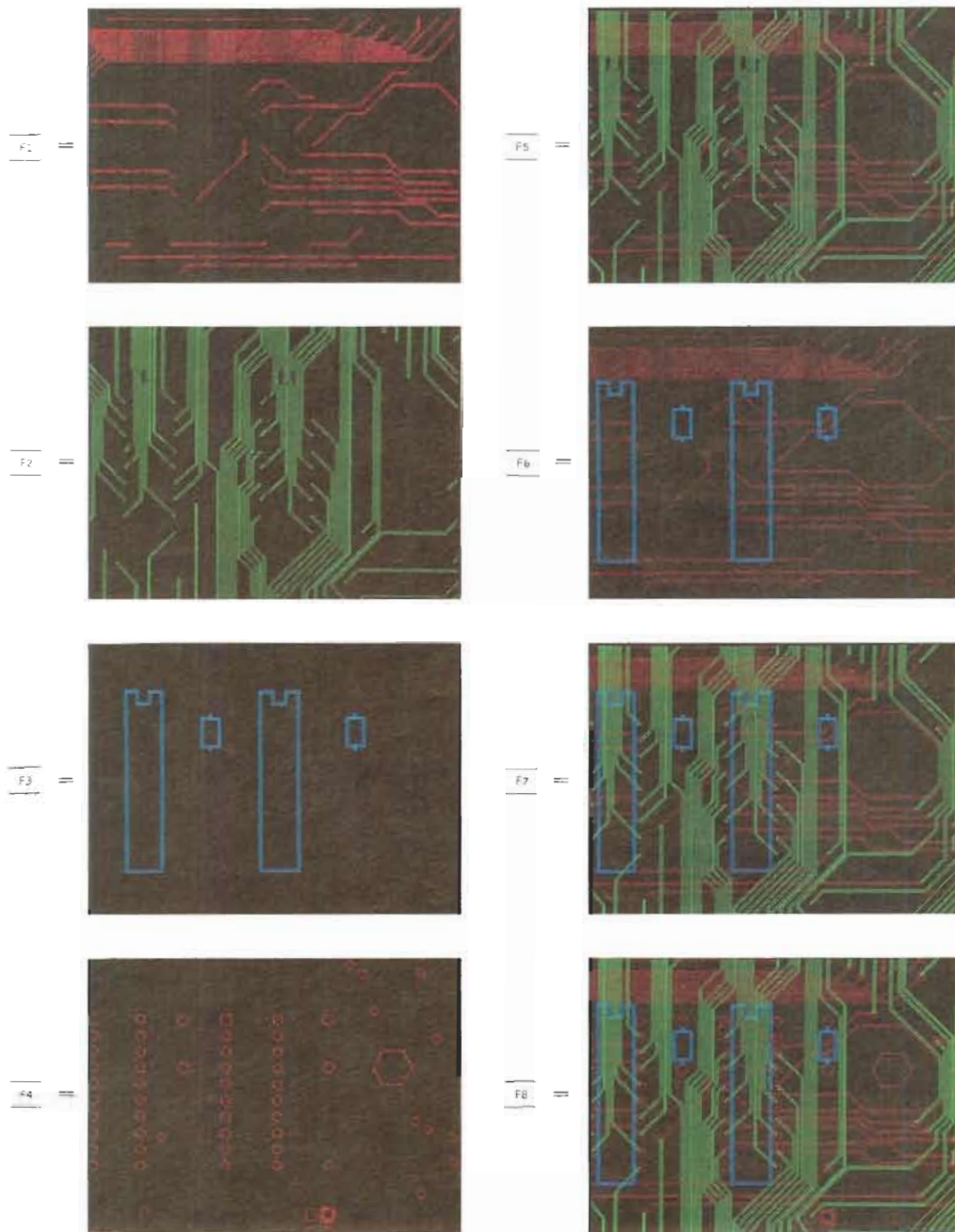


Figure A-2. Printed Circuit Board Example

READING FROM THE PICTURE FILE

The program shown in figure A-3 demonstrates how data in the picture file can be read by a program running on a host computer.

```

EXA
10 REM *****
20 REM
30 REM *          GRAPHIO
40 REM
50 REM * This program can be used to store the current picture
60 REM * file of an HP2700 in a host CPU. The picture file is
70 REM * stored as a BASIC program which may be run to recreate
80 REM * the picture file. The program accomplishes this in
90 REM * three steps:
100 REM * 1) Determine what part of the picture file is to be
110 REM * saved
120 REM * 2) Loop until last line is read from terminal
130 REM * - read a line of the picture file into I$
140 REM * - move I$ to the output buffer (O$)
150 REM * - output characters up to the first esc char
160 REM * beyond the beginning of the buffer until no
170 REM * escape char. is found beyond the beginning of
180 REM * the buffer
190 REM * - print 78 char blocks of the output buffer
200 REM * until the output buffer is < 78 characters
210 REM * - continue loop
220 REM * 3) Call in the new basic program
230 REM * The BASIC program which is created by this procedure
240 REM * is written into the temporary file "FDATA". This
250 REM * procedure forces each new escape sequence from the
260 REM * HP2700 to begin a new PRINT statement in the finished
270 REM * program. This is to make the finished picture program
280 REM * easier to modify. each PRINT statement in the final
290 REM * program will contain 78 or fewer characters.
300 REM *
310 REM Variables-
320 REM A : dummy variable for function return codes
330 REM A$: keyboard response input string
340 REM E : pointer to char in front of esc char in output
350 REM buffer
360 REM I$: input buffer
370 REM O$: output buffer
380 REM Z1: line number counter in output file
390 REM
400 REM *****
410 DIM I$(80),O$(255)
420 DIM A$(12)
430 DEF FNP(Z)
440 REM *****
450 REM * This is a function to format the output buffer and
460 REM * add a line to the output file we are building.
470 REM * It prints the first Z characters of the output buffer
480 REM * then resets the output buffer to eliminate the printed
490 REM * characters. Z1 is the outputted line number.
500 REM *****
510 PRINT #1;Z1,"print ctl(208)";34;O$(1,Z1);34;"
520 IF Z>80 THEN DO
530 PRINT Z
540 PRINT '27"g"
550 STOP
560 DOEND

```

Figure A-3. GraphIO Program

```

570 IF LEN(O*)>255 THEN DO
580   PRINT "o$ too long"
590   PRINT '27"g"
600   STOP
610 DOEND
620 O$=O$[Z+1,LEN(O*)]
630 REM * bump the line number counter and print it out
640 Z1=Z1+10
650 PRINT "line ";Z1
660 RETURN 0
670 FNEND
680 REM *****
690 REM     THE PROCEDURE
700 REM
710 REM *****
720 REM     initialize the line # counter
730 REM
740 Z1=10
750 REM *****
760 REM     open output file
770 REM
780 FILES *
790 SYSTEM A,"purge fdata"
800 SYSTEM B,"build fdata;rec=-132,,f,ascii"
810 ASSIGN "fdata",1,A
820 IF A<>0 OR B<>0 THEN DO
830   REM * return code(s) were bad; print error and abort
840   PRINT "return codes: create ";B;" assign ";A
850   GOTO 1560
860 DOEND
870 REM *****
880 REM     program description
890 REM
900 PRINT "This program will read the entire picture file or just a given"
910 PRINT
920 PRINT "object and its associated vector list. This information will"
930 PRINT
940 PRINT "be converted into a basic program you can use to reconstruct the"
950 PRINT
960 PRINT "file or object."
970 PRINT
980 REM *****
990 REM *     set up to begin transfer
1000 REM
1010 INPUT "Name the object to be saved (with its vector list) or 'ALL':",A$
1020 IF UPS$(A$)="ALL" THEN DO
1030   REM     set pic save mask for "ALL" response
1040   REM           2: current color palette
1050   REM           8: active views
1060   REM           32: user defined text fonts
1070   REM           128: all hidden definitions
1080   REM           256: all objects
1090   REM           1024: creation object and displayed definition
1100   A=2+8+32+128+256+1024
1110   PRINT '27"*w";A;"Y";
1120 DOEND
1130 ELSE DO
1140   PRINT '27"*g<";A$;">A";
1150   PRINT '27"*w512Y"
1160 DOEND
1170 INPUT "Enter ESC ; to turn off echo",I$
1180 PRINT '27",cCopy All :picture to :datacomm"
1190 REM *****

```



Figure A-3. GraphIO Program (Continued)

Application Notes

```
1200 REM      process picture file
1210 REM
1220 PRINT #1;"scratch"
1230 O$=""
1240 ENTER 2,A,I$
1250 IF A<>-256 THEN DO
1260   REM * an input line was read, add it to the output buffer
1270   O$=O$+I$
1280   E=POS(O$[2,LEN(O$)],'27)
1290   REM * note that E points in front of the esc char in o$
1300   IF E<>0 THEN DO
1310     REM * an esc char. was found in the output buffer beyond
1320     REM * the first character so print the output buffer up to
1330     REM * the escape char and loop until no esc char is found
1340     REM * beyond the first char
1350     A=FNP(E MIN 78)
1360     GOTO 1280
1370   DOEND
1380   REM * no esc char. was found in the output buffer beyond
1390   REM * the first character
1400   REM * if the buffer is >78 characters output until the
1410   REM * buffer is <78 characters
1420   IF LEN(O$)>78 THEN DO
1430     A=FNP(78 MIN LEN(O$))
1440     GOTO 1420
1450   DOEND
1460   REM * go get another input buffer
1470   GOTO 1240
1480 DOEND
1490 ELSE DO
1500   REM * no input line was received
1510   PRINT #1;"list"
1520   PRINT #1;"purge fdata"
1530   PRINT "end of picture file"
1540   GOTO 1560
1550 DOEND
1560 REM *****
1570 REM      clean up
1580 REM
1590 INPUT "Enter ESC : to turn on echo",I$
1600 PRINT "Please wait while your new program is loaded."
1610 PRINT '27"&f2a1k8d10LJUNK   xeq fdata"
1620 PRINT '27"&f1E"
1630 PRINT '27"&jB"
1640 STOP
```

Figure A-3. GraphIO Program (Continued)

HP 2647/48 AND HP 2700 GRAPHICS DIFFERENCES

Display Size

The HP 2700's resolution is 512×390 vs the HP 264X's 720×360 .

Escape Sequences

$\text{\textbackslash}a\dots$

HP 2648A Autoplot escape sequences are not supported on the HP 2700.

$\text{\textbackslash}dg, dh$

Zoom on/off sequences are not implemented by the HP 2700. These sequences are treated as NOP's.

$\text{\textbackslash}ds$

In 264X graphics text mode, alpha and graphics cursor movement keys are unaffected by rotation. In the HP 2700, cursor control rotates according to the current text attributes.

$\text{\textbackslash}l$

Labels which are left justified are not displayed on the HP 2700 as they are built, unlike the 264X.

The maximum length of a label is 250 characters on the HP 2700. This compares to 103 characters for the 264X. This limit applies even for the left justified lables, unlike the 264X.

Graphics characters will not clear the character cell underneath them.

Graphics characters are vector characters, they do not get thicker as they get larger.

The 264X uses JAM2 mode (4) for labels written in NOP mode (0). The HP 2700 uses NOP mode.

$\text{\textbackslash}mb$

Area fills on the HP 2700 do not use the current line type.

$\text{\textbackslash}me, mf$

Rectangular area fills use the current area fill pattern specified by a new escape sequence, versus the current line type as for the 264X.

Application Notes

`ESCmq`

A text justification of zero has a slightly different meaning which may cause the left/right position to be different by about 2 units.

`ESCmr`

This sequence is 264X compatible, and performs new actions specific to the HP 2700.

`ESCpd`

The pen is raised after this sequence.

`ESCre`

The 2647 always returns "1" in bit 0 of the status byte; the HP 2700 uses this bit to report whether or not the terminal has been reset since the last status request.

`ESCg^`

The 264X terminals allow any capital letter to end this sequence as a status request, and will return some kind of status. The HP 2700 will not reply to anything but "↑".

On 264X products, the status request returns angles in degrees. The HP 2700 reports angles in radians.

`ESCt`

Compatibility mode escape sequences are not supported on the HP 2700.

Transformation Equations _____ B

This appendix contains the equations used by the firmware to perform object, viewing, and color transformations.

OBJECT ATTRIBUTE TRANSFORMATIONS

Vector lists are mapped from vector space to virtual space by applying object attributes: transformation center, scaling, rotation, and translation.

- x_o, y_o = original coordinates of the vector list
- x_{tc}, y_{tc} = transformation center
- s_x, s_y = scale factors
- θ = angle of rotation
- T_x, T_y = translation
- x_v, y_v = coordinates of object in virtual space

The transformations in order of occurrence are:

- Transformation Center:
 $x_o = x_o - x_{tc}$
 $y_o = y_o - y_{tc}$
- Scaling:
 $x_s = s_x * x_o$
 $y_s = s_y * y_o$
- Rotation:
 $x_r = \cos(\theta)x_s - \sin(\theta)y_s + x_{tc}$
 $y_r = \sin(\theta)x_s + \cos(\theta)y_s + y_{tc}$
- Translation:
 $x_v = x_r + T_x + x_{tc}$
 $y_v = y_r + T_y + y_{tc}$

The terminal firmware translates the object transformation equations into 3×3 matrices in order to concatenate a sequence of transformations into one arithmetic operation.

Transformation Equations

CASE 1. Assume that the transformation center is 0,0. Then the 3×3 representations of the object transformation are:

- Scaling:
$$[X_s \ Y_s \ 1] = [X_o \ Y_o \ 1] \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

- Rotation:
$$[X_r \ Y_r \ 1] = [X_o \ Y_o \ 1] \begin{vmatrix} \cos & \sin & 0 \\ -\sin & \cos & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

- Translation:
$$[X_t \ Y_t \ 1] = [X_o \ Y_o \ 1] \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{vmatrix}$$

Concatenate the three matrices which contain the scaling, rotation, and translation parameters:

$$X_v \ Y_v \ 1 = [X_o \ Y_o \ 1] \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos & \sin & 0 \\ -\sin & \cos & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{vmatrix} \longrightarrow$$

$$X_v \ Y_v \ 1 = [X_o \ Y_o \ 1] \begin{vmatrix} S_x \cos & S_y \sin & 0 \\ -S_y \sin & S_x \cos & 0 \\ T_x & T_y & 1 \end{vmatrix} \longrightarrow$$

$$X_v = S_x \cos(\theta) X_o - S_y \sin(\theta) Y_o + T_x$$

$$Y_v = S_x \sin(\theta) X_o + S_y \cos(\theta) Y_o + T_y$$

CASE 2. The transformation center $(X_{tc}, Y_{tc}) \neq (0, 0)$.

The transformation matrix is:

$$X_v \ Y_v \ 1 = [X_o \ Y_o \ 1] \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -T_{xc} & -T_{yc} & 1 \end{vmatrix} \begin{vmatrix} S_x \cos & S_y \sin & 0 \\ -S_y \sin & S_x \cos & 0 \\ T_x & T_y & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{xc} & T_{yc} & 1 \end{vmatrix} \longrightarrow$$

$$X_v = S_x \cos(\theta) (X_o - T_{xc}) - S_y \sin(\theta) (Y_o - T_{yc}) + T_x + T_{xc}$$

$$Y_v = S_x \sin(\theta) (X_o - T_{xc}) + S_y \cos(\theta) (Y_o - T_{yc}) + T_y + T_{yc}$$

VIEWING TRANSFORMATIONS

Once the x, y coordinates are in virtual space (X_v, Y_v), the next step is to convert them into screen coordinates (X_s, Y_s). Recall that virtual space has an address range of $-16K, \dots, +16K$. Within this virtual space, one can define a rectangle known as the virtual window. (See Sections 1 and 4.) In addition, one can define a rectangle on the screen where the contents of the virtual window are displayed.

Conversion of virtual units to screen units:

Let:

$W_{xmin}, W_{ymin}, W_{xmax}, W_{ymax}$ = virtual window limits
 $V_{xmin}, V_{ymin}, V_{xmax}, V_{ymax}$ = screen viewport limits
 X_v, Y_v = point within the virtual window

Then:

$$X_s = (X_v - W_{xmin}) * \frac{(V_{xmax} - V_{xmin})}{(W_{xmax} - W_{xmin})} + V_{xmin}$$

$$Y_s = (Y_v - W_{ymin}) * \frac{(V_{ymax} - V_{ymin})}{(W_{ymax} - W_{ymin})} + V_{ymin}$$

The matrix interpretation of the above equations is:

$$X_s \ Y_s \ 1 = [X_v \ Y_v \ 1] \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -W_{xmin} & -W_{ymin} & 1 \end{vmatrix} \begin{vmatrix} Z_{fx} & 0 & 0 \\ 0 & Z_{fy} & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ V_{xmin} & V_{ymin} & 1 \end{vmatrix}$$

where:

$$Z_{fx} = \frac{(V_{xmax} - V_{xmin})}{(W_{xmax} - W_{xmin})} = X \text{ Zoom Factor}$$

$$Z_{fy} = \frac{(V_{ymax} - V_{ymin})}{(W_{ymax} - W_{ymin})} = Y \text{ Zoom Factor}$$

Transformation Equations

Zoom Factor — defines the ratio of the screen viewport and virtual window.

Zoom Center — defines the zoom origin, and is the center of the virtual window.

$$\text{[Equation 1]} \quad X_{zc} \text{ (X zoom center)} = \frac{(W_{x\max} - W_{x\min})}{2} + W_{x\min}$$

$$\text{[Equation 2]} \quad Y_{zc} \text{ (Y zoom center)} = \frac{(V_{y\max} - V_{y\min})}{2} + V_{y\min}$$

At power on the virtual window = screen viewport = (0, 0, 511, 389).

Therefore:

$$Z_{fx} = Z_{fy} = 1, \text{ and,}$$

$$X_{zc}, Y_{zc} = 255, 194$$

Given the values for zoom factors (Z_{fx} , Z_{fy}), zoom centers (X_{zc} , Y_{zc}), and viewport limits ($V_{x\min}$, $V_{y\min}$, $V_{x\max}$, $V_{y\max}$), you can calculate the limits of the virtual window ($W_{x\min}$, $W_{y\min}$, $W_{x\max}$, $W_{y\max}$) since:

$$\text{[Equation 3]} \quad Z_{fx} = \frac{(V_{x\max} - V_{x\min})}{(W_{x\max} - W_{x\min})}$$

$$\text{[Equation 4]} \quad Z_{fy} = \frac{(V_{y\max} - V_{y\min})}{(W_{y\max} - W_{y\min})}$$

and

$$W_{x\min} = X_{zc} - \frac{(W_{x\max} - W_{x\min})}{2} \quad (\text{from [1]})$$

therefore:

$$\text{[Equation 5]} \quad W_{x\min} = X_{zc} - \frac{(V_{x\max} - V_{x\min})}{2 Z_{fx}}$$

(by substituting [3] into [1]).

Similarly:

$$\text{[Equation 6]} \quad W_{ymin} = X_{yc} - \frac{(V_{ymax} - V_{ymin})}{2 Zfy}$$

Since we have the values for W_{xmin} , we can obtain the values for W_{xmax} .

$$\text{[Equation 7]} \quad W_{xmax} = 2 * X_{zc} - W_{xmin} \text{ (from [1])}$$

and

$$\text{[Equation 8]} \quad W_{ymax} = 2 * Y_{zc} - W_{ymin}$$

The following table describes what happens when the screen viewport, virtual window, zoom center, or zoom factor is modified.

Table B-1. View Manipulation Effects

Screen Viewport (Ec * e v)

- virtual window is not affected
- zoom center is not affected
- zoom factor is recalculated using the relationship described in equations [3] and [4]

Virtual Window (Ec * e w)

- screen viewport is not affected
- zoom factor is recalculated using equations [3] and [4]
- zoom center is recalculated using equations [1] and [2]

Zoom Factor (Ec * d i)

- screen viewport is not affected
- zoom center is not affected
- virtual window is recalculated using equations [5], [6], [7], and [8]

Zoom Center (Ec * d j)

- screen viewport is not affected
- zoom factor is not affected
- virtual window is recalculated using equations [5], [6], [7], and [8]

COLOR TRANSFORMATIONS**HSL to RGB**

The terminal uses RGB values to control color operation. When values are input using the HSL notation method, they are converted to RGB values using the following equations:

Given: H, S, L values in the range [0,1]

```

Hue := 6 * H
J   := INTEGER(Hue)
Frac := Hue - J

X   := L*(1-S)
Y   := L*(1-(S*Frac))
Z   := L*(1-(S*(1-Frac)))

CASE J OF
0: {R,G,B} := {L,Z,X}
1: {R,G,B} := {Y,L,X}
2: {R,G,B} := {X,L,Z}
3: {R,G,B} := {X,Y,L}
4: {R,G,B} := {Z,X,L}
5: {R,G,B} := {L,X,Y}

```

RGB to HSL

The terminal uses the following equations to convert RGB values to HSL values:

```

L := MAX(R,G,B)
X := MIN(R,G,B)
S := (L-X)/L
If S=0 then RETURN (Hue is set to previous Hue)

r := (L-R)/(L-X)
g := (L-G)/(L-X)
b := (L-B)/(L-X)

IF R=L THEN IF G=X THEN Hue := 5+b
              ELSE Hue := 1-g
IF G=L THEN IF B=X THEN Hue := 1+r
              ELSE Hue := 3-b
              ELSE IF R=X THEN Hue := 3+g
                    ELSE Hue := 5-r

H := Hue/6

```


Color Key

C

This appendix displays some of the 4096 colors that can be selected using R, G, B color combinations. The colors shown in this appendix are intended only as a guide. Actual colors displayed will vary depending on room lighting and terminal alignment. Hardcopy colors will vary with inks, paper, or the configuration and adjustment of external video devices such as monitors and cameras.

The color charts on the following pages are arranged as blocks of red-green combinations. The blue value for each block is fixed. The labeling is in parts and percentages of color based on 15 parts being equal to 100%. Actual component parts are obtained by dividing parts by 15 to obtain a decimal fraction between 0 and 1.

Additional colors beyond 4096 can be obtained by building "dithered" patterns on the display. This technique combines several pixels to create the effect of additional colors. This will result in the loss of available resolution.

Color Proportions

PARTS	DECIMAL VALUE
0	0.00
1	0.06
2	0.13
3	0.20
4	0.27
5	0.33
6	0.40
7	0.46
8	0.53
9	0.60
10	0.66
11	0.73
12	0.80
13	0.86
14	0.93
15	1.00

Color Key

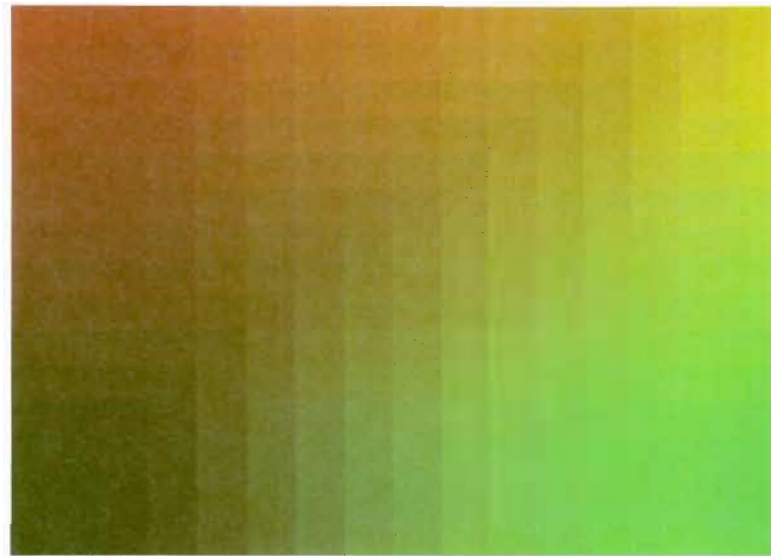
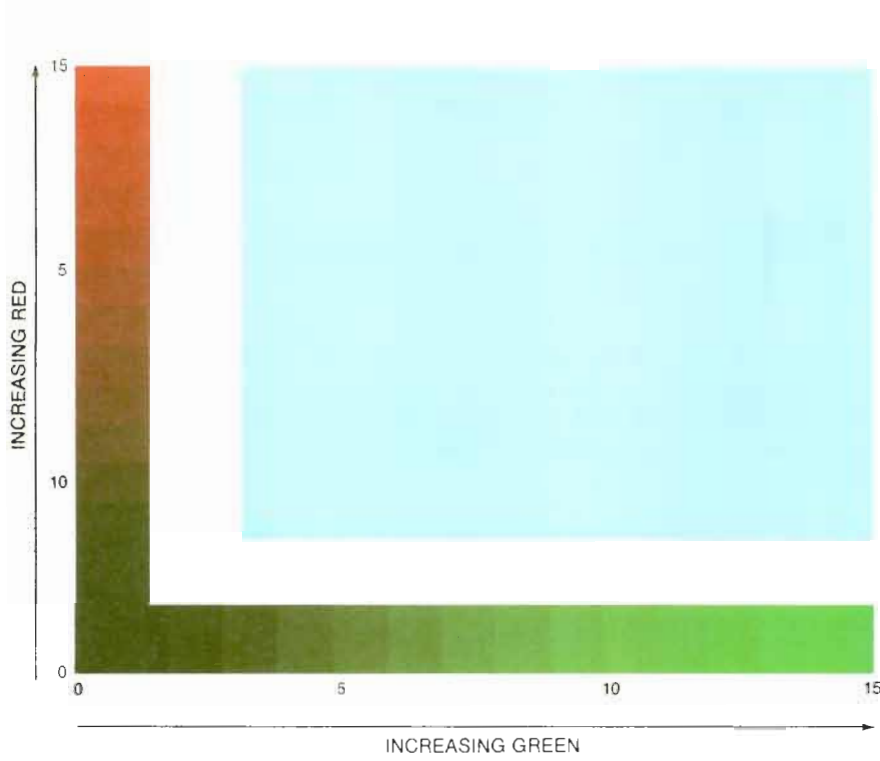


Figure C-1. Color Keys

C-2

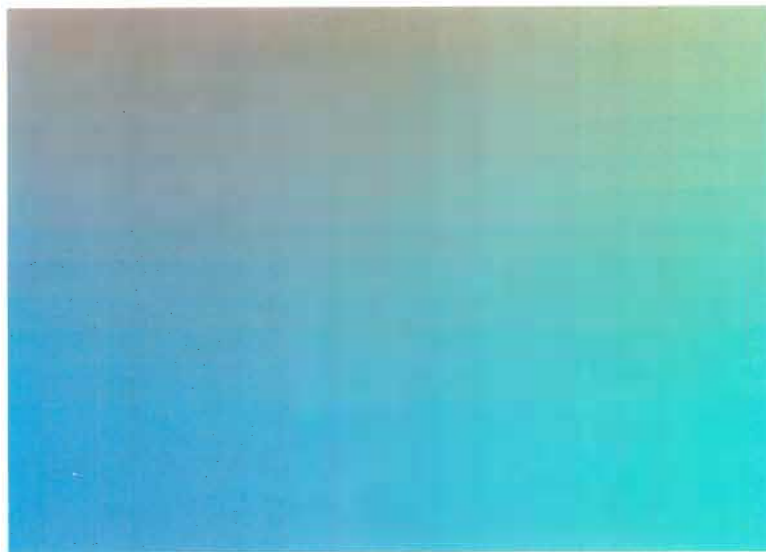
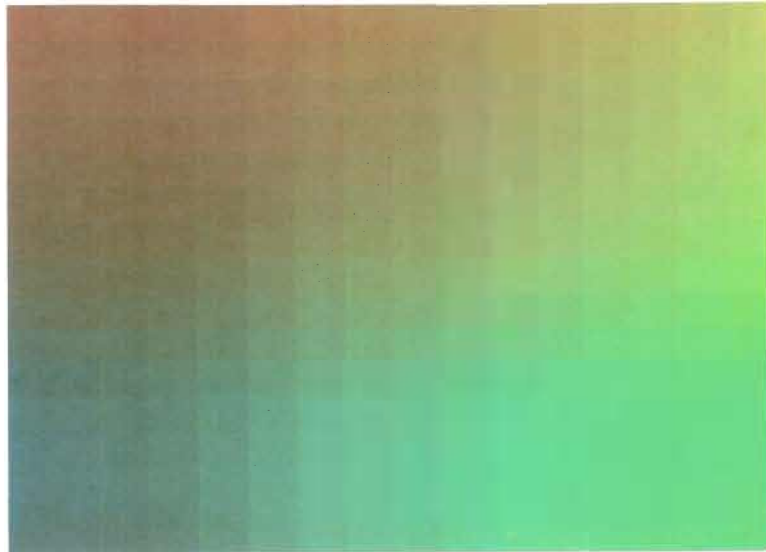


Figure C-1. Color Keys (continued)

Escape Sequence Directory _____ D

RASTER DUMP — Section 6

ESCAPE CODE	FUNCTION
⌘•b<# of data bytes>p	Read binary block into datacomm (0,,255)
⌘•b<# of data bytes>q	Read binary block into datacomm; set up next plane (0,,255)
⌘•b<# of data bytes>r	Read binary block into datacomm; set up next line (0,,255)
⌘•b<# of data bytes>u	Write into raster memory (0,,255)
⌘•b<# of data bytes>v	Write into raster memory; set up next plane (0,,255)
⌘•b<# of data bytes>w	Write into raster memory; set up next line (0,,255)
⌘•b<x>x	Set temporary X offset (0,,9999)
⌘•b<dy>y	Set temporary Y offset (0,,9999)
⌘•b<dz>z	Set temporary Z offset (0,,9999)

DISPLAY CONTROL — Section 2

ESCAPE CODE	FUNCTION
⌘•d<pen#>a	Set raster memory/default=0 (-32768,,32767)
⌘•d<pen#>b	Set raster memory/default=15 (-32768,,32767)

 DISPLAY CONTROL

ESCAPE CODE	FUNCTION
<code>Esc*dc</code>	Turn on graphics display
<code>Esc*dd</code>	Turn off graphics display
<code>Esc*de</code>	Turn on alpha display
<code>Esc*df</code>	Turn off alpha display
<code>Esc*d<x factor>,<y factor>i</code>	Set zoom size (.001,,63.999)
<code>Esc*d<x>,<y>j</code>	Set zoom center (-16383,,16383)
<code>Esc*d<type>k</code>	Turn on graphics cursor, and specify type: 0 = short crosshair, "+" 1 = long crosshair, full screen 2 = box cursor
<code>Esc*dl</code>	Turn off rubberband line; turn off graphics cursor
<code>Esc*dm</code>	Turn on rubberband line; turn on graphics cursor
<code>Esc*dn</code>	Turn off rubberband line
<code>Esc*d<x>,<y>o</code>	Move graphics cursor absolute (0,,511) "display" cursor (-16383,,16383) "virtual" cursor
<code>Esc*d<dx>,<dy>p</code>	Move graphics cursor relative (-511,,511) "display" cursor (-32767,,32767) "virtual" cursor
<code>Esc*d<type>q</code>	Turn on alphanumeric cursor, and select type: 0 = underline 1 = blob
<code>Esc*dr</code>	Turn off alphanumeric cursor
<code>Esc*ds</code>	Turn on graphics text mode
<code>Esc*dt</code>	Turn off graphics text mode
<code>Esc*d<mode>u</code>	Define (display/virtual) graphics cursor: 0 = "display" cursor 1 = "virtual" cursor

ESCAPE CODE	FUNCTION
<code>␣*d<read mask>v</code>	Set display read mask Bits set—raster planes on Bits clear—raster planes off (-32768,,32767)
<code>␣*d<write mask>w</code>	Set display write mask (-32768,,32767)
<code>␣*d<xsize>,<ysize>x</code>	Set box cursor size (1,,32767)

IMAGE CONTROL — Section 4

ESCAPE CODE	FUNCTION										
<code>␣*e<mode>a</code>	Set auto redraw 0 = off 1 = on										
<code>␣*e<pen#>b</code>	Set viewport background color (-32768,,32767)										
<code>␣*e<view#>d</code>	Delete view from memory (0,,255)										
<code>␣*e<pen#>h</code>	Highlight view (-32768,,32767)										
<code>␣*eh</code>	Turn off highlighting										
<code>␣*e<view#>i</code>	Activate/create view (0,,255)										
<code>␣*e1</code>	Delete all inactive views										
<code>␣*e<mode>m</code>	Set redraw mode (-32768,,32767)										
	<table> <thead> <tr> <th>bit</th> <th>meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>clear screen to pen zero</td> </tr> <tr> <td>1</td> <td>set viewport to background color</td> </tr> <tr> <td>2</td> <td>double buffer mode</td> </tr> <tr> <td>3</td> <td>halftone mode</td> </tr> </tbody> </table>	bit	meaning	0	clear screen to pen zero	1	set viewport to background color	2	double buffer mode	3	halftone mode
bit	meaning										
0	clear screen to pen zero										
1	set viewport to background color										
2	double buffer mode										
3	halftone mode										
<code>␣*eq</code>	Reset active view to default values: viewport = 0,0,511,389 window = 0,0,511,389 highlighting = off viewport background color = 0 zoom factor = 1										

IMAGE CONTROL

ESCAPE CODE	FUNCTION																					
<code>Esc er</code>	Redraw view																					
<code>Esc e<x1>,<y1>,<xu>,<yu>v</code>	Set viewport limits (0,,511)																					
<code>Esc e<x1>,<y1>,<xu>,<yu>w</code>	Set window limits (-16383,,16383)																					
<code>Esc e<bit mask>x</code>	Set output device (-32768,,32767)																					
	bit meaning 0 8 bit mask mode 1 primary raster only 2 auxillary raster only 3 both the primary and auxillary raster																					
<code>Esc e<display mode>y</code>	Display output devices (-32768,,32767)																					
	mode meaning 0 enable 8 bit display mode 1 primary raster only 2 auxillary raster only 3 ORed rasters 4 foreground/background mode, 8 bit mask mode 5 primary raster only 6 auxillary raster only 7 foreground/background mode, 4 bit mask mode																					
<code>Esc e<mask>,<view#>^</code>	Inquire view <mask> = (-32768,,-1,1,,32767) <view#> = (0,,255)																					
	<table border="1"> <thead> <tr> <th>bit</th> <th>format</th> <th>meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>n</td> <td>view exists (0=No/1=Yes)</td> </tr> <tr> <td>1</td> <td>n</td> <td>view is active (0=No/1=Yes)</td> </tr> <tr> <td>2</td> <td>n</td> <td>view is highlighted (0=No/1=Yes)</td> </tr> <tr> <td>3</td> <td>+nnnnn</td> <td>highlight pen (valid only if highlighted)</td> </tr> <tr> <td>4</td> <td>+nnnnn</td> <td>background color</td> </tr> <tr> <td>5</td> <td>+nnnnn, +nnnnn, +nnnnn, +nnnnn</td> <td>viewport</td> </tr> </tbody> </table>	bit	format	meaning	0	n	view exists (0=No/1=Yes)	1	n	view is active (0=No/1=Yes)	2	n	view is highlighted (0=No/1=Yes)	3	+nnnnn	highlight pen (valid only if highlighted)	4	+nnnnn	background color	5	+nnnnn, +nnnnn, +nnnnn, +nnnnn	viewport
bit	format	meaning																				
0	n	view exists (0=No/1=Yes)																				
1	n	view is active (0=No/1=Yes)																				
2	n	view is highlighted (0=No/1=Yes)																				
3	+nnnnn	highlight pen (valid only if highlighted)																				
4	+nnnnn	background color																				
5	+nnnnn, +nnnnn, +nnnnn, +nnnnn	viewport																				

ESCAPE CODE	FUNCTION
	6 +nnnnn, window +nnnnn, +nnnnn, +nnnnn
	7 +nnn.nnnn, zoom factor +nnn.nnnn
	8 +nnnnn, zoom center +nnnnn
	9 +nnnnn active view number

OBJECT COMMANDS — Section 4

ESCAPE CODE	FUNCTION
$\text{\textasciitilde}g\langle\langle\text{object attribute list name}\rangle\rangle a$	Specify active object
$\text{\textasciitilde}g b$	Set default attributes of active object attribute list: Visibility 1 Highlighting off Pick Priority 1 Transformation center 0,0 translation 0,0 rotation 0 scaling 1,1 Object write mask 15
$\text{\textasciitilde}g\langle\langle\text{vector list name}\rangle\rangle c$	Create object (use with $\text{\textasciitilde}g a$)
$\text{\textasciitilde}g d$	Delete active object attribute list
$\text{\textasciitilde}g\langle\text{mode}\rangle h$	Turn on or off object highlighting 0 = off 1 = on
$\text{\textasciitilde}g k$	Draw object
$\text{\textasciitilde}g l$	Delete all object attribute lists
$\text{\textasciitilde}g\langle\langle\text{new object name}\rangle\rangle n$	Rename object attribute list
$\text{\textasciitilde}g\langle\langle\text{vector list name}\rangle\rangle o$	Replace vector list reference
$\text{\textasciitilde}g\langle\text{value}\rangle p$	Set pick priority (0,,32767)

OBJECT COMMANDS

ESCAPE CODE	FUNCTION																																													
Ⓔⓖ⟨run⟩,⟨rise⟩r or Ⓔⓖ⟨theta⟩r	Set rotation ⟨run⟩,⟨rise⟩,⟨theta⟩ = (-127.996,,127.996) run,rise: direction vector theta: radians counter clockwise																																													
Ⓔⓖ⟨Sx⟩,⟨Sy⟩s	Set scale ⟨Sx⟩,⟨Sy⟩=(-127.996,,127.996)																																													
Ⓔⓖ⟨Tx⟩,⟨Ty⟩T	Set translation ⟨Tx⟩,⟨Ty⟩=(-32766,,32766)																																													
Ⓔⓖgu	Undraw object																																													
Ⓔⓖ⟨value⟩v	Set visibility (-32768,,32767)																																													
Ⓔⓖ⟨mask⟩w	Set object write mask (-32768,,32767)																																													
Ⓔⓖ⟨x⟩,⟨y⟩x	Set transformation center ⟨x⟩,⟨y⟩=(-16383,,16383)																																													
Ⓔⓖ⟨mask⟩^	Inquire object attribute list (-32768,,-1,1,,32767) <table border="1"> <thead> <tr> <th>bit</th> <th>format</th> <th>meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>n</td> <td>object attribute list name exists (0=No/1=Yes)</td> </tr> <tr> <td>1</td> <td>n</td> <td>highlighting on (0=No/1=Yes)</td> </tr> <tr> <td>2</td> <td>+nnnnn</td> <td>pick priority</td> </tr> <tr> <td>3</td> <td>+nnnnn</td> <td>visibility</td> </tr> <tr> <td>4</td> <td>+nnnnn</td> <td>write mask</td> </tr> <tr> <td>5</td> <td>+nnnnn,</td> <td>transformation center</td> </tr> <tr> <td>6</td> <td>+nnn.nnnn,</td> <td>scaling</td> </tr> <tr> <td></td> <td>+nnn.nnnn</td> <td></td> </tr> <tr> <td>7</td> <td>+nnn.nnnn,</td> <td>rotation</td> </tr> <tr> <td></td> <td>+nnn.nnnn</td> <td></td> </tr> <tr> <td>8</td> <td>+nnnnn,</td> <td>translation</td> </tr> <tr> <td></td> <td>+nnnnn</td> <td></td> </tr> <tr> <td>9</td> <td>12 chars</td> <td>object attribute list name</td> </tr> <tr> <td>10</td> <td>12 chars</td> <td>vector list name</td> </tr> </tbody> </table>	bit	format	meaning	0	n	object attribute list name exists (0=No/1=Yes)	1	n	highlighting on (0=No/1=Yes)	2	+nnnnn	pick priority	3	+nnnnn	visibility	4	+nnnnn	write mask	5	+nnnnn,	transformation center	6	+nnn.nnnn,	scaling		+nnn.nnnn		7	+nnn.nnnn,	rotation		+nnn.nnnn		8	+nnnnn,	translation		+nnnnn		9	12 chars	object attribute list name	10	12 chars	vector list name
bit	format	meaning																																												
0	n	object attribute list name exists (0=No/1=Yes)																																												
1	n	highlighting on (0=No/1=Yes)																																												
2	+nnnnn	pick priority																																												
3	+nnnnn	visibility																																												
4	+nnnnn	write mask																																												
5	+nnnnn,	transformation center																																												
6	+nnn.nnnn,	scaling																																												
	+nnn.nnnn																																													
7	+nnn.nnnn,	rotation																																												
	+nnn.nnnn																																													
8	+nnnnn,	translation																																												
	+nnnnn																																													
9	12 chars	object attribute list name																																												
10	12 chars	vector list name																																												

VECTOR LIST COMMANDS — Section 4

ESCAPE CODE	FUNCTION									
<code>␣.ha</code>	Define hidden vector list									
<code>␣.hb</code>	Clear vector list									
<code>␣.h<<vector list name>>c</code>	Close vector list/ Assign vector list name									
<code>␣.hc</code>	Abort vector list									
<code>␣.h<<vector list name>>d</code>	Delete vector list									
<code>␣.h<<vector list name>>e</code>	Delete vector list and all associated objects attribute lists									
<code>␣.hl</code>	Delete vector list library									
<code>␣.h<<new name>><<old vector list name>>n</code>	Rename vector list									
<code>␣.h<<new name>><<old vector list name>>o</code>	Replace vector list's references									
<code>␣.h<<vector list name>>s</code>	Set vector list name for status									
<code>␣.h<mask>^</code>	Inquire vector list (-32768,-1,1,,32767)									
	<table border="1"> <thead> <tr> <th>bit</th> <th>format</th> <th>meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>n</td> <td>vector list name exists, and is not null (0=No/1= Yes)</td> </tr> <tr> <td>1</td> <td>n</td> <td>vector list is referemced (0=No/1= Yes)</td> </tr> </tbody> </table>	bit	format	meaning	0	n	vector list name exists, and is not null (0=No/1= Yes)	1	n	vector list is referemced (0=No/1= Yes)
bit	format	meaning								
0	n	vector list name exists, and is not null (0=No/1= Yes)								
1	n	vector list is referemced (0=No/1= Yes)								



INTERACTIVE FUNCTIONS — Section 8

ESCAPE CODE	FUNCTION												
<code>␣.ip</code>	Pick primitive within pick aperture												
<code>␣.i<<text>>t</code>	Inquire text length												
<code>␣.i<mask>^</code>	Inquire pick (-32768,,32767)												
	<table border="1"> <thead> <tr> <th>bit</th> <th>format</th> <th>meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>n</td> <td>pick sucess (0=No/1= Yes)</td> </tr> <tr> <td>1</td> <td>+nnnnn, +nnnnn</td> <td>pick aperture device coordinates, x and y</td> </tr> <tr> <td>2</td> <td>+nnnnn, +nnnnn</td> <td>pick aperture size (device coordinates), x and y</td> </tr> </tbody> </table>	bit	format	meaning	0	n	pick sucess (0=No/1= Yes)	1	+nnnnn, +nnnnn	pick aperture device coordinates, x and y	2	+nnnnn, +nnnnn	pick aperture size (device coordinates), x and y
bit	format	meaning											
0	n	pick sucess (0=No/1= Yes)											
1	+nnnnn, +nnnnn	pick aperture device coordinates, x and y											
2	+nnnnn, +nnnnn	pick aperture size (device coordinates), x and y											

ESCAPE CODE	FUNCTION
	3 +nnnnn pick ID
	4 12 chars object attribute list name
	5 12 chars vector list name

GRAPHICS TEXT LABEL — Section 2

ESCAPE CODE	FUNCTION
$\text{\textasciitilde} * l \langle \text{text label} \rangle \langle \text{c}, \text{c}', \text{f}, \text{f}', \text{h}, \text{h}', \text{r}, \text{r}' \rangle$	Display text label

VECTOR DRAWING MODE — Section 6

ESCAPE CODE	FUNCTION																																																																																
$\text{\textasciitilde} * m \langle \text{mode} \rangle a$	Select drawing mode																																																																																
	<table border="0" style="margin-left: 40px;"> <thead> <tr> <th>mode</th> <th>pattern</th> <th>value</th> <th>effect</th> <th></th> </tr> </thead> <tbody> <tr> <td>0 NOP</td> <td></td> <td>0,1</td> <td>NOP</td> <td></td> </tr> <tr> <td>1 Clear1</td> <td></td> <td>0</td> <td>NOP</td> <td></td> </tr> <tr> <td></td> <td></td> <td>1</td> <td>pixel</td> <td>← viewport background color</td> </tr> <tr> <td>2 Jam1</td> <td></td> <td>0</td> <td>NOP</td> <td></td> </tr> <tr> <td></td> <td></td> <td>1</td> <td>pixel</td> <td>← primary pen</td> </tr> <tr> <td>3 Comp1</td> <td></td> <td>0</td> <td>NOP</td> <td></td> </tr> <tr> <td></td> <td></td> <td>1</td> <td>pixel</td> <td>← NOT pixel</td> </tr> <tr> <td>4 Jam2</td> <td></td> <td>0</td> <td>pixel</td> <td>← secondary pen</td> </tr> <tr> <td></td> <td></td> <td>1</td> <td>pixel</td> <td>← primary pen</td> </tr> <tr> <td>5 OR</td> <td></td> <td>0</td> <td>NOP</td> <td></td> </tr> <tr> <td></td> <td></td> <td>1</td> <td>pixel</td> <td>← pixel OR primary pen</td> </tr> <tr> <td>6 Comp2</td> <td></td> <td>0</td> <td>NOP</td> <td></td> </tr> <tr> <td></td> <td></td> <td>1</td> <td>pixel</td> <td>← pixel XOR (primary pen XOR background pen)</td> </tr> <tr> <td>7 Clear2</td> <td></td> <td>0</td> <td>NOP</td> <td></td> </tr> <tr> <td></td> <td></td> <td>1</td> <td>pixel</td> <td>← pixel AND (NOT primary pen)</td> </tr> </tbody> </table>	mode	pattern	value	effect		0 NOP		0,1	NOP		1 Clear1		0	NOP				1	pixel	← viewport background color	2 Jam1		0	NOP				1	pixel	← primary pen	3 Comp1		0	NOP				1	pixel	← NOT pixel	4 Jam2		0	pixel	← secondary pen			1	pixel	← primary pen	5 OR		0	NOP				1	pixel	← pixel OR primary pen	6 Comp2		0	NOP				1	pixel	← pixel XOR (primary pen XOR background pen)	7 Clear2		0	NOP				1	pixel	← pixel AND (NOT primary pen)
mode	pattern	value	effect																																																																														
0 NOP		0,1	NOP																																																																														
1 Clear1		0	NOP																																																																														
		1	pixel	← viewport background color																																																																													
2 Jam1		0	NOP																																																																														
		1	pixel	← primary pen																																																																													
3 Comp1		0	NOP																																																																														
		1	pixel	← NOT pixel																																																																													
4 Jam2		0	pixel	← secondary pen																																																																													
		1	pixel	← primary pen																																																																													
5 OR		0	NOP																																																																														
		1	pixel	← pixel OR primary pen																																																																													
6 Comp2		0	NOP																																																																														
		1	pixel	← pixel XOR (primary pen XOR background pen)																																																																													
7 Clear2		0	NOP																																																																														
		1	pixel	← pixel AND (NOT primary pen)																																																																													
$\text{\textasciitilde} * m \langle \text{line type} \rangle b$	Select line type																																																																																
	<ul style="list-style-type: none"> 1 = Solid line 2 = User defined line pattern 3 = Current areafill pattern 4 = Predefined line pattern #1 5 = Predefined line pattern #2 6 = Predefined line pattern #3 7 = Predefined line pattern #4 8 = Predefined line pattern #5 9 = Predefined line pattern #6 10 = Predefined line pattern #7 11 = Point plot 																																																																																

VECTOR DRAWING MODE

ESCAPE CODE	FUNCTION
$\text{\textasciitilde{m}}\langle\text{pattern}\rangle,\langle\text{scale}\rangle\text{c}$	Define user line pattern $\langle\text{pattern}\rangle = (-32768,,32767)$ $\langle\text{scale}\rangle = (1,,255)$
$\text{\textasciitilde{m}}\langle\text{row0}\rangle,\dots,\langle\text{row 7}\rangle\text{d}$	Define user area fill pattern $\langle\text{row}\#\rangle = (-32768,,32767)$
$\text{\textasciitilde{m}}\langle\text{x1}\rangle,\langle\text{y1}\rangle,\langle\text{xu}\rangle,\langle\text{yu}\rangle\text{e}$	Rectangle fill, absolute $(-16383,,16383)$
$\text{\textasciitilde{m}}\langle\text{xL}\rangle,\langle\text{yL}\rangle,\langle\text{xU}\rangle,\langle\text{yU}\rangle\text{f}$	Rectangle fill, relocatable $(-32766,,32766)$
$\text{\textasciitilde{m}}\langle\text{area pattern}\rangle\text{g}$	Select area pattern 1 Solid area fill 2 User defined area fill pattern 3 $(128,0,32,0,8,0,2,0)$ 45 deg. dashed hatching 4 $(0,0,0,16,8,4,2,1)$ 45 deg. dashed hatching 5 $(128,64,32,16,8,4,2,1)$ 45 deg. hatching 6 $(129,66,36,24,24,36,66,129)$ 2-way cross-hatching 7 $(136,68,34,17,136,68,34,17)$ 45 deg. fine hatching 8 $(153,102,153,102,153,102,153,102)$ 3-way cross-hatching 9 $(85,170,85,170,85,170,85,170)$ 1:1 blend 10 $(187,238,187,238,187,238,187,238)$ 3:1 blend
$\text{\textasciitilde{m}}\langle\text{pen}\#\rangle\text{h}$	Define area fill boundary pen $(-32768,,32767)$
$\text{\textasciitilde{m}}\text{h}$	Disable area boundary pen
$\text{\textasciitilde{m}}\langle\text{pick ID}\rangle\text{i}$	Set pick ID $(0,,32767)$
$\text{\textasciitilde{m}}\langle\text{x}\rangle,\langle\text{y}\rangle\text{j}$	Set relocatable origin $(-16383,,16383)$
$\text{\textasciitilde{m}}\text{k}$	Set relocatable origin to current pen position
$\text{\textasciitilde{m}}\text{l}$	Set relocatable origin to graphics cursor position
$\text{\textasciitilde{m}}\langle\text{x size}\rangle,\langle\text{y size}\rangle\text{m}$	Set graphics text size $(-16383.996,,16383.996)$

VECTOR DRAWING MODE

ESCAPE CODE	FUNCTION																																						
$\text{\textbackslash}^{\text{t}}^{\text{m}}\langle\text{angle code}\rangle\text{n}$	Set graphics text angle in virtual coordinate units angle code meaning 1 0 degrees 2 90 degrees 3 180 degrees 4 270 degrees																																						
$\text{\textbackslash}^{\text{t}}^{\text{m}}\langle\tan(\text{slant angle})\rangle\text{o}$	Turn on text slant $\langle\tan(\text{slant angle})\rangle = (-1,1)$																																						
$\text{\textbackslash}^{\text{t}}^{\text{m}}\text{p}$	Turn off text slant																																						
$\text{\textbackslash}^{\text{t}}^{\text{m}}\langle\text{origin}\rangle\text{q}$	Set graphics text justification left justified . . . 0 origin at horizontal baseline 1 origin at bottom of text cell 2 origin at middle of text cell 3 origin at top of text cell center justified . . . 10 origin at horizontal baseline 4 origin at bottom of text cell 5 origin at middle of text cell 6 origin at top of text cell right justified . . . 11 origin at horizontal character axis 7 origin at bottom of text cell 8 origin at middle of text cell 9 origin at top of text cell																																						
$\text{\textbackslash}^{\text{t}}^{\text{m}}\text{r}$	Set all graphics defaults (listed under $\text{\textbackslash}^{\text{t}}^{\text{m}}\text{r}$)																																						
$\text{\textbackslash}^{\text{t}}^{\text{m}}\text{r}$	Set starred (*) defaults																																						
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Default</th> </tr> </thead> <tbody> <tr> <td>*Move/draw flag</td> <td>"Draw"</td> </tr> <tr> <td>*Line type</td> <td>1(solid)</td> </tr> <tr> <td>*Drawing mode</td> <td>4(jam2)</td> </tr> <tr> <td>*User defined line pattern</td> <td>255,1</td> </tr> <tr> <td>*User defined area pattern</td> <td>255,255, ...255</td> </tr> <tr> <td>*Primary Pen</td> <td>15</td> </tr> <tr> <td>*Secondary Pen</td> <td>0</td> </tr> <tr> <td>Relocatable origin</td> <td>0,0</td> </tr> <tr> <td>*Pick ID</td> <td>0</td> </tr> <tr> <td>*Text size</td> <td>1</td> </tr> <tr> <td>*Text direction</td> <td>1</td> </tr> <tr> <td>*Text origin</td> <td>1(left,bottom)</td> </tr> <tr> <td>*Text slant</td> <td>off</td> </tr> <tr> <td>*Text angle</td> <td>0</td> </tr> <tr> <td>*Character spacing</td> <td>0</td> </tr> <tr> <td>*Text pen</td> <td>7(on)</td> </tr> <tr> <td>*Symbol mode</td> <td>1(off)</td> </tr> <tr> <td>*Symbol scale</td> <td>1.0</td> </tr> </tbody> </table>	Parameter	Default	*Move/draw flag	"Draw"	*Line type	1(solid)	*Drawing mode	4(jam2)	*User defined line pattern	255,1	*User defined area pattern	255,255, ...255	*Primary Pen	15	*Secondary Pen	0	Relocatable origin	0,0	*Pick ID	0	*Text size	1	*Text direction	1	*Text origin	1(left,bottom)	*Text slant	off	*Text angle	0	*Character spacing	0	*Text pen	7(on)	*Symbol mode	1(off)	*Symbol scale	1.0
Parameter	Default																																						
*Move/draw flag	"Draw"																																						
*Line type	1(solid)																																						
*Drawing mode	4(jam2)																																						
*User defined line pattern	255,1																																						
*User defined area pattern	255,255, ...255																																						
*Primary Pen	15																																						
*Secondary Pen	0																																						
Relocatable origin	0,0																																						
*Pick ID	0																																						
*Text size	1																																						
*Text direction	1																																						
*Text origin	1(left,bottom)																																						
*Text slant	off																																						
*Text angle	0																																						
*Character spacing	0																																						
*Text pen	7(on)																																						
*Symbol mode	1(off)																																						
*Symbol scale	1.0																																						

GRAPHICS TEXT

ESCAPE CODE	FUNCTION
$\text{\textbackslash}t \cdot n \langle x \rangle, \langle y \rangle d$	Define character coordinate (-63,,63)
$\text{\textbackslash}t \cdot n \langle font \rangle e$	Delete user defined font(s) (3,,16)
$\text{\textbackslash}t \cdot n e$	Delete all user defined fonts
$\text{\textbackslash}t \cdot n \langle font \rangle f$	Set text font 1 = Standard 2 = Standard Roman Extension Font 3,,16 = User defined fonts
$\text{\textbackslash}t \cdot n \langle mode \rangle m$	Set graphics text margin 0 = Text margin follows pen movement 1 = Text margin is set permanently at the current pen position
$\text{\textbackslash}t \cdot n \langle run \rangle, \langle rise \rangle r$ or $\text{\textbackslash}t \cdot n \langle theta \rangle r$	Set text rotation $\langle run \rangle, \langle rise \rangle, \langle theta \rangle =$ (-127.996,,127.996)
$\text{\textbackslash}t \cdot n \langle x \text{ size} \rangle, \langle y \text{ size} \rangle s$	Set graphics text size (-16383,,16383)
$\text{\textbackslash}t \cdot n \langle character \rangle, \langle font \rangle u$	Set user defined character as symbol $\langle character \rangle = (32,,126)$ $\langle font \rangle = (13,,16)$
$\text{\textbackslash}t \cdot n \langle spacing \rangle w$	Set character spacing (-127.966,,127.966)
$\text{\textbackslash}t \cdot n \langle pen \rangle x$	Set graphics text pen color (-32768,,32767)

PLOTTING COMMANDS — Section 2

ESCAPE CODE	FUNCTION
$\text{\textbackslash}t \cdot p a$	Lift pen
$\text{\textbackslash}t \cdot p b$	Lower pen
$\text{\textbackslash}t \cdot p c$	Use graphics cursor as new point
$\text{\textbackslash}t \cdot p d$	Draw a point at the current pen position
$\text{\textbackslash}t \cdot p e$	Set relocatable origin to the current pen position

 PLOTTING COMMANDS

ESCAPE CODE	FUNCTION
$\text{\textasciitilde}^*pf$	Data is ASCII absolute
$\text{\textasciitilde}^*pg$	Data is ASCII incremental
$\text{\textasciitilde}^*ph$	Data is ASCII relocatable
$\text{\textasciitilde}^*pi$	Data is binary medium absolute
$\text{\textasciitilde}^*pj$	Data is binary short incremental
$\text{\textasciitilde}^*pk$	Data is binary long incremental
$\text{\textasciitilde}^*pl$	Data is binary long relocatable
$\text{\textasciitilde}^*pm$	Data is binary long absolute
$\text{\textasciitilde}^*ps$	Start area fill plot specification
$\text{\textasciitilde}^*pt$	Terminate area fill specification
$\text{\textasciitilde}^*pu$	Lift area fill boundary pen
$\text{\textasciitilde}^*pv$	Lower area fill boundary pen
$\text{\textasciitilde}^*pz$	NOP

RASTER HARDCOPY FORMATS — Section 7

ESCAPE CODE	FUNCTION												
$\text{\textasciitilde}^*qa$	Selects active values												
$\text{\textasciitilde}^*q<device ID>c$	Set device class												
	<table> <tbody> <tr> <td>device id</td> <td>printer class</td> </tr> <tr> <td>1</td> <td>PRINTERx</td> </tr> <tr> <td>2</td> <td>PLOTTERx</td> </tr> <tr> <td>3</td> <td>GID</td> </tr> <tr> <td>4</td> <td>RASTER</td> </tr> <tr> <td>5</td> <td>EXTERNAL MONITOR</td> </tr> </tbody> </table>	device id	printer class	1	PRINTERx	2	PLOTTERx	3	GID	4	RASTER	5	EXTERNAL MONITOR
device id	printer class												
1	PRINTERx												
2	PLOTTERx												
3	GID												
4	RASTER												
5	EXTERNAL MONITOR												
$\text{\textasciitilde}^*qd$	Selects default values												
$\text{\textasciitilde}^*q<device index>i$	Set device index												

 RASTER HARDCOPY FORMATS

ESCAPE CODE	FUNCTION	
$\text{Esc} \cdot \text{q} \langle \text{field ID} \rangle \text{f}$	Set current field	
	Field ID	Configuration Field
	PRINTERx	
	1	Raster dump orientation: 0 = Horizontal 1 = Vertical
	2	Raster dump content: 0 = Color 1 = Black and White 2 = Halftoned 3 = Alpha only
	3	Raster dump expansion factors, specified as (x factor, y factor): (0.0000,,4.0000)
	4	Raster dump expansion mode: 0 = Internal expansion 1 = External expansion
	5	Printer auto centering: 0 = No 1 = Yes 2 = None
	6	Raster dump alternate driver options: (0,,255)
	7	Termination formfeed: 0 = No 1 = Yes
	8	Data compaction 0 = No 1 = Yes
	Field ID	Configuration Field
	PLOTTERx	
	1	Default P1,P2—specified as (xlower, ylower, xupper, yupper): (-32768,,32767)
	2	P1, P2 default use: 0 = No 1 = Yes
	3	Aspect preservation: 0 = No 1 = Yes
	4	Chart advance: 0 = None 1 = Full 2 = Half
	5	Number of pens: (1,,99)
	6	Pen speed: (0,,99)
	7	Areafill spacing: (0,,99)

RASTER HARDCOPY FORMATS

ESCAPE CODE	FUNCTION	
	8	Options: (0,,255) Bit Meaning 0 Plot pen zero (0=No/1=Yes)
	GID	
	1	X sensitivity
	2	Y sensitivity
	RASTER	
	1	Raster dump/load screen limits, specified as (x limit, y limit): (0,,511)
	2	Options (0,,255)
	EXTERNAL MONITOR	
	1	External monitor enable 0 = Off 1 = On
<code>␣*qp</code>		Selects power on values
<code>␣*q<value>v</code>		Changes current field to "value"
<code>␣*q<mask>^</code>		Graphics device configuration inquiry (-32768,,-1,1,,32767)
	Bit	Format Meaning
	0	n Device configuration menu is locked (unlock with <code>␣*q0L</code>)
	1	n Device type exists
	2	n Ordinal device pen exists
	3	n Current field type: 0 Non existent 1 Selection field 2 Integer 3 Decimal 4 Integer array 5 Decimal array 6 Bounded string
	4	* Current field value:
Type	Format	Meaning
	0	Non existent
	1 +nnnnn	Selection field
	2 +nnnnn	Integer
	3 +nnnnn.nnnn	Decimal
	4 n,+nnnnn,+nnnnn,. . .	n* integer array
	5 n,+nnnnn.nnnn, +nnnnn.nnnn,. . .	n* decimal array
	6 1 char	bounded string of length <=80 chars.

 RASTER DUMP FORMATS

ESCAPE CODE	FUNCTION
$\text{\textasciitilde{r}}\langle\# \text{of planes}\rangle\text{o}$	Level window address (0,,9999)
$\text{\textasciitilde{r}}\langle\# \text{of pixels}\rangle\text{p}$	Horizontal window dimension
$\text{\textasciitilde{r}}\langle\# \text{of lines}\rangle\text{q}$	Vertical window dimension
$\text{\textasciitilde{r}}\langle\# \text{of planes}\rangle\text{r}$	Level window dimension (0,,9999)
$\text{\textasciitilde{r}}\langle\# \text{of data pixels}\rangle\text{s}$	Specify source x size
$\text{\textasciitilde{r}}\langle\# \text{of data lines}\rangle\text{t}$	Specify source y size
$\text{\textasciitilde{r}}\langle\# \text{of data planes}\rangle\text{u}$	Specify source z size
$\text{\textasciitilde{r}}\langle\text{X origin}\rangle\text{x}$	Set X origin (-9999,,9999)
$\text{\textasciitilde{r}}\langle\text{Y origin}\rangle\text{y}$	Set Y origin (-9999,,9999)
$\text{\textasciitilde{r}}\langle\text{Z origin}\rangle\text{z}$	Set Z origin (-9999,,9999)

 GRAPHICS STATUS — Section 8

ESCAPE CODE	FUNCTION
$\text{\textasciitilde{s}}\langle\text{parameter}\rangle\text{\textasciitilde{^}}$	Read graphics status
$\langle\text{parameter}\rangle = 0$	Read error code
	$\langle\text{error code}\rangle$
	+nnnnn
	bit error description
	0 wrong parameter type
	1 wrong number of parameters
	2 parameter out of range
	3 illegal parameter
	4 undefined character
	5 undefined font
	6 undefined operand
	7 illegal operand
	8 arithmetic overflow
	9 memory not available
1	Read device ID
	$\langle\text{device ID}\rangle 1 \langle\text{term}\rangle$
	2703A
2	Read pen position
	$\langle\text{X}\rangle, \langle\text{Y}\rangle, \langle\text{pen}\rangle$
	+nnnnn,+nnnnn,n
	$\langle\text{pen}\rangle + 0,$
	pen up, next coordinate is a move
	1, pen down, next coordinate is a draw

 GRAPHICS STATUS

ESCAPE CODE	FUNCTION
3	Read graphics cursor position <X>,<Y> + nnnnn,+ nnnnn
4	Read graphics cursor position and wait for key <X>,<Y>,<keycode> + nnnnn,+ nnnnn,nnn
5	Read raster size <xmin>,<ymin>,<xmax>,<ymax>, <xres>,<yres> +00000,+00000,+00511,+00389,00002.,00002.
6	Read graphics capabilities <b1>,<b2>,...,<b15>,<b16> 3,16,2,16,1,0,0,1,2,3,2,2,0,0,0,0 <b1> Clear display 0 no clear 1 paper advance 2 full screen clear 3 partial clear by area <b2> Number of pens <b3> Color capability 0 black and white 1 grey levels 2 color <b4> Grayplane capability <b5> Area shading 0 no 1 yes <b6>-<7> not used <b8> Dynamic modification 0 no 1 yes <b9> Graphics character size 0 fixed 1 integer multiples of base size 2 any size <b10> Graphics character angles 0 fixed 1 multiples of 90 degrees 2 multiples of 45 degrees 3 any angle <b11> Graphics character slant 0 fixed 1 45 degrees 2 any angle from -45 to 45 degrees <b12> Line patterns 0 none 1 predefined only 2 user defined and predefined <b13>-<b16> not used
7	Read graphics text status <xsize>,<ysize>,<lorg>,<angle>,<slant> + nnnnn,+ nnnnn,nn,nnn.nnnn,+ nnn.nnnn (<angle>,<slant> in radians)

 GRAPHICS STATUS

ESCAPE CODE	FUNCTION
	8 Read zoom status
	<zoom size>,<zoom on/off> nnn.nnnn,1 <zoom on/off> 0 = zoom off 1 = zoom on
	9 Read relocatable origin
	<X coordinate>,<Y coordinate> +nnnn,+nnnn
	10 Read reset status
	<reset>,<b1>,<b2>,...,<b6>,<b7> <b1>-<b7> not used n,0,0,0,0,0,0 <reset> 0 = no reset since last check 1 = terminal reset since last check
	11 Read area shading capability
	<capability>,<xpattern>,<ypattern> 2,8,8 <capability> Area fill restrictions 1 rectangular only 2 closed polygon only
	12 Read graphics modification capabilities
	<erase capability>,<complement capability> 1,1 <erase capability> Selective erase 0 no 1 yes <complement capability> Complement mode 0 no 1 yes
	13 Read secondary cursor information
	<type>,<dx>,<dy>,<units>,<validity> n,+nnnn,+nnnn,n,n <type> Cursor type 0 short crosshair 1 full screen crosshair 2 box <units> Cursor definition units 0 display 1 vector list <validity> Cursor validity 0 cursor position and size info is valid at this time in these units 1 cursor info is not valid
	14 Read graphics redraw flag
	<flag> 0 = off 1 = on
	15 Read graphics redraw mode
	<mode> +nnnn
	16 Read graphics output device
	<device> +nnnn

GRAPHICS STATUS

ESCAPE CODE	FUNCTION
	17 Read graphics displayed output <device> + nnnnn
	18 Read transparent mode flag <flag> n 0 = off 1 = on
	19 Read graphics save mode <mode> + nnnnn
	20 Read pick ID <ID> + nnnnn
	21 Read current graphics draw mode <mode> + nnnnn
	22 Read current graphics line type <line> + nnnnn
	23 Read current graphics area pattern <area> + nnnnn
	24 Return symbol size <size> + nnn.nnnn
	25 Return graphics symbol mode <mode> + nnnnn
	26 Return graphics symbol units <units> + nnnnn
	27 Return user defined symbol letter <usymb> + nnnnn
	28 Return user defined symbol font <ufont> + nnnnn
	29 Return primary pen # <pen> + nnnnn
	30 Return secondary pen # <pen> + nnnnn
	31 Read graphics free space <freespace> + nnnnnnnn

RASTER HARDCOPY FORMATS — Sections 6 and 7

ESCAPE CODE	FUNCTION																																		
$\text{\textasciitilde}u\langle mode \rangle a$	Set black and white halftone mode 0 = Halftone off 1 = Halftone on																																		
$\text{\textasciitilde}u\text{b}$	Initialize default hardcopy format. Sets default parameters as follows: Black and white halftone mode = off Halftone pen = 0 Pattern map: <table border="0" style="margin-left: 40px;"> <thead> <tr> <th>Index</th> <th>Level</th> </tr> </thead> <tbody> <tr><td>0</td><td>1/16</td></tr> <tr><td>1</td><td>5/16</td></tr> <tr><td>2</td><td>9/16</td></tr> <tr><td>3</td><td>13/16</td></tr> <tr><td>4</td><td>3/16</td></tr> <tr><td>5</td><td>7/16</td></tr> <tr><td>6</td><td>11/16</td></tr> <tr><td>7</td><td>15/16</td></tr> <tr><td>8</td><td>2/16</td></tr> <tr><td>9</td><td>6/16</td></tr> <tr><td>10</td><td>10/16</td></tr> <tr><td>11</td><td>14/16</td></tr> <tr><td>12</td><td>4/16</td></tr> <tr><td>13</td><td>8/16</td></tr> <tr><td>14</td><td>12/16</td></tr> <tr><td>15</td><td>16/16</td></tr> </tbody> </table>	Index	Level	0	1/16	1	5/16	2	9/16	3	13/16	4	3/16	5	7/16	6	11/16	7	15/16	8	2/16	9	6/16	10	10/16	11	14/16	12	4/16	13	8/16	14	12/16	15	16/16
Index	Level																																		
0	1/16																																		
1	5/16																																		
2	9/16																																		
3	13/16																																		
4	3/16																																		
5	7/16																																		
6	11/16																																		
7	15/16																																		
8	2/16																																		
9	6/16																																		
10	10/16																																		
11	14/16																																		
12	4/16																																		
13	8/16																																		
14	12/16																																		
15	16/16																																		
$\text{\textasciitilde}u\langle pen \rangle i$	Select halftone pen (-32768,,32767) mod 16																																		
$\text{\textasciitilde}u\langle row0 \rangle, \dots, \langle row7 \rangle p$	Define halftone pattern (-32768,,32767) mod 256																																		
$\text{\textasciitilde}u\langle mode \rangle t$	Set auto-halftone-tracking and map pattern to pen <table border="0" style="margin-left: 40px;"> <thead> <tr> <th>< mode >:</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Auto-halftone-tracking off</td> </tr> <tr> <td>1</td> <td>Halftone patterns match intensity of associated palette pens</td> </tr> <tr> <td>2</td> <td>Density pattern to least intense pen color, least dense pattern to most intense pen color</td> </tr> </tbody> </table>	< mode >:		0	Auto-halftone-tracking off	1	Halftone patterns match intensity of associated palette pens	2	Density pattern to least intense pen color, least dense pattern to most intense pen color																										
< mode >:																																			
0	Auto-halftone-tracking off																																		
1	Halftone patterns match intensity of associated palette pens																																		
2	Density pattern to least intense pen color, least dense pattern to most intense pen color																																		



RASTER HARDCOPY FORMATS

ESCAPE CODE	FUNCTION	
$\text{\textasciitilde}u\langle\text{mask}\rangle,\langle\text{pattern pen}\rangle^{\wedge}$	Raster hardcopy formats inquiry <mask>=(-32768,,-1,1,,32767) <pattern pen>:	
	Bit	Format
	0	+nnnnn
	1	+nnnnn
	2	+nnnnn,
		+nnnnn,+nnnnn,
		+nnnnn,+nnnnn,
		+nnnnn,+nnnnn,nnnnn,nnnnn
		Meaning
		Auto -halftone-
		tracking
		Halftone pen
		Halftone pattern
		for
		this pen

COLOR SPECIFICATION — Section 3

ESCAPE CODE	FUNCTION
$\text{\textasciitilde}v\langle\text{parm1}\rangle a$	Set first color parameter (0,1)
$\text{\textasciitilde}v\langle\text{parm2}\rangle b$	Set second color parameter (0,1)
$\text{\textasciitilde}v\langle\text{parm3}\rangle c$	Set third color parameter (0,1)
$\text{\textasciitilde}v\langle\text{palette}\rangle d$	Delete palette (0,,255)
$\text{\textasciitilde}v e$	Delete all palettes
$\text{\textasciitilde}v\langle\text{pen}\rangle i$	Select pen (0,,15)
$\text{\textasciitilde}v l$	Load selected palette
$\text{\textasciitilde}v\langle\text{method}\rangle m$	Set color specification method 0 = RGB 1 = HSL
$\text{\textasciitilde}v\langle\text{palette}\rangle p$	Select palette (0,,255)
$\text{\textasciitilde}v\langle\text{palette}\rangle r$	Reset palette (0,,255)

COLOR SPECIFICATION

ESCAPE CODE	FUNCTION	
$\text{E}^* \text{v} \langle \text{mask} \rangle, \langle \text{pen} \# \rangle, \langle \text{palette} \# \rangle ^*$	Inquire palette $\langle \text{mask} \rangle = (-32768, -1, 1, 32767)$ $\langle \text{pen} \rangle = (-32768, 32767)$ $\langle \text{palette} \# \rangle = (0, 127), (128, 255)$	
	bit	format meaning
	0	n palette exists (0=No/1=Yes)
	1	n palette is current palette (0=No/1=Yes)
	2	n method (0=RGB/1=HSL)
	3	+nnnnn current palette ID
	4	n.nnnn,n.nnnn, n.nnnn current pen RGB
	5	n.nnnn,n.nnnn, n.nnnn current pen HSL

PICTURE FILE COMMANDS — Section 5

ESCAPE CODE	FUNCTION	
$\text{E}^* \text{w} \langle \text{mode} \rangle \text{i}$	Inquire picture file	
	mode	format meaning
	1	$\langle \text{aaaaaaaaaaaa} \rangle$ object attribute list name
	2	$\langle \text{aaaaaaaaaaaa} \rangle, \langle \text{aaaaaaaaaaaa} \rangle, \text{n}$ object attribute list name, vector list name, vector list exists (0/1)
	3	$\langle \text{aaaaaaaaaaaa} \rangle$ vector list name
$\text{E}^* \text{wr}$	Graphics hard reset (See "Power-On Default Values" in this section)	
$\text{E}^* \text{w} \langle \text{mode} \rangle \text{t}$	Set transparent mode 0 = off 1 = on	
$\text{E}^* \text{w} \langle \text{fence} \rangle \text{v}$	Set Object visibility fence (-32768, 32767)	
$\text{E}^* \text{w} \langle \text{mode} \rangle \text{w}$	Set picture protection 0 = off 1 = on	

PICTURE FILE COMMANDS

ESCAPE CODE	FUNCTION	
$\text{\textasciitilde}w<save\ mask>y$	Set graphics save mode (-32768,,32767)	
	bit	meaning
	0	all color palettes
	1	active color palette
	2	system text fonts
	3	user defined text fonts
	4	current text font
	5	all views
	6	active view
	7	all hidden vector lists
	8	all objects
	9	active object and associated vector list
	10	psuedo object and displayed vector list
$\text{\textasciitilde}w<mask>^$	Inquire view (-32768,,32767)	
	bit	format meaning
	0	+nnnnn number of objects in picture file
	1	+nnnnn number of vector lists in picture file
	2	+nnnnn number of views in picture file
	3	+nnnnn number of palettes in picture file
	4	+nnnnn number of system graphics fonts
	5	+nnnnn number of user defined graphics fonts

POWER-ON DEFAULT VALUES

ITEM	VALUE	ESCAPE GROUP
raster buffer	clear	da,db
graphics display	on	dc,dd
alpha display	on	de,df
zoom factor	1,1	di
zoom center	255,185	dj
graphics cursor	off,short	dk,dl
rubberband line	off	dm,dn
graphics cursor address		
virtual	0,0	do,dp
display	0,0	
binding	1(virtual)	du
alpha cursor	on,0(ul)	dq,dr
Graphics text mode	off	ds,dt
display visibility	on(15)	dv
system write mask	on(15)	dw

POWER-ON DEFAULT VALUES

ITEM	VALUE	ESCAPE GROUP
Active view	0(default)	ei
viewport limits	0,0,511,389	ev
window limits	0,0,511,389	ew
highlighting	off	eh
viewport background color	0	eb
redraw mask	1	em
auto-redraw	on	ea
implied object	system	ga
system object attributes		
visibility	1	gv,gb
highlighting	off	gh,gb
detectability	1	gp,gb
transform center	0,0	gx
translation	0,0	gt,gb
rotation	0	gr,gb
scale	1,1	gs,gb
object write mask	15	gw,gb
drawing Mode	4(Jam2)	ma,mr
line type	1(solid)	mb,mr
user line pattern	255,1(solid)	mc
user fill pattern	255,255,255,255 255,255,255,255md	
area type	2(user)	mg
area highlighting	off	mh
pick id	0	mi
relocatable origin	0,0	mj,mk,ml,mr
text size	1	mm,mr
	6,9	ns
text angle	1	mn,mr
	0	nr
text slant	0	mo,mp,mr
text justification	1	mq,mr
symbol size	1.0000	ms
symbol mode	0(off)	mt
symbol units	0(display)	mu
primary pen	15	mx
secondary pen	0	my
text font	1(Standard)	nf
user symbol	"*"	nu
text spacing	0	nw
pen condition	up	pa,pb,mr
pen position	0,0	pc
plotting command		
data type	ASCII absolute	pf,pg,ph,pi,pj, pk,pl,pm


Escape Sequence Directory

POWER-ON DEFAULT VALUES

ITEM	VALUE	ESCAPE GROUP
raster dump configuration		
graphics video	on	rd,ri
horizontal window		
address 0	rm,ri	
vertical window		
address	0	rn,ri
plane window		
address	0	ro,ri
horizontal window		
dimension	9999	rp,ri
vertical window		
dimension	9999	rq,ri
level window		
dimension	9999	rr,ri
horizontal size	9999	rs,ri
vertical size	9999	rt,ri
levels	9999	ru,ri
x offset	0	rx,ri
y offset	0	ry,ri
z offset	0	rz,ri
raster hardcopy formats		
black & white		
halftone mode	off	er
halftone index	0	ui
auto-halftone-tracking	2 (high contrast)	ut
pattern map	as per auto values	up
initial graphics		
color palette	system	vl
current palette	0(default)	vp
current index	15	vi
color method	1(RGB)	vm
object visibility	fence	0
picture protection	1(enabled)	ww
graphics save mode	1474	wy

Glossary

E

active view	Although multiple views may be defined, only one view may be “active”. The zoom, pan, and cursor controls all work with the active view. A view is activated through the  keys or with the <code>⌘*e1</code> sequence.
color mapper	A look-up table on the Color Mapper board which associates the value of a raster pixel with a color on the active palette. The Color Mapper sends signals to the sweep circuitry to control the intensities of the red, green, and blue electron guns.
display cursor	The graphics cursor when it is defined for display (screen) units (<code>⌘*du</code>).
displayed vector list	A vector list ready to receive primitives at power-on. As primitives are entered in the displayed vector list, they are viewable on the screen through a set of system object attributes.
display write mask	Enables or disables the flow of picture file data to specified planes of the raster buffer. If a bit of the write mask is set to a “1”, data may be written to the associated raster plane. If a bit is set to “0”, data may not be written to the associated plane. Write masks, set with <code>⌘*dw</code> , may alter the original pen specification of a vector.
drawing mode	A mode selected with the <code>⌘*ma</code> command which determines the way in which bits are set in raster memory. The drawing mode is a primitive attribute.
HSL	A color selection method which defines a color by its hue (frequency), saturation (purity of color), and lightness (sharpness of color).
hidden vector lists	Vector lists opened with the <code>⌘*ha</code> command. Primitives entered into a hidden vector list are not displayed on the screen until the vector list is linked with object attributes.
object	A viewable image in virtual space consisting of a set of object attributes and a vector list reference.

- object attributes** General characteristics and transformation parameters applied to a vector list. The combination of a vector list and a set of object attributes constitutes an object. The image stored in a vector list is not viewable until the vector list is linked with a set of object attributes. Object attributes are:
- transformation center
 - translation
 - scaling
 - rotation
 - pick priority
 - visibility
 - object highlighting
 - object write mask
- object highlighting** An object attribute specified by the `*gh` command which causes the object to be redrawn in complement mode (drawing mode 3).
- object visibility fence** Objects with a visibility lower than or equal to the object visibility fence are not displayed. The object visibility fence is set with the `*wv` command.
- object write mask** An object attribute specified by the `*gw` command which determines the raster planes which may hold object data. Performs a function similar to the display write mask.
- output device** Output devices on the terminal are the primary and auxiliary raster, and the external monitor. The logical output device for picture file data is specified in the `*ex` command, and is used in conjunction with the display write mask.
- palettes** A set of 16 color pens. Up to 255 palettes may be stored in the terminal.
- pens** The decimal value of a raster pixel sent to the color mapper. The actual color values are set by the active palette. Possible pen assignments are as follows:
- Primary Pen - used to draw vectors (`*mx`).
 - Secondary Pen - used with the primary pen to draw line and area fill patterns in JAM2 mode (`*my`).
 - Text Pen - used to draw labels (`*nx`).
 - Viewport Background Pen - (`*eb`).
 - View Highlighting Pen - (`*eh`).
 - Area Boundary Pen - (`*mh`).
- pick** To select an object by pointing to it with the graphics cursor. Used in conjunction with the "Inquire Pick" command (`*i^`) which returns information about the selected object.

pick priority	An object attribute selected with the <code>*gp</code> command which determines the order in which objects are picked by the user. Resolves ambiguous picks among objects in the same vicinity in favor of the object with the highest pick priority.
picture file	The elements stored in vector memory used to create (or regenerate) a picture on the screen. These elements are: <ul style="list-style-type: none">• vector lists• object attributes• views• user defined fonts• palettes
pixel	Smallest addressable unit of the screen or raster. A screen pixel refers to a red-green-blue phosphor trio. A raster pixel is a 4-bit number in raster memory. The value of the raster pixel together with the color mapper determines the color of the screen pixel.
primitives	Elements of a vector list: vectors, area fills, labels, and pick ID's.
primitive attributes	Elements of a vector list: <ul style="list-style-type: none">• pens• line and area fill patterns• drawing modes• label attributes• symbol modes
psuedo object	The displayed vector list and system object attributes.
RGB	A color method specification which defines a color by its red, green, and blue components.
raster buffer	Represents the screen display as a binary array (512 x 512 x 4). The value of each raster element (pixel) is converted using the color mapper to a color dot on the screen.
raster dump	The transfer of raster data to printer, disc, or datacomm using the <code>COPY ALL :RASTER</code> command.
raster load	The transfer of raster data from disc or datacomm to a selected portion of the terminal's raster buffer.
raster plane	One level (512 x 512 x 1) of a raster buffer.
raster reads	Data read by a host CPU from the raster buffer.

read mask	Enables or disables the flow of data from a raster plane to the screen. If a bit of the read mask is set to "1", data may be read from the associated raster plane. Conversely, if a bit of the mask is set to "0", data may not be read from the associated raster plane. Read masks, set with the <code>ℱ*dv</code> command, alter the value of a raster pixel before it is sent to the color mapper.
screen viewport	An area of raster memory into which the contents of the virtual window are mapped. Specified by the <code>ℱ*ev</code> command.
symbol mode	A mode set by <code>ℱ*mt</code> which causes the specified system or user defined character to be plotted at each vector end point.
transformation center	An object attribute specified by the <code>ℱ*gx</code> command. It is the x,y coordinate pair about which all other object transformations (translation, rotation, scaling) take place.
transparent mode	A mode set by the <code>ℱ*wt</code> command which does not allow incoming primitives to be saved in a vector list.
vector list	A data structure composed of drawing elements (vectors, area fills, etc.) which defines a graphical entity.
vector space	Conceptual definition space (-16383,,+16383) for vector lists.
view	The mapping of a portion of virtual space (virtual window) to a portion of raster memory (screen viewport).
view highlighting	Outlining a view in the color of your choice. Set with the <code>ℱ*eh</code> command.
virtual cursor	Refers the graphics cursor when it is defined for virtual units. Set with the <code>ℱ*du</code> command.
virtual space	Conceptual plotting space for objects (-16383,,+16383).
virtual window	The area of virtual space, defined by the <code>ℱ*ew</code> , which is mapped to the raster
visibility	An object attribute specified by the <code>℄*gv</code> command which determines the order in which objects are drawn on the screen (or if they are drawn at all). Objects with a low visibility are drawn first.

Index

- 8-bit mask mode 6-7, 6-18

- Absolute P1<P2>, plotter config menu 7-22
- absolute/relative mode, tablet 7-9
- accessories iii
- activating and saving configuration
 - menu fields 7-14
- activating/creating a view 4-27
- active object 4-14
- active view 1-7, 4-27
- active view, redrawing 4-30
- additive and subtractive color models 3-4
- alphanumeric color control 3-25
- alphanumeric cursor 2-4
- alphanumeric video 2-3
- area boundary pen 2-29
- area fill patterns, predefined 2-24
- area fills 2-22
- area fills, polygonal
 - rectangular 2-27
- area shading capability read 8-11
- Areafill Delta field, plotter config menu .. 7-24
- ASCII absolute format 2-9
- ASCII codes for symbols 4-32
- ASCII formats 2-9
- ASCII incremental format 2-10
- ASCII relocatable format 2-10
- attributes, object
 - primitive 1-2, 4-4
 - pseudo object 4-24
- Auto-Centering field, printer config menu 7-20
- auto-redraw mode 4-30
- autotracking 3-22
- auxiliary raster 6-1
- auxiliary raster buffer 1-11

- B&W or halftone dumps 6-24
- background color 3-18, 4-28
- binary formats 2-11
- binary long absolute format 2-15
- binary long incremental format 2-16
- binary long relocatable format 2-17
- binary medium absolute format 2-13
- binary short incremental format 2-16
- bit masks 1-17
- black and white output 3-21
- black and white raster dumps 6-24
- board, color-mapper 1-11
- box cursor 2-4, 2-5

- cameras and external monitors 7-10
- character cell 2-35, 4-32
- character definition space 1-7
- character spacing 2-42
- characters and fonts, user defined 4-31
- characters, user defined 1-7
- Chart Advance field, plotter config menu .. 7-23
- closing the vector list 4-8
- collecting and reporting errors 1-20
- color CRT 3-10
- color definition 3-5
- color graphics 3-1
- color graphics, raster 1-9
- COLOR key 1-9
- color mapper 1-13, 3-11
- color models 3-4
- color notation systems 3-5
- color output modes 3-22
- color palette library commands 5-9
- color palettes 1-2, 1-8
- color parameters 3-15
- color raster dumps 6-24, 7-6, 7-21
- color specification, HSL method 1-9
- RGB method 1-9
- color status 3-18, 8-3
- color transformations B-6
- color, text 2-43
- color-mapper board 1-11
- command, data compaction
 - graphics inquiry 8-1
 - picture file 1-9
 - plotting 2-6
- compaction, data 6-34
- compatibility with other HP graphics terminals iv
- complement mode 4-21
- configuration menu fields, activating
 - and saving 7-14
- configuration menu function keys 7-13
- configuration menu, keyboard 7-15
 - raster 7-15
 - terminal 7-15
- configuration menus, printer and plotter 7-15
- configuration values stored in
 - non-volatile memory 7-14
- Content field, printer config menu 7-19
- controls, tablet 7-8
- copying/examining the picture file 5-9
- creating/activating a view 4-27
- cursor position, read 8-12
- cursor control 2-4
- cursor status, graphics 8-16

Index

- Data compact field, printer config menu . . . 7-21
- data compaction 6-34
- data compaction command 6-25
- data formats, absolute 2-10
 - ASCII incremental 2-10
 - ASCII relocatable 2-10
 - binary 2-11
 - vector 2-8
- data transfers, raster 6-21
- datacomm, raster loads 6-37
- default attributes, view 4-29
- default object attributes 4-23
- defaults, drawing 2-44
 - primitive 4-11
- defining the vector list type 4-7
- delete vector list 5-3
- delimiters 1-16
- device capabilities read 8-13
- device configuration menus 7-12, 8-3
- device error, read 8-11
- device I.D. read 8-11
- device menus, escape sequence operation of . . 7-17
- devices, external 7-1
 - hardcopy 7-1
 - shared 7-1
- display control 2-3
- display cursor 2-4
- display modes 6-18
- display modes and read masks 1-13
- display size read 8-12
- display write mask 1-12
- displayed vector list 4-24
- displayed vector lists 4-7
- double buffer mode 1-13, 4-30
- double buffering 1-14
- drawing defaults 2-44
- drawing functions 2-1
- drawing modes 2-30, 6-37
- drawing modes, vector 6-11

- entering primitives 4-8
- errors 1-18
 - device 8-11
 - execution 1-20
 - sequence 1-18
- escape sequence directory D-1
- escape sequence operation of device menus . . 7-17
- escape sequences, graphics 1-14
 - tablet 7-10
- execution errors 1-20
- Exp mode field, printer config menu 7-19
- Expansion width/height field, printer
 - config menu 7-19
- explicit redraw 4-30

- Ext monitor field 7-11
- extended roman characters 4-36
- external devices 7-1
- external monitors and cameras 7-10
- external video interface 7-10

- features and accessories iii
- fields, configuration menu 7-14
- file, picture 1-1, 4-1
- font, undefined 1-18
- fonts, user defined 1-7
- foreground/background 4-bit mask mode . . . 6-20
- foreground/background 8-bit mask mode . . . 6-19
- foreground/background mode 6-18
- formats, raster load 6-30
 - vector data 2-8
- Formfeed field, printer config menu 7-22
- frame, intersection 6-30
 - source 6-27
 - window 6-28
- freespace, read graphics 8-16
- function keys, configuration menu 7-13

- G AIDS key 1-14
- GID fields 7-25
- glossary E-1
- graphics concepts 1-1
 - cursor 2-4
 - cursor position read 8-12
 - cursor status, secondary 8-16
 - device configuration inquiry 7-26
 - escape sequences 1-14
 - freespace 8-16
 - input devices 7-7
 - inquiry commands 8-1
 - keypad iv
 - label 2-34
 - modification capabilities read 8-15
 - relocatable origin 2-18
 - status 8-6
 - status, sample program 8-6, 8-17
 - tablet 7-7
 - text 2-33
 - text attributes 2-34
 - text mode 2-33
 - text status read 8-14
 - video 2-3
 - zoom 2-3
- grey scale 3-21

- halftone output 3-21
- halftone patterns 4-30
- halftone raster dumps 6-24
- halftone status 3-25

- halftoning 7-21
- hardcopy devices 7-1
- hidden vector lists 4-8
- highlighting 1-6
- highlighting a viewport 4-28
- highlighting, object 4-21, 6-13
- Horiz Limit (Vert Limit) field,
 - raster config menu 7-25
- horizontal intersection frame 6-32
- how to create and activate an object 4-14
- HSL (Hue, Saturation, Lightness)
 - color method 1-9, 3-5
- Hue 3-6

- illegal operand 1-18
- illegal parameter errors 1-18
- implied object inquiry 8-3
- inquire pick command 4-22
- inquiry command 1-17
- inquiry graphics 8-1
 - color 8-3
 - device configuration 8-3
 - implied object 8-3
 - picture file 8-3
 - picked object 8-3
 - raster hardcopy format 8-3
 - vector list 8-3
 - view 8-2
- inquiry, graphics device configuration .. 7-26, 8-3
- installation 7-8
- interface, external video 7-10
 - tablet 7-8
- intersection frame 6-30

- justification/origin, text 2-38

- keyboard configuration menu 7-9, 7-15
- keyboard operation of configuration menus .. 7-13
- keypad, graphics iv
- keys, A AID 7-15
 - ACTIVE VALUE 7-13
 - config keys 7-14
 - DEFAULT VALUES 7-13
 - DSPYFNC 7-14
 - NEXT CHOICE 7-13
 - POWER ON VALUES 7-13
 - PREVIOUS CHOICE 7-13
 - SAVE CONFIG 7-13
 - TEMPSAVE 7-14
- label, graphics 2-34
 - undefined character in 1-18
- lefthand/righthand operation, tablet 7-9
- library commands 5-3
 - library commands and status requests,
 - vector list 4-12
 - view 4-31
 - library commands, color palette 5-9
 - user defined character and font 5-8
 - vector list 5-3
 - view 5-8
 - lift pen 2-7
 - Lightness 3-6
 - line types 2-19
 - line types, user defined 2-21
 - lists, vector 4-3
 - logical output device 6-7
 - look-up table 1-11
 - lower pen 2-7

 - major sequence errors 1-18
 - manipulations, raster 6-1
 - margins, text 2-40
 - mask mode, 8-bit 6-7, 6-18
 - masks, bit 1-17
 - mode, picture protect 5-9
 - transparent 5-2
 - modes, display 6-18
 - move graphics cursor 2-5
 - multiple views 4-30

 - negative numbers 1-17
 - negative scaling 4-16
 - negative text size 2-36
 - non-operatives 1-16
 - non-parameter errors 1-18
 - Number of Pens field, plotter config menu .. 7-23

 - object attribute library commands
 - and status requests 4-25
 - object attribute transformation B-1
 - object attributes, 1-1, 1-4, 4-14
 - default 4-23
 - objects 4-12
 - object highlighting 4-21, 4-28
 - inquiry 8-3
 - library commands 5-7
 - rotation 4-17
 - scaling 4-15
 - transformation center 4-15
 - translation 4-18
 - undraw and redraw 4-24
 - undraw commands 6-12
 - visibility 4-22
 - visibility fence 5-3
 - write mask 4-22, 6-10
 - object, active 4-14
 - how to create and activate 4-14
 - pseudo 4-7, 4-14

Index

- objects, object attributes 4-12
- open vector list 1-3
- operand, illegal 1-18
 - undefined 1-18
- operation, tablet 7-8
- Option field, printer config menu 7-21
 - raster config menu 7-25
- Option Mask field, plotter config menu 7-24
- origin, relocatable graphics 2-18
- origin/justification, text 2-38
- output devices 6-5
- output devices/write masks 1-12
- output, black and white 3-21
 - color 3-22
 - halftone 3-21

- palettes, color 1-2, 1-8
- panning and zooming 1-7
- patterns, user defined 3-23
- pen position read 8-11
- pen selection, alphanumeric 3-29
- Pen Speed field, plotter config menu 7-24
- pen, area boundary 2-29
 - lifting 2-7
 - lowering 2-7
 - primary 1-8, 2-19, 6-13
 - secondary 1-8, 2-19, 6-13
- pens 1-8, 2-19
- pens and palettes 3-12
- pens, alphanumeric 3-25
- pick priority 4-22
- picture file 1-1, 4-1
- picture file commands 1-9
- picture file defaults 5-12
- picture file inquiry commands 5-14, 8-3
- picture file, copying/examining 5-9
- picture protect mode 5-9
- pixel 1-9
- plotter configuration menu 7-5
- plotter configuration menu fields 7-22
- plotters and printers 3-21
- plotting 7-5
- plotting commands 2-6
- plotting limits 7-22
- polygonal area fills 2-28
- predefined area fill patterns 2-24
- Preserve Aspect Ratio field,
 - plotter config menu 7-22
- primary pen 1-8, 2-19, 6-13
- primary pens, selecting 3-17
- primary raster 6-1
- primitive attributes 4-4
- primitive defaults 4-11
- primitives 1-2, 4-4

- primitives, entering 4-8
- printer and plotter configuration menus 7-15
- printer configuration menu 7-2
- printer configuration menu fields 7-19
- printer specifications 6-23
- printers and plotters 3-21
- pseudo object 4-7, 4-14
- pseudo object attributes 4-24

- raster
 - buffer 1-10, 6-1
 - buffer, auxiliary 1-11
 - color graphics 1-9
 - configuration menu 7-15
 - configuration menu fields 7-25
 - data transfers 6-21
 - defaults 6-41
 - dump dimensions 6-23
 - dump orientation 7-3
 - dump/data compaction 6-26
 - dumps 6-21, 6-22, 7-2
 - dumps, color 7-6
 - load formats 6-30
 - loads 6-27
 - loads from datacomm 6-37
 - manipulations 6-1
 - memory 1-9, 3-10, 6-1
 - memory operations 6-2
- Raster Orientation field, printer
 - config menu 7-19
- raster plane 1-10
 - reads 6-21, 6-38
 - device format 8-3
 - status 6-37, 6-41
 - visibility 6-16

- read
 - and write masks 1-17
 - area shading capability 8-15
 - current pen position 8-11
 - device capabilities 8-13
 - device I.D. 8-11
 - display size 8-12
 - graphics cursor position 8-12
 - graphics cursor position with wait 8-12
 - graphics modification capabilities 8-15
 - graphics text status 8-14
 - mask, system 6-17
 - masks and display modes 1-13
 - relocatable origin 8-15
 - reset status 8-15
 - zoom status 8-14
- rectangular area fills 2-27
- Red, Green, and Blue color
 - definition method 6-5

- redraw modes 4-30
- redraw, explicit 4-30
- object 4-24
- redrawing the active view 4-30
- relocatable origin read 8-15
- reset status read 8-15
- RGB color definition method 1-9, 3-5, 3-7
- roman extension font 4-32
- rotation 1-5, B-1
- rotation, object 4-17
- text 2-37
- rubberband line mode 2-8
- Saturation 3-6
- scaling 1-5, B-1
- negative 4-16
- screen viewport 1-5, 1-6, 4-26, B-3
- limits 4-28
- secondary pen 1-8, 2-19, 6-13
- selecting 3-17
- cursor status 8-16
- sequence errors, major 1-18
- setting raster memory 6-3
- shading capability 8-15
- shared devices 7-1
- size, display 8-12
- slant, text 2-38
- source file 6-27
- source frame 6-27
- specifying output devices 6-5
- specifying output devices/write masks 1-12
- status requests and library commands,
- vector list 4-12
- view 4-31
- status, graphics 8-6
- sample programs 8-6, 8-17
- raster 6-41
- subtractive and additive color models 3-4
- symbols 1-7, 2-31
- symbols, ASCII codes 4-32
- system object attributes and the
- pseudo object 4-24
- system read mask 6-17
- system write mask 6-7
- table, look-up 1-11
- tablet controls 7-8
- escape sequences 7-10
- interface 7-8
- operation 7-8
- tablet, absolute/relative mode 7-9
- graphics 7-7
- lefthand/righthand operation 7-9
- terminal configuration menu 7-15
- terminal configuration menu fields 7-26
- text, graphics 2-33
- text attributes, graphics 2-34
- color 2-43
- justification/origin 2-38
- margins 2-40
- origin 4-35
- rotation 2-37
- size 2-35
- size, negative 2-36
- slant 2-38
- status 8-14
- thumbwheels and Gid key 7-7
- transfers, raster data 6-21
- transformation attributes 4-14
- center 1-5, B-1
- equations B-1
- translation 1-5, B-1
- translation, object 4-18
- transparent mode 5-2
- two's complement 1-17
- undefined character in label 1-18
- undefined font 1-18
- undefined operand 1-18
- undraw commands, object 6-12
- undraw, object 4-24
- Use Absolute field, plotter config menu 7-22
- user defined area fill patterns 2-24
- user defined character and font
- library commands 5-8
- user defined characters and fonts .. 1-2, 1-7, 4-31
- user defined font library commands 4-37
- user defined line types 2-21
- user defined patterns 3-23
- vector data formats 2-8
- vector drawing modes 6-11
- vector list, 1-1, 1-2
- "header" 4-6
- elements 4-4
- inquiry 8-3
- library commands 5-3
- library commands and status requests .. 4-12
- primitive attributes 4-5
- structure 4-6
- type, defining 4-7
- vector list, closing 4-8
- delete 5-3
- displayed 4-24
- vector lists 4-3
- vector lists, displayed 4-7
- hidden 4-8
- vector space 1-3

Index

- vector space 1-2
- vectors 2-6
- vertical intersection frame 6-32
- video, alphanumeric and graphic 2-3
 - external interface 7-10
- view 1-1
- view attributes 1-6
- view default attributes 4-29
- view library commands 5-8
 - commands and status requests 4-31
- view inquiry 8-2
- view, activating/creating 4-27
 - active 1-7, 4-27
- viewing transformations B-3
- viewport background color 1-6, 4-28
- viewport highlighting 4-28
- viewport, screen 1-6, 4-26
- views 1-5, 1-6, 4-25
- views, multiple 4-30
- virtual cursor 2-4
- virtual space 1-4
- virtual window 1-5, 1-6, 4-25, B-3
- virtual windows 4-28
- visibility fence, object 5-3
- visibility, object 4-22
 - raster 6-16
- window frame 6-28
- window, virtual 1-6, 4-25
- write mask, object 4-22, 6-10
 - system 6-7
- write masks/output devices 1-12
- writing to raster 6-4
- x resolution field, horizontal**
 - thumbwheel 7-25
- y resolution field, vertical**
 - thumbwheel 7-25
- zoom center 1-7, 2-3, B-4
- zoom factor 1-7, B-4
- zoom status read 8-14
- zooming and panning 1-7