SUPERSESSION/UPDATE INFORMATION: This document supersedes the document of the same name, Order No. AA-5428B-TC.

OPERATING SYSTEMS AND VERSIONS:
RT-11 V3
RSX-11M V3.1
RSX-11D V6.2
IAS V2.0

SOFTWARE AND LANGUAGE VERSIONS:
DECgraphic-11 FORTRAN Graphics Package (QJ647) V1.1
FORTRAN IV V2
FORTRAN IV-PLUS V02

DECGRAPHIC-11

FORTRAN Programming Manual

Order No. AA-5428C-TC

March 1978

This manual describes the new DECgraphic-11 FORTRAN Graphics Package for the VS60 and VT11 graphic display subsystems, usable in the RT-11, RSX-11M, RSX-11D, and IAS operating systems.

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, July 1977
Revised, March 1978

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

CONTENTS

FIGURES

TABLES

PREFACE

DECgraphic-11 is the family name for PDP-11 graphic products using the VT11 and VS60 graphic-display subsystems. Each subsystem consists of a display processor and a cathode-ray-tube (CRT) display screen. The display processor can be on the UNIBUS. In this case it functions as an autonomous processor for handling graphic instructions, thereby freeing the central processor of much of the burden of running the graphic program. As an alternative to this stand-alone system, the display subsystem can also serve as an intelligent terminal in a multiprocessor, or host-satellite, graphic system.

This manual is intended for DECgraphic-11 users who are familiar with PDP-11 FORTRAN under the RT-11, RSX-11M, RSX-11D, or IAS operating systems.

The DECgraphic-11 FORTRAN Graphics Package controls the VS60 or VT11 display hardware with subroutines that are called from your FORTRAN graphic program. The subroutines are easy to use, and graphic programming with DECgraphic-11 involves a minimal number of new concepts that a FORTRAN programmer must master. Anyone who has written PDP-11 FORTRAN programs and called subroutines to perform discrete functions can begin writing graphic programs from the outset, at least on an elementary level.

This manual describes all subroutines available in the DECgraphic-11 FORTRAN Graphics Package and gives instructions on their use in graphic programming. There are many sample programs and displays that show the precise effect of individual subroutines and hardware features. The manual also contains suggestions for making graphic programs more efficient, and instructions for constructing your graphic subroutine libraries from the sources supplied in the software distribution kit. However, the manual does not provide comprehensive information on the FORTRAN language or on the system resources of RT-11, RSX-11, or IAS.

ASSOCIATED DOCUMENTATION

PDP-11 FORTRAN Language Reference Manual
DEC-11-LFLRA-C-D

RT-11/RSTS/E FORTRAN IV User's Guide
DEC-11-LRRUA-A-D

RT-11 System Reference Manual
DEC-11-ORUGA-C-D, DN1, DN2

IAS/RSX-11 FORTRAN IV User's Guide
DEC-11-LMFUA-C-D

FORTRAN IV-PLUS User's Guide
DEC-11-LFPUA-B-D, DN1

RSX-11M Operator's Procedures Manual
AA-2567D-TC

RSX-11 Utilities Procedures Manual
DEC-11-OXMDA-A-D

RSX-11D User's Guide
DEC-11-OXDUA-B-D

IAS User's Guide
DEC-11-OIUGA-B-D

IAS Editing Utilities Reference Manual
DEC-11-OIEUA-A-D, DN1

VT11 Graphic Display Processor Manual
EK-OVT11-TM-001

CHAPTER 1

THE DECGRAPHIC-11 SYSTEMS

1.1 INTRODUCTION TO DECGRAPHIC-11

The DECgraphic-11 FORTRAN Graphics Package is a set of FORTRAN-callable subroutines that you can use in regular FORTRAN programs written for the PDP-11 computer. When these programs are run on a PDP-11 configuration that includes DECgraphic-11 display hardware, the DECgraphic-11 subroutines produce graphic images on the display screen. This manual gives full descriptions of the subroutines and their uses, so that you can precisely control the appearance of the graphic display.

This chapter provides an overview of the manual, describes the documentation conventions used in the presentation of subroutines, introduces the features of the DECgraphic-11 display hardware, and discusses the unique features of graphic programming in DECgraphic-11 FORTRAN.

1.1.1 Overview of the Manual

This manual decribes the use of the DECgraphic-11 FORTRAN Graphics Package in RT-11, RSX-11M, RSX-11D, and IAS. The first three chapters are relevant to programming in all four operating systems.

Chapter 1 introduces basic concepts you need for understanding DECgraphic-11 programming. It defines new terms used in the manual, discusses the characteristics of the VT11 and VS60 display hardware, and summarizes the most important programming features of the DECgraphic-11 FORTRAN Graphics Package.

Chapter 2 describes the DECgraphic-11 subroutines in detail, along with examples of their use. Most of the subroutines are used interchangeably for the VT11 and VS60 displays. Where differences exist--for example, where subroutines use special VS60 hardware features--these differences are clearly distinguished.

Chapter 3 suggests general programming techniques for making your graphic programs more efficient.

Chapter 4 contains start-up instructions for RT-11 programmers. It explains how to build a library of graphic subroutines from the DECgraphic-11 distribution kit, and how to link your finished graphic programs to the library.

Chapter 5 contains instructions for RSX-11M, RSX-11D, and IAS programmers. Chapter 5 also contains instructions for building graphic libraries and for creating graphic task images in these three systems. Two sets of instructions are included for RSX-11M

programmers, since they can use DECgraphic-11 software in either a stand-alone PDP-11 configuration or in a multiprocessor (host-satellite) system. In IAS and RSX-11D, DECgraphic-11 software is always used in a host-satellite mode. Section 5.3 discusses the host-satellite software.

In addition, a series of appendixes summarizes important information for quick reference.

Appendix A alphabetically lists the FORTRAN subroutine calls and briefly describes the function of each.

Appendix B summarizes the error messages that are generated by the DECgraphic-11 package, the reasons for their occurrence, and the routines that might produce these errors.

Appendix C discusses the internal format of the DECgraphic-11 display file as it is constructed in memory.

Appendix D contains FORTRAN programming examples developed to demonstrate the varied capabilities of DECgraphic-11 interactive graphics.

Finally, a glossary of graphic terms defines the special words and concepts used in this manual.

1.1.2 Documentation Conventions

The following list describes the documentation conventions used in describing the FORTRAN subroutine calls in this manual.

Convention	Meaning
Brackets ([ ])	The enclosed arguments are optional.
FORTRAN variable name with uppercase letters (X11, I0, M2)	Standard default FORTRAN variables whose values are returned by the subroutine; alternately, an array name passed to the subroutine.
FORTRAN variable name with lowercase letters (x11, i0, m2)	Standard default FORTRAN variables whose values are to be supplied by the user; can be any valid arithmetic expression of the specified type (e.g., x is real, i is an integer).
l, i, f, t	Special display parameters that are all integers; these parameters are the only exceptions to the rules described above.
x axis	The horizontal axis.
y axis	The vertical axis.
RET	Symbol for a RETURN, Carriage Return, or <CR> key.
CTRL/X	Indicates the CTRL key and another key (x) on your terminal. To perform some functions, you must press the CTRL key and the other key (represented here by x) at the same time.

SPECIAL DOCUMENTATION NOTES

1. All real constants that you supply as arguments in calls to the FORTRAN subroutines must have decimal points. If you do not adhere to this convention, the results of program execution are unpredictable.

2. DECgraphic-11 subroutine libraries can be specially built to take only integer arguments. In that case all arguments shown in the subroutine descriptions of Chapter 2 are integers (see Chapters 4 and 5).

3. In "dialogs" with the computer (see Chapters 4 and 5), the parts typed by the user are printed in red ink.

4. Each subroutine description in Chapter 2 gives the calling sequence of the subroutine. When the notation "(h-s)" follows a subroutine name, it means that the subroutine is used only in host-satellite systems (see Section 5.3.2).

1.1.3 Summary of DECgraphic-11 System Capabilities

The following list summarizes the major features of the DECgraphic-11 interactive graphic system. Graphic terms such as beam, vector, and menu are defined in a glossary at the end of this manual. They are explained in greater detail in Section 1.3.

Support of varied display elements

You can display a variety of elements on the screen, including points, line segments, characters, and graphs. Elements are normally defined at coordinate positions relative to the current beam position, although points can be defined at absolute coordinate positions, as can line segments on the VS60. The display elements are called "primitives" in this manual.

Variable line types

Vectors can be drawn on the screen in any of four formats: solid line, long-dash line, short-dash line, or dot-dash line. These line types are illustrated in Figure 1-1.

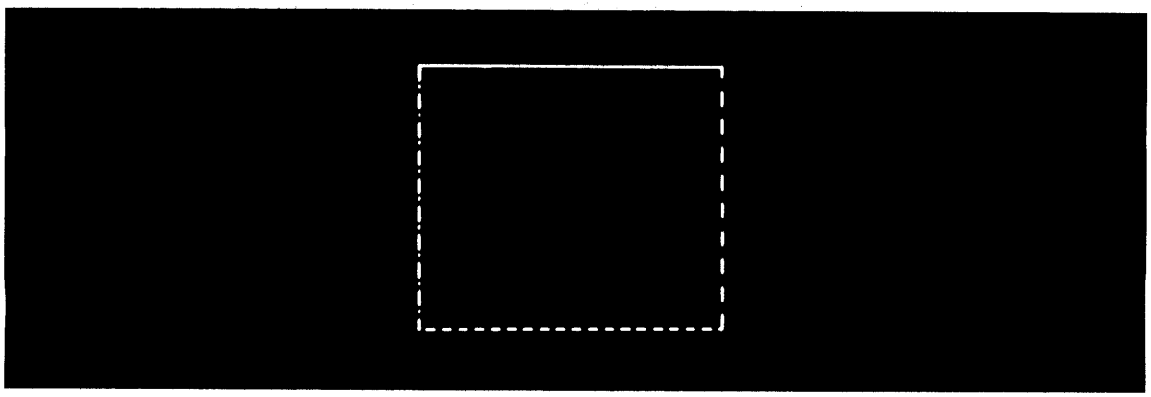

Figure 1-1 Line Types

Variable intensity level

You can vary the brightness of pictures or components of pictures on the display screen. In this way, you can emphasize portions of the display. There are eight different levels of brightness, from 1 (faintest) to 8 (brightest), as shown in Figure 1-2.

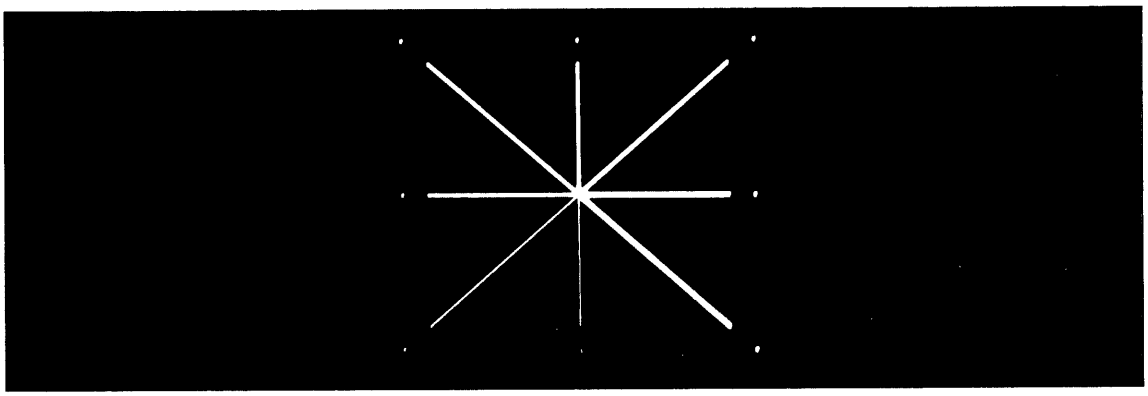

Figure 1-2 Intensity Levels

Hardware blink feature

You can specify that a display or a portion of a display blink on and off. This feature is useful for warnings or for identifying a particular picture component to be changed or highlighted.

Italic mode

Characters can be displayed in ordinary type or in italics. You can specify the text to be italicized by including a special control code before the character-string specification. Examples of the two available type fonts are shown in Figure 1-3.

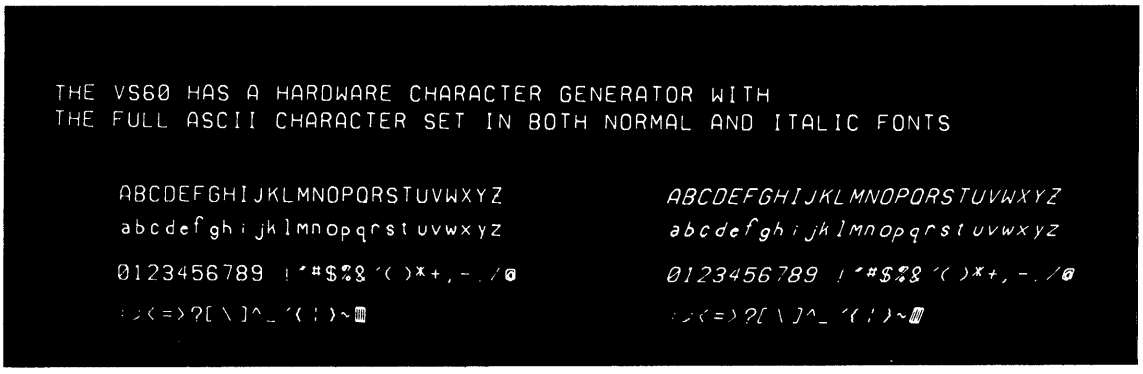

Figure 1-3 Type Fonts

Extended character set

You can display any of the 96 conventional ASCII characters and any of 31 additional characters representing Greek letters and mathematical symbols. To select the additional characters from this extended set, you must include a "shift-out" code before the

ordinary ASCII character string to which the shift-out character string corresponds. Extended characters can be displayed in either of the available type fonts, as shown in Figure 1-4.

Figure 1-4 Extended Characters

A list of extended characters and their ASCII character correspondences is included in Section 2.3.7.

Rotated characters

On the VS60, a character string can be displayed in the normal horizontal position, or it can be rotated 90 degrees counterclockwise. As in the specification of italic mode, you can select rotated characters by including a special control code before the character string to be rotated. Examples of normal and rotated characters are shown in Figure 1-5.

Figure 1-5 Rotated Characters

Subscripts and superscripts

On the VS60, you can display text that includes subscripts and superscripts on the screen--for example, in the expression shown in Figure 1-6.

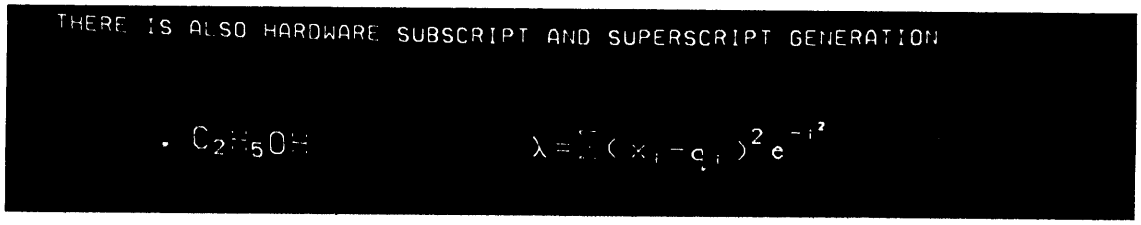

Figure 1-6 Subscripts and Superscripts

As with other special character modes, you specify a subscript or superscript by including a special control code before the character string to be displayed.

Character and vector scaling

On the VS60, you can display characters and lines on the screen in the standard size and you can enlarge or compress them as desired. The normal size of each character is 6 by 8 raster units, and each character is displayed in a space 14 raster units wide and 24 raster units tall. In other words, characters displayed side-by-side have a spacing of 4 units on either side and 16 units between lines.

NOTE

A "raster unit" is the smallest resolvable point on the display screen of either a VT11 or VS60.

On the VS60, you can change the standard character size to display characters that are one-half normal size, one and one-half normal size, or twice normal size, as shown in the example of Figure 1-7. Vectors and other images can be scaled in increments of one fourth, from one-fourth normal size to three and three-fourths normal size.

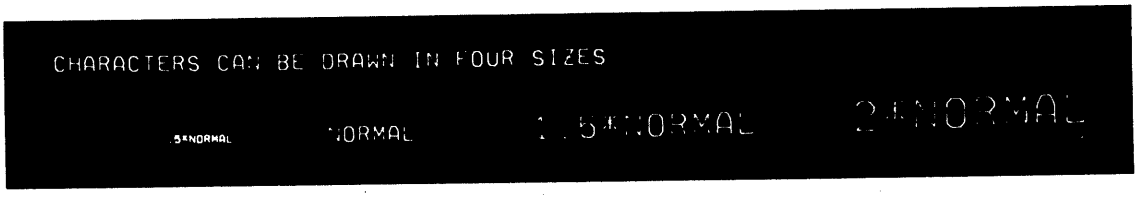

Figure 1-7 Character Scaling

Variable coordinate systems and viewports

On the VS60, you can define the screen to function as a "viewport" on a much larger image definition area (see Section 1.3). You can vary the position of this viewport to examine different areas of the total image. You can also define your own coordinate system on either a VT11 or VS60.

Light-pen support

You can interact with the displayed image by using the light pen to select options, to identify pictures to be moved, or to otherwise manipulate displays on the screen. Support of the light pen on the VT11 and VS60 and the light-pen tip switch on the VS60 facilitates the dynamic alteration of displays on the screen and makes DECgraphic-11 an interactive system.

Programmable pushbutton box

You can use a 16-key pushbutton box as an optional interactive tool. Each button can be programmed to represent a menu item, subpicture, or any other graphic operation to be performed.

Keyboards as interactive tools

Using the standard software, you can program individual keys on your keyboard to represent menu selections or other graphic operations.

Menu area

You can display a special menu of character strings as a formatted list on the screen, and you can select menu items by touching them with the light pen. You can specify your own menu area or, on the VS60, use the hardware menu area described in Section 1.3.2.

Subpicture support

You can group together a series of calls, data, and picture components to form an entity called a "subpicturE," which can then be displayed, copied, erased, and turned on and off. Repeated displays of identical images can be defined with greater modularity and efficiency by using subpictures.

1.2 HARDWARE AND SOFTWARE REQUIREMENTS

This section describes the minimum hardware and software requirements for graphic programming with the DECgraphic-11 FORTRAN Graphics Package.

1.2.1 Generating an Operating System

The fundamental software requirement is a functioning operating system: RT-11, RSX-11M, RSX-11D, or IAS. RT-11 can be used only for stand-alone systems. RSX-11D and IAS can be used only for host-satellite systems. RSX-11M can be used for either type.

IMPORTANT NOTES

- If you are using a mapped RSX-11M system for stand-alone graphics with a VT11, the FORTRAN COMMON area containing the DECgraphic-11 display file (COMMON DFILE) must be located in the physical lower 28K of memory. (This restriction does not apply if you are using a VS60.) It is suggested that you set up a global common of adequate size for this purpose when you generate your RSX-11M system. As is described in Chapter 5, this global common must be specified as a TKB option whenever you create a graphic task. For details on creating global commons (also called "shared regions") and on the COMMON option of TKB, consult the RSX-11M System Generation Manual and the IAS/RSX-11 FORTRAN IV User's Guide, respectively. This restriction does not apply to unmapped RSX-11M systems or to any host-satellite system. The RSX-11M Task Builder Reference Manual describes global commons (shared regions) in detail.

- For stand-alone RSX-11M systems, the AST (asynchronous system trap) option must be selected when the system is generated. See the RSX-11M System Generation Manual for details.

- For all RSX-11M systems, the READ/WRITE TRANSPARENT option must be selected when you generate the operating system.

- For all RSX-11M systems, you must answer the system-generation question WHAT IS THE SIZE OF PHYSICAL MEMORY (1024,-WORD BLOCKS)? with a figure that is less than or equal to 124 (decimal).

- If, when you are generating an RSX-11M system, the question DO YOU WANT THE BASELINE TERMINAL DRIVER? is asked, you must answer N.

- If you are using the LK-11 pushbutton box on a stand-alone RSX-11M system, you must also install its driver at system-generation time. See Chapter 5 for more details.

- When generating an RSX-11D system, you must include SP and BP devices so that the BATCH feature will work properly. The SP device must be redirected to a disk at runtime. See the RSX-11D System Generation Manual.

1.2.2 Hardware for Stand-Alone Systems

The following hardware is required for the DECgraphic-11 FORTRAN Graphics Package in a stand-alone system:

- PDP-11 central processing unit (CPU) with 16K or more of memory
- VS60 or VT11 display subsystem
- user terminal
- disk or another random-access system storage device for use by the operating system

The amount of memory required to support the graphic subsystem depends on programming requirements, but normally ranges from 8K to 16K.

In a stand-alone configuration (all RT-11 and some RSX-11M systems), the VS60 and VT11 are supported as autonomous processors that are attached as UNIBUS peripherals to the PDP-11. Both processors are direct-memory-access devices.

1.2.3 Additional Host-Satellite Hardware Requirements

THE DECgraphic-11 FORTRAN Graphics Package can also be used in a host-satellite system, with the following changes and additions to the minimum hardware:

- PDP-11 host CPU with sufficient memory to run a mapped RSX-11M, RSX-11D, or IAS operating system
- GT41 or GT43 graphic terminal (VT11) or GT62 graphic terminal (VS60) for use as the satellite
- serial line interface connecting the host computer and satellite terminal

A DECgraphic-11 program running in the host passes graphic commands to the satellite via the terminal handler (or driver) of the host operating system. These commands create display files in the satellite's memory, and thus display images on the satellite screen. The satellite CPU can also be programmed to handle graphic attentions from an interactive satellite display so that the host computer is not burdened.

1.2.4 Baud Rates

If you are using RSX-11M, RSX-11D, or IAS for a host-satellite system, the host computer and satellite terminal communicate through the host's terminal handler (or driver). You must set the transmission and reception speeds (baud rates) of the host-satellite communication to the values given in the following tables:

RSX-11M:

	HOST	SATELLITE
SEND	9600	4800
RECV	4800	9600

RSX-11D, IAS:

SEND	9600	2400
RECV	2400	9600

1.2.5 General Software Requirements

Once you have generated an operating system (see Section 1.2.1), the only additional software requirements are the DECgraphic-11 FORTRAN Graphics Package and PDP-11 FORTRAN. The FORTRAN Graphics Package is distributed in source form, with sections written in PDP-11 FORTRAN and in MACRO-11 assembly language. The same basic distribution kit can therefore be compiled and built into libraries on any of the four operating systems, including FORTRAN IV-PLUS compilation on systems with that feature.

In host-satellite systems, you must use the FORTRAN IV compiler to compile the satellite software (see Chapter 5). If the FORTRAN IV compiler is not included in the standard system-generation kit for your operating system and you are using a host-satellite system, be sure to add this compiler as a special option.

When you install FORTRAN on your operating system, you must enter a particular FORTRAN library in the system library (SYSLIB). The FORTRAN library you choose will then be used by default when the Task Builder creates a new task image. Some FORTRAN libraries that you may choose assume the presence of special hardware, such as a floating-point processor or EIS. If you choose such a library for your default FORTRAN library, you must also keep the "no hardware" library (FORNHD.OBJ) on your system device. The reason is that your satellite computer may not match the hardware features of your host computer--for example, the PDP-11/10 computer in a GT41 terminal does not have the EIS feature. If you attempt to build a satellite control task (see Chapter 5) with an EIS FORTRAN library, the Task Builder will generate EIS instructions that are illegal on a PDP-11/10. You should specify FORNHD explicitly when you build a satellite control task for a GT41. The indirect command file GRSBLD, which builds a new satellite control task (see Chapter 5), uses the default FORTRAN library. If your choice of the default library does not match the satellite hardware, you should edit GRSBLD to insert an explicit reference to FORNHD in the task-builder command string.

1.3 BASIC GRAPHIC CONCEPTS

This section describes basic concepts that you need to know before you program with DECgraphic-11 FORTRAN. The discussion of DECgraphic-11 features in this section focuses primarily on hardware, such as the display screen and the light pen.

1.3.1 The Display Screen

Two graphic-display subsystems can be used with DECgraphic-11: the VS60 and the VT11. Both graphic subsystems have "refreshed" CRTs and solid-state light pens. The VS60 display processor can optionally support a second CRT.

The display screen has a square main viewing area that measures 9 1/4 by 9 1/4 inches (20.74 by 20.74 centimeters) on the VT11 and 12 by 12 inches (30,48 by 30.48 centimeters) on the VS60. Tne viewing area capacity, when normal-size characters are produced, is 73 characters per line and 31 lines per screen.

For the VS60, this viewing area can be considered a viewport on a larger image definition area. The image definition area that can be addressed on the VS60 is a 96-by-96-inch (243.84-by-243.84-centimeter) square. You can change the position of the viewport to make different parts of the image definition area appear on the main viewing area of the VS60 screen (see Section 2.2.5).

Both the x axis and the y axis of the VS60 and VT11 viewing areas consist of 1024 individual points (1777 octal). Thus, there is a total of 1,048,576 individually addressable positions on the display screen.

The image definition area on the VS60 has 8192 individual points on each axis. Therefore, this area consists of 67,108,864 individually addressable positions. In defining points, lines, and other elements on the VS60 display screen, your coordinate specifications may normally extend beyond the bounds of the viewport. Any display that extends beyond the bounds of the viewport will appear to be truncated at the edge of the viewport, but will actually extend into the remaining image definition area. However, if you address positions that fall outside the image definition area, the display will be truncated at the edge of the image definition area.

Figure 1-8 illustrates the relationship of the DECgraphic-11 viewport and image definition area on the VS60.

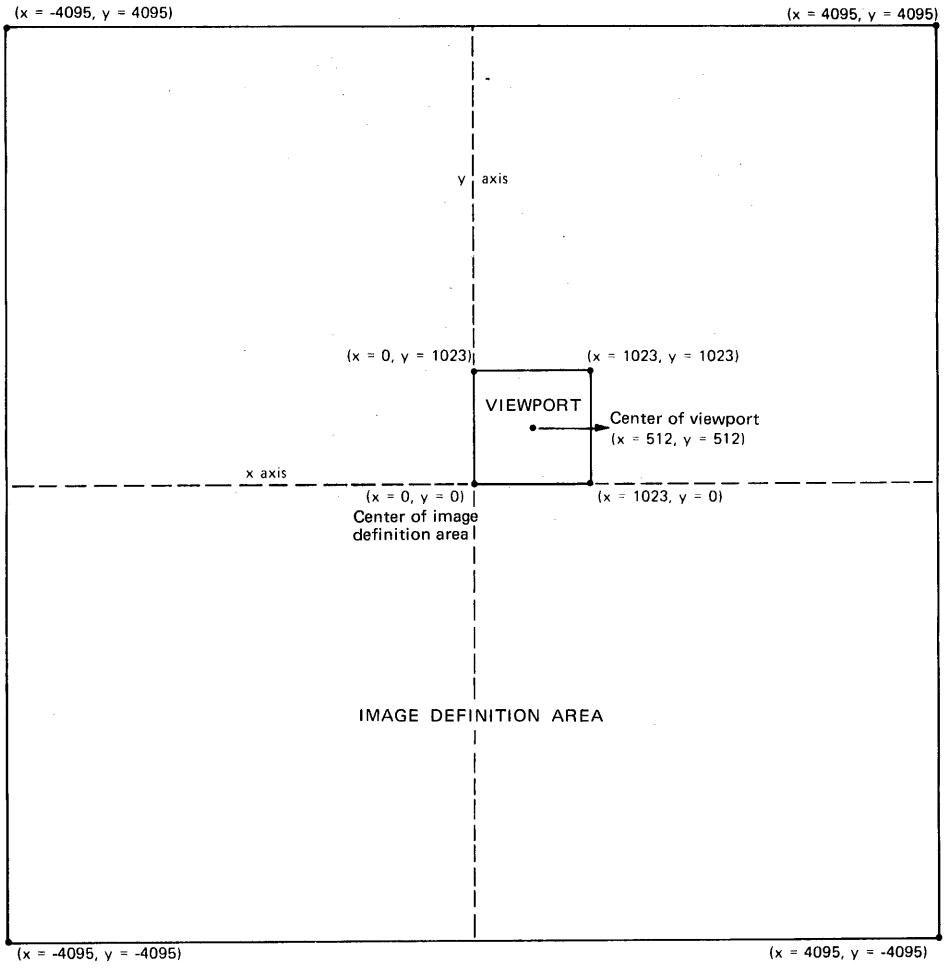

Figure 1-8 VS60 Display Screen

1.3.2 **Menus**

In addition to the main viewing area, the VS60 display screen has a separate menu area. The main area is the 12-by-12-inch screen discussed above, and the menu area is a vertical strip on the right side of the screen, measuring 1 1/2 inches (128 raster units) by 12 inches (4 by 30 centimeters). The menu area can be used for any purpose, but it is usually most helpful in displaying a list of options called a menu, from which the user of your application program can choose. An example of such a display is shown in Figure 1-9.

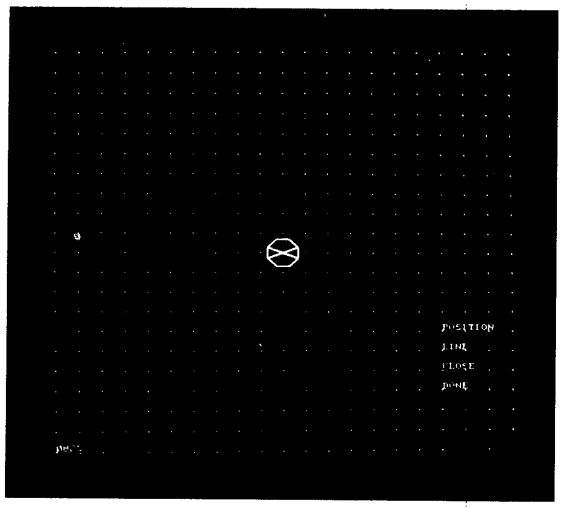

Figure 1-9 Menu Area

The display types listed here represent the choices available to the user of the application program. For example, to draw a line with the light pen, the user touches the LINE label in the menu area with the light pen.

You can construct a menu by calling the MENU subroutine (see Section 2.3.8). This subroutine allows you to specify the character strings to be displayed as menu items, to assign a name or tag to each of these items, and to specify the spacing between the items in the menu area.

You can position a menu area at any location on the screen by including in the MENU call the coordinate positions at which the menu will begin. If you do not specify these coordinates, the menu is constructed in the separate menu area to the right of the main viewing area.

With VT11 subsystems, no separate menu area is built into the hardware, but the MENU subroutine still constructs the menu on the right side of the display screen. In this case, the menu is superimposed on any objects displayed at the extreme right end of the VT11 viewing area.

The subroutine AREA, which is used only with VS60 subsystems, allows you to display any small object or text string in the hardware menu area (see Section 2.2.2).

1.3.3 Coordinate Systems, Windows, and Viewports

The display screen of a VT11 or VS60 subsystem is a "window" on a plane of real-valued coordinates. In the default case (when the WINDW subroutine has not been called), the window defined by the screen has its lower left corner at the coordinate position (0.,0.) and its upper right corner at (1023.,1023.).

There are always 1024 addressable points, or "raster units," on a VT11 or VS60 screen. Therefore, the default window defines a "unit-scaled" coordinate system, in which each axis coordinate corresponds to one raster unit. In this unit-scaled system, the first three raster units in the x direction correspond to the coordinate positions (0.,0.), (1.,0.), and (2.,0.).

With the WINDW subroutine (see Section 2.2.3), you can change the default window to any other window you choose. The operation of the WINDW subroutine is analogous to approaching or backing away from a window in your home; the size of the window itself remains the same, but the portion of the outside world that you observe through the window changes, by an equal amount in the horizontal and vertical directions. On a VT11 or VS60 display screen, this operation changes the range of coordinates on the x and y axes. The ratio of this new range (0-1023) is called the "scaling factor."

For example, the call

```
CALL WINDW (0.,0.,511.,1023.)
```

leaves the lower left corner of the window unchanged, but changes the coordinates of the upper right corner to (511.,1023.). The scaling factor of the x axis is now .5; the scaling factor of the y axis remains 1. Note that, unlike the case of backing away from the window in your home, this operation changes the aspects of the x and y axes by unequal amounts. This feature of separate scaling factors is useful when, for example, the y values of your data fall within a much smaller range than the x values. You can still spread the y values across the full height of the screen to make them more legible.

In the case of the VS60, the display screen can be defined more exactly as a "viewport" on a much larger window called the "image definition area." The VS60 has 8192 addressable positions on each axis of the image definition area. Of these positions, 1024 are visible in each axis direction at any given time in the viewport. With the VIEWPT subroutine (see Section 2.2.5), you can move the viewport anywhere on the image definition area.

The WINDW subroutine has the same visible effect on the VS60 viewport as it has on the VT11 display. The scaling factors created by WINDW, because they affect the entire addressable window, also affect the parts of the image definition area that are not currently visible in the VS60 viewport.

Figure 1-10 shows the relationship between the viewport and the image definition area (window) in a VS60 subsystem. In this figure the default window is in use, so that all points from (-4095.,-4095.) are inside the window. In this case, the area labeled "Old Viewport" has the same aspect as the entire screen of the VT11. The area labeled "New Viewport" results from a call such as

```
CALL VIEWPT (-4095.,-4095.)
```

After this VIEWPT call, which redefines the lower left corner of the viewport, the new viewport will be visible on the VS60 screen.

The information in this section can be summarized as follows:

1. The viewport and window are always identical on a VT11. Therefore, the VIEWPT subroutine has no effect on the VT11.

2. The window, or image definition area, of a VS60 has 64 times the number of addressable positions as are available with the VT11, although the same number of points are visible on both subsystems at any given time.

3. The WINDW subroutine has the same effect on the visible aspects of both subsystems, but the scaling factors defined by a WINDW operation also affect the unseen portions of the VS60 image definition area.

4. For a more detailed discussion of the WINDW and VIEWPT subroutines, see Sections 2.2.3 and 2.2.5, respectively.

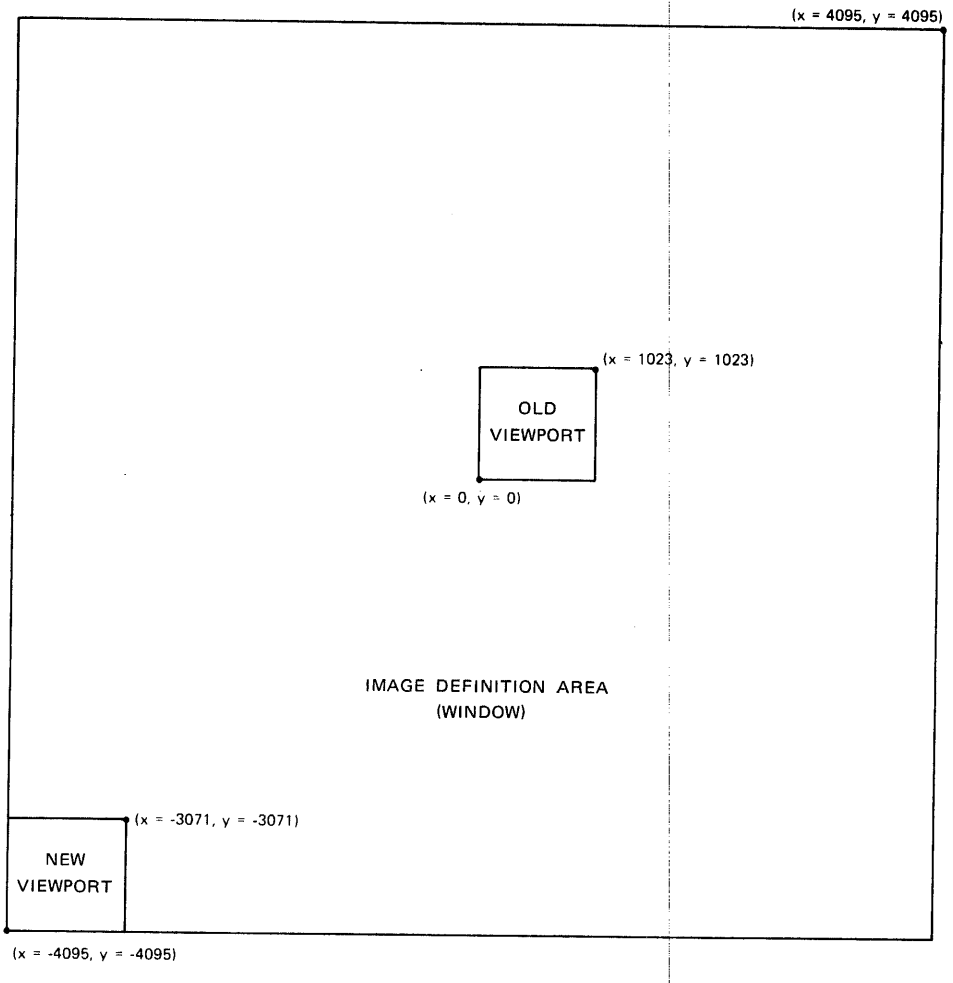

Figure 1-10 VS60 Window and Viewports

1.3.4 Interaction and Tracking

The DECgraphic-11 FORTRAN Graphics Package is an interactive graphic system. It allows you to display a wide variety of graphic elements on the display screen, to change these elements dynamically, and to interact with the system.

To facilitate this user interaction, the system supports a light pen, a solid-state light-detecting device that can be pointed at any display element on the screen. If a primitive has been made sensitive to the light pen, a PDP-11 interrupt occurs when the pen touches the displayed primitive on the screen. This specific interrupt is known as a "light-pen hit." The display-processor hardware retains information on the characteristics and coordinate positions of the hit.

In general, signals that allow you to select particular objects on the display and thus trigger some programmed action are called "graphic attentions."

The VS60 light pen also creates a second type of graphic attention with its tip switch, whose status (in or out) is set when you press or remove the tip from the screen. This distinct graphic attention provides a second kind of control in interacting with the display. For example, the tip switch can be used to confirm detection of a particular primitive or menu item. You might also perform a certain action after a light-pen hit with the tip switch on, and an alternative action after a light-pen hit with the tip switch off.

The light pen is often used to select an element from the menu area of the display screen. The application program can display a list of options, as shown in Figure 1-11.

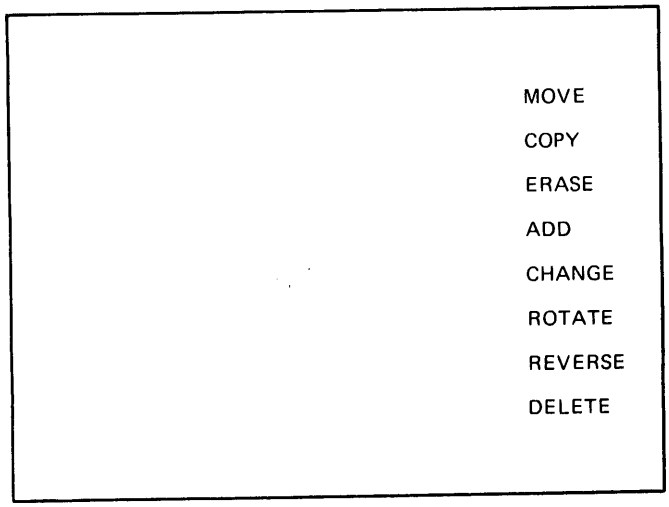

Figure 1-11 Light Buttons

Each of the labels displayed in the menu area of the screen is known as a light button when it is sensitive to the light pen. If you touch the ADD element with the light pen, for example, the appropriate information about the position of the light pen will be returned to the FORTRAN program, and you can initiate a program branch to the appropriate FORTRAN statements. For details on the returned information, see the description of LPEN (Section 2.8.1).

The coordinates of a light-pen hit will be returned with an accuracy of precisely one raster unit for the VT11 and approximately four raster units for the VS60.

The interactive nature of the DECgraphic-11 system is increased by support of a tracking object. The tracking object is an octagonal image that can be displayed to keep track of light-pen hits. It can be moved freely around the screen to follow the light pen.

Figure 1-12 illustrates the tracking object.

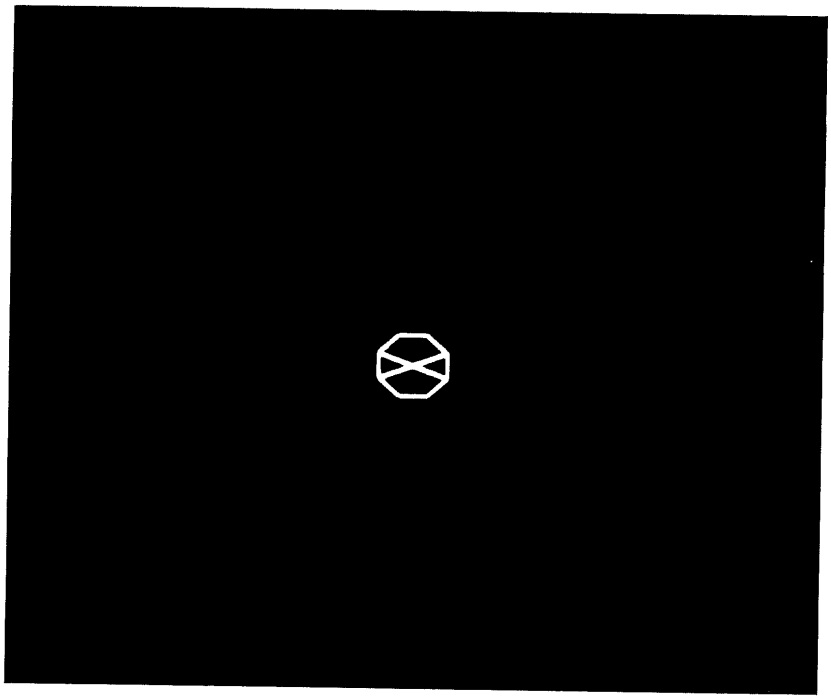

Figure 1-12 Tracking Object

Initially you can place the tracking object at any location on the display screen with the TRAK subroutine (see Section 2.8.2). When the light pen hits the tracking object, the object automatically centers itself on the hit.

You can alter such primitives as points and vectors by attaching them to the tracking object, moving the tracking object and the attached primitives, and then detaching the primitives (see Sections 2.8.3 and 2.8.4).

You can also define an invisible grid of points that are evenly spaced at user-defined intervals on the display screen. The tracking object can then be moved to the nearest point on the grid.

On VS60 display subsystems that have two display scopes, each scope has its own light pen. Since the LPEN subroutine can distinguish light-pen and tip-switch hits on one scope from hits on the other scope, two users can interact separately with the displays.

The light pen is not the only interactive device that you can use with the DECgraphic-11 FORTRAN Graphics Package. Another interactive device that you can use is the LK-11 pushbutton box, which is an optional feature with all DECgraphic-11 display subsystems. This small keypad has 16 keys, each with an internal light. With the subroutines PBS, PBH, and PBL (see Section 2.10), you can program each of these 16 keys to represent menu items, subpictures, or even large-scale graphic functions that will be immediately executed when their keys are pressed. With the LK-11 you can program for at least 16 graphic attentions that are either the same as or distinct from the attentions created by the light pen.

With the subroutine KBC (see Section 2.11.1), you also can use the keyboard of your programming console as an interactive device. KBC allows you to associate each keyboard character with a menu item, subpicture, or graphic function. Because of the large number of distinct ASCII codes on a typical keyboard, KBC extends the potential number of graphic attentions that a single program can use. KBC is a standard feature that comes with all DECgraphic-11 FORTRAN Graphics Package kits and can be used with any hardware configuration.

Finally, the subroutine GRATTN (see Section 2.9) can "poll" all the interactive devices used in a given program by responding to the graphic attentions they create. Unlike LPEN, PBH, and KBC, GRATTN can make the program wait until an attention occurs and can then identify the attention source. In programs with more than one interactive subroutine, GRATTN is extremely useful in simplifying the program logic. Furthermore, GRATTN is not limited in the number of devices that it can poll; it can perform its functions no matter how many new interactive devices are added to your system.

As with all DECgraphic-11 features, these interactive features are available on all four operating systems. However, when you are programming a host-satellite system (IAS, RSX-11D, and some RSX-11M systems), you may have to consider the large number of graphic attentions that an interactive program will create. Your FORTRAN program must contain subroutine calls to handle each type of graphic attention, and the host and satellite computers must normally communicate over their data link for each attention. DECgraphic-11 can reduce much of the time-consuming host-satellite communication by using the satellite computer, rather than the host, to handle your program's attentions.

Appendix D contains a FORTRAN listing of the program DRAW. This program demonstrates virtually all features of the DECgraphic-11 FORTRAN Graphics Package being used in an interactive program. DRAW.FOR is supplied with your DECgraphic-11 software kit, and can be compiled, linked, and run on any of the four operating systems, with either a VT11 or VS60 display subsystem. If this version of DRAW is run on a host-satellite system, the host computer will handle graphic attentions. Another example in Appendix D shows DRAW slightly modified so that the attentions can be handled by the satellite CPU.

In summary, each of the interactive devices--the keyboard, pushbutton box, and light pen--creates graphic attentions for your interactive program. You can think of graphic attentions as logical signals that tell the FORTRAN program to perform some specific function. A single key stroke, pushbutton stroke, or light-pen hit can be programmed to represent any graphic procedure, from a simple addition of some new primitive to a complex rearrangement of the total picture.

1.4 DECGRAPHIC-11 PROGRAMMING PRINCIPLES

The DECgraphic-11 FORTRAN Graphics Package is easy for knowledgeable FORTRAN programmers to use, since it uses the subroutine as the basic building block for graphic programs. Each subroutine in the package transmits all the appropriate display instructions for a given image or procedure to the VS60 and VT11 display processors. Thus, you can concentrate on the image you want on the screen rather than on special programming considerations.

Before you begin work with the DECgraphic-11 systems, there are five programming principles you should understand:

1. The DECgraphic-11 FORTRAN Graphics Package uses a software concept called a "subpicture"; it is to graphic programming what the subroutine is to regular FORTRAN programming.

2. The DECgraphic-11 subroutiles that display images usually display one graphic element for each subroutine call. These graphic elements are called "primitives."

3. The DECgraphic-11 FORTRAN Graphics Package includes 20 "pointers" that you can use to identify individual primitives in a program and on the screen. Pointers are used together with subpictures to allow you to single out a certain primitive--a vector, for example--and to make it flash on the screen, change its brightness, or exercise any of several options, depending on the type of primitive.

4. Certain characteristics of a graphic program are memorized by the DECgraphic-11 software. The DECgraphic-11 FORTRAN Graphics Package will, for example, remember throughout a program that the primitives are sensitive to the light pen, that phe primitives are supposed to flash on and off, and/or that the vectors are to be displayed as solid lines rather than dashed lines. These characteristics -- light-pen sensitivity, flash mode, and line type, as well as the intensity level or brightness of primitives -- are called "display parameters."

5. The DECgraphic-11 subroutines create display instructions for the VT11 and VS60 that are stored in a special area of the computer's memory. This area is called the "display file." As a FORTRAN programmer, you do not have to pay much attention to the display file once you have initialized it, although you do have direct control of its contents if you need it.

The following sections examine each of these principles.

1.4.1 Subpictures

The DECgraphic-11 FORTRAN Graphics Package has a subpicture facility that allows you to combine individual primitives into a single structure. Because many displayed pictures contain repeated images, it is often more efficient to define such images as subpictures. You can then display the subpicture as often as needed and at different parts of the screen without having to repeat a sequence of subroutine calls.

This is a sample of a DECgraphic-11 subpicture definition as it might appear in one of your programs:

```
CALL SUBP(1)
CALL APNT(512.,512.,,-4)
CALL VECT(100.,0.)
CALL VECT(0.,-10.)
CALL VECT(-100.,0.)
CALL VECT(0.,10.)
CALL ESUB
```

CALL SUBP and CALL ESUB are the subroutine calls that always begin and end a subpicture definition, respectively. In that sense, they are like the SUBROUTINE and RETURN statements used in FORTRAN subroutine definitions.

The number 1 in CALL SUBP(1) is the tag of the subpicture you are defining. A tag is the same in function as the name of a subroutine, except that the tag is a number (from 1 to 32767) instead of a word.

This subpicture contains five primitives: an absolute point (the APNT call) and four relative vectors (the VECT calls). If you wrote a program containing this subpicture, the primitives would appear on the screen forming a small rectangle near the screen's center.

The order of the primitives in the subpicture definition is important. For example, you can single out a particular primitive by setting a pointer to "Primitive 1 of Subpicture 1." Primitive 1, in this case, is the absolute point created by the APNT call. Primitive 5 is the last of the four VECT calls. However, pointers cannot identify single primitives unless the primitives are in a subpicture definition.

Subpictures can be copied, erased, turned on or off, and, on the VS60, changed in size; primitives can be added to or deleted from existing subpictures. Calls to existing subpictures can be nested to a depth of eight in order to allow complex structures to be built from simpler components.

Many of the FORTRAN Graphics Package subroutines described in Chapter 2 are for use with subpictures. Some of the most important subpicture operations you can perform in the DECgraphic-11 system are listed below:

- Starting a new subpicture definition or producing multiple images of an existing subpicture (SUBP; see Section 2.4.1).
- Terminating a subpicture definition and, on VS60 systems, optionally restoring any display parameters (see Section 1.4.4) that were reset in the subpicture to the values they had before the subpicture was defined (ESUB; see Section 2.4.2).
- Copying a subpicture and assigning a new tag to the copied subpicture (COPY; see Section 2.4.3).
- Turning a subpicture off temporarily and turning it on again, often to display an image all at once on the screen after it has been totally constructed in the display file (OFF and ON; see Sections 2.4.4 and 2.4.5).
- Erasing a subpicture definition from the display file (ERAS; see Section 2.4.6).
- Creating special subpictures called numeric, graph, and figure subpictures (NMBR, XGRA, YGRA, and FIGR; see Sections 2.4.7, 2.5.1, 2.5.2, and 2.5.3, respectively).
- On the VS60, changing the size of characters and/or vectors to be displayed in a particular subpicture (CVSCAL; see Section 2.4.8).
- Changing individual primitives in a subpicture definition by advancing a pointer through that subpicture's portion of the display file (see Sections 1.4.3 and 2.6).

Special programming techniques that use subpictures are described in Section 3.1.

1.4.2 Primitives

The following graphic images are treated as primitives by the DECgraphic-11 FORTRAN Graphics Package:

- vectors, displayed with the VECT, AVECT, SVECT, and LVECT subroutines
- points, displayed with the APNT and RPNT subroutines
- text strings, displayed with the TEXT subroutine

Primitives do not have tags or labels to distinguish them from each other. The only way a primitive can be identified is by its numerical order within a subpicture definition, as described in Section 1.4.1.

When you draw primitives, the position of the DECgraphic-11 display beam is important. The beam is the stream of electrons that points to a particular position on the display screen. When you initialize the display file (see the INIT subroutine, Section 2.1.1), the beam is positioned at the lower left corner of the screen (x=0., y=0.). As you draw primitives at different coordinate positions on the display screen, the beam changes position.

At your option, each primitive can have some unique properties. With a vector, for example, you have a choice of making it sensitive to the light pen, a choice of making it a constant or flashing line segment, a choice of eight levels of brightness on the screen, and a choice of four line types. Text strings can be in regular English letters, Greek letters, mathematical symbols, or italics, and they can be displayed in the normal horizontal way or (on the VS60) rotated 90 degrees. Each subroutine call that displays a primitive also allows you to set these display parameters in the same subroutine call.

You can combine any kind or number of primitives into a subpicture.

1.4.3 Pointers

When you have defined a subpicture, you can use one or more pointers to single out particular primitives within the subpicture. After identifying a particular primitive in this way, you can then change it with a variety of pointer-oriented subroutines in the DECgraphic-11 FORTRAN Graphics Package.

The subroutine POINTR does the first job, that of associating a pointer with a primitive. One example of a pointer-oriented subroutine is CHANGE, which changes the coordinates of a primitive. To illustrate these jobs, consider the sample subpicture from Section 1.4.1:

```
CALL SUBP(1)
CALL APNT(512.,512.,,-4)
CALL VECT(100.,0.)
CALL VECT(0.,-10.)
CALL VECT(-100.,0.)
CALL VECT(0.,10.)
CALL ESUB
```

Starting with the SUBP call, this set of statements translates as follows:

1. Begin the definition of Subpicture 1.

2. Go to the center of the screen. (Primitive 1 is an absolute point at [x=512., y=512.], which is the center. The -4 argument means that the point itself will not be visible, but that all the following primitives will have an intensity level [brightness] of 4.)

3. Draw a vector from (512.,512.) to (612.,512.). (Primitive 2)

4. Draw a vector from (612.,512.) to (612.,502). (Primitive 3)

5. Draw a vector from (612.,502.) to (512.,502.) (Primitive 4)

6. Draw a vector from (512.,502.) to (512.,512.). (Primitive 5)

7. End of subpicture definition.

Notice that the absolute point is the starting place for the first vector; in other words, these are relative vectors that are all defined with respect to the point (512.,512.). The entire subpicture (that is, the rectangle) can be moved around on the screen by changing the coordinates of the absolute point. The POINTR and CHANGE subroutines are used as follows:

POINTR has the form CALL POINTR(k,m,j), where k is the pointer number, m is the subpicture tag, and j is the primitive number.

CHANGE has the form CALL CHANGE(k,x,y), where k is the pointer number, and x and y are the new coordinates.

Suppose these two subroutine calls are added to the example:

```
        CALL SUBP(1)
        CALL APNT(512.,512.,,-4)
        CALL VECT(100.,0.)
        CALL VECT(0.,-10.)
        CALL VECT(-100.,0.)
        CALL VECT(0.,10.)
        CALL ESUB
C
C       WAIT FOR <RETURN>
C
        READ (5,1) KR
1       FORMAT(I2)
C
        CALL POINTR(20,1,1)
        CALL CHANGE(20,0.,512.)
```

If this fragment is part of a program, the rectangle in Subpicture 1 appears on the screen, and then the program pauses, waiting for a carriage return. When you type a carriage return on the terminal keyboard, the program executes the two new subroutine calls, with the following effects:

1. POINTR assigns pointer number 20 to Subpicture 1, Primitive 1 [CALL POINTR(20,1,1)]. Pointer 20 now points to the APNT call in Subpicture 1.

2. CHANGE now changes the coordinates of the primitive pointed to by Pointer 20 (APNT) to (x=0., y=512.). At this point, the subpicture has a new starting position; that is, as soon as CHANGE is executed, the rectangle moves immediately to the left edge of the screen, halfway up from the bottom. Pointer 20 still points to the absolute point.

From this elementary example, you can see how pointers and subpictures are used for dynamic displays; some parts of the display--some subpictures--can move around on the screen while others remain still.

In a similar way, pointers can be used to change points and text strings. The detailed subroutine descriptions in Chapter 2 clarify the use of pointers with other kinds of primitives.

The availability of 20 pointers enables you to keep track of as many as 20 primitive locations.

The following list summarizes operations that you can perform by manipulating pointers:

- Setting a pointer at a particular primitive within a subpicture (POINTR; see Section 2.6.1).
- Advancing a pointer by a certain number of primitive elements (ADVANC; see Section 2.6.2).
- Returning the coordinate positions of the primitive currently identified by a pointer (GET; see Section 2.6.3).
- Changing the screen coordinates of a primitive identified by a pointer (CHANGE, CHANGA, and CHANGT; see Sections 2.6.4 through 2.6.6).
- Inserting a new element in the display file just before the primitive identified by a pointer (INSRT; see Section 2.6.7).
- Erasing the primitive identified by a pointer (ERASP; see Section 2.6.8).

1.4.4 Display Parameters

Display parameters control different "modes" of the display. That is, when a mode is entered at a certain point in the program, all primitives from that point on will be displayed in that same mode, unless or until the mode is changed again.

There are four display parameters in the DECgraphic-11 FORTRAN Graphics Package. They control the following properties of the display:

Display Parameter l controls light-pen sensitivity. If l is positive, all subsequent primitives on the screen will be sensitive to the light pen. When the light pen is pointed at a sensitive primitive on the screen, a graphic attention will occur. If l is negative, the subsequent primitives are not sensitive.

Display Parameter i controls intensity level (the brightness of primitives on the screen). The intensity level has eight values, with 1 the faintest and 8 the brightest.

NOTE

1. DECgraphic-11 display scopes also have a knob on the scope cabinet for adjusting the intensity. Therefore, the intensity level is a relative property, depending on the adjustment of the knob.

2. The light pen cannot detect primitives that are below a certain brightness. When the intensity knob is set near the middle of its range, Intensity Level 4 is the minimum intensity at which primitives can be detected.

Display Parameter f controls flash mode. If f is positive, the subsequent primitives on the screen will flash; if f is negative, the primitives will be constant.

Display Parameter t controls line type. The line type controls the appearance of subsequent vectors on the screen. There are four line types: Type 1 is a solid line; Type 2 is a line of long dashes; Type 3 is a line of short dashes; and Type 4 is a line of alternating dashes and dots.

You can set values for the display parameters by including the values as optional arguments in some of the subroutine calls. In such calls, the display parameters are always the last four arguments, and they are always in the order 1, i, f, t. For instance, the call

```
CALL VECT(100.,100.,1,5,1,4)
```

will display a light-pen-sensitive vector at Intensity Level 5. The vector will flash on and off and will appear as a dot-dash line. If any or all of the display parameters are omitted from such a call, or if they are set to 0, those display parameters will not change.

Another way of setting display-parameter values is with the SENSE, INTENS, FLASH, and LINTYP subroutines (see Section 2.7). These are pointer-oriented subroutines. In effect, each supplies a new value for a single display parameter to the referenced primitive.

The intensity level is a special parameter in the sense that you can use it to change the intensity of part of the display without affecting the rest of the display. If, instead of supplying a number from 1 to 8 for Display Parameter i, you supply a value from -1 to -8, the primitive containing the negative value will not be displayed at all, but the intensity level of primitives that follow will be the absolute value of the negative number. For example, in the following segment:

```
CALL SUBP(1)
CALL APNT(512.,512.,,-4)
CALL VECT(100.,0.)
CALL VECT(0.,-100.)
CALL VECT(-100.,0.,,8)
CALL VECT(0.,100.,,4,1,4)
CALL ESUB
```

the absolute point described by the APNT call will not be visible on the screen. However, the first and second vectors will be displayed, at Intensity Level 4. In the third VECT call, the intensity level is changed to 8, and so the vector will appear at maximum brightness. The fourth VECT call changes the intensity level to 4 again, and also turns on flash mode and switches the line type to Line Type 4.

Now consider this pair of subroutine calls:

```
CALL POINTR(20,1,2)
CALL SENSE(20,1)
```

In the first call, Pointer 20 is pointed to Primitive 2 of Subpicture 1 (the first VECT call). Then the SENSE call passes the new value of 1 (1) to the VECT call, effectively adding it on to the list of arguments. Now all four vectors are sensitive to the light pen.

The subroutines INTENS, FLASH, and LINTYP, described in Section 2.7, pass new values of i, f, and t, respectively. INTENS can, by passing a 0 for i, also effectively erase the minus sign on the intensity parameter of a certain primitive. If, for instance, you use INTENS to pass a 0 to this call:

```
CALL APNT(10.,0.,,-4)
```

the new intensity parameter will be 4 instead of -4. The previously invisible point will suddenly appear on the screen.

1.4.5 The Display File

The display file is a list of graphic instructions and data that is created by the DECgraphic-11 subroutines.

To display images on the screen, the VS60 or VT11 display processing unit (DPU) interacts with the display file much as the PDP-11 central processing unit (CPU) interacts with PDP-11 memory. Just as the PDP-11 CPU retrieves successive program instructions and data from memory, the VS60 or VT11 DPU retrieves display instructions and data from the display file, independent of the operation of the PDP-11 CPU. The internal configuration is shown in Figure 1-13.

To display an image on the screen, the DPU simply executes the display instructions stored in the display file, starting with the first instruction. The DPU continues executing display instructions until it reaches the instructions "DHALT, 0," which mark the end of the active display list. The display processor then automatically jumps back to the beginning of the file and begins to execute the display instructions again. This process displays a seemingly continuous, or "refreshed," image on the screen.

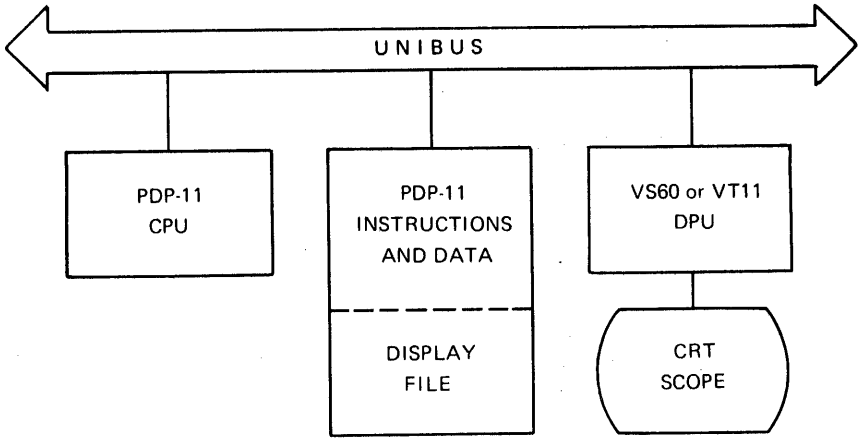

Figure 1-13 CPU and DPU

As you call subroutines to display graphs, points, or other objects on the screen, the display file fills up with instructions and data. Although you can "erase" primitives or subpictures from the screen, the space occupied by these elements cannot be reused until you call the CMPRS subroutine (see Section 2.13.1), which removes erased display elements and reclaims their display-file space.

As mentioned previously, you do not have to concentrate on the display file because the display instructions are created automatically.

However, at the beginning of each graphic program, you must define a FORTRAN COMMON area for the display file, and you must also initialize the file with the subroutine INIT. Section 2.1 describes INIT and other subroutines that control the display file.

When you initialize the display file, the display parameters are set to the following initial values:

- Light-pen sensitivity (l in the subroutines) is disabled.
- Intensity level (i) is set to 4 in the range 1 (faintest) to 8 (brightest).
- Flash (f) or blink mode is disabled.
- Line type (t) is set to solid.

There are other default conditions associated with an initialized display file. These are described in the discussion of the INIT subroutine (see Section 2.1.1).

Instructions in the display file occur in essentially the same order as DECgraphic-11 subroutine calls in a FORTRAN program. Consequently, as discussed in Section 2.2.5, a call to the VIEWPT subroutine must precede the calls that display objects in a new viewport.

The special ramifications of display-file structure on the operation of individual subroutines are described in the subroutine descriptions in Chapter 2.

A display file can be saved as a data file on a mass-storage device, such as a disk, DECtape, or floppy disk. By saving and subsequently restoring a display file, you can display a screen image without having to rerun the FORTRAN program that was used to create the image. Use the SAVE and RSTR subroutines (see Sections 2.13.2 and 2.13.3) to save and restore DECgraphic-11 display files.

Section 2.14 describes three subroutines that allow you to create unusual graphic instructions directly in the display file without using any other DECgraphic-11 subroutines. Appendix C describes the structure of the display file as it appears in the computer memory. If you intend to use any of the subroutines in Section 2.14, read Appendix C and be sure that you understand the structure of the display file.

1.4.6 Summary

The elements displayed by the DECgraphic-11 FORTRAN Graphics Package are called primitives. A FORTRAN program builds a complete picture by repeated calls to subroutines that display single primitives.

Primitives can be combined into modular structures called subpictures. You can then operate on an entire subpicture instead of addressing its individual primitives.

You can use DECgraphic-11 pointers to identify and manipulate individual primitives inside a subpicture.

Several modes, or display parameters, can be changed by optional subroutine arguments or special subroutine calls to control light-pen interaction, flashing displays, the brightness of objects, and the appearance of lines on the screen.

The instructions and data created by your DECgraphic-11 subroutine calls are stored in a memory area called the display file. The display file can be stored permanently on a mass-storage device and later recalled by other programs. The display processor (DPU) retrieves and executes display instructions from the part of the display file called the active display list.

CHAPTER 2

DECGRAPHIC-11 FORTRAN SUBROUTINES

This chapter describes in detail the graphic subroutines in the DECgraphic-11 FORTRAN Graphics Package. You can call any of the subroutines discussed here from your FORTRAN programs.

All subroutines in this chapter can be used in either a stand-alone or host-satellite system.

However, this chapter does not describe the special subroutines used for host-satellite communication, such as TOSAT. It also does not describe the ACCESS and READWR subroutines that store and recall satellite files. The subroutines in these special categories are described in Section 5.3.2.

As an option, you can build DECgraphic-11 libraries that take only integers as arguments (see Chapters 4 and 5). This option has some advantages, but the arguments must then be within the range of the unit-scaled coordinate system. The user dialog of the program COND, described in Chapters 4 and 5, details the legal ranges of integer arguments.

Each section in this chapter describes the use of subroutines in one of the functional categories listed below:

- Initializing and controlling the display file
- Choosing screen area and scaling
- Creating graphic primitives
- Defining and using subpictures
- Displaying graphs and figures
- Using display-file pointers
- Changing display parameters
- Interacting with the display
- Polling interactive devices
- Using the optional pushbutton box
- Controlling the keyboard
- Controlling the overall display
- Compressing, saving, and restoring the display file
- Inserting advanced display-file instructions

2.1 INITIALIZING AND CONTROLLING THE DISPLAY FILE

The subroutines described in this section allow you to allocate memory for the DECgraphic-11 display file, initialize the display file, and control access to it.

2.1.1 INIT: Initializing the Display File

Form: CALL INIT [(n)]

Arguments:

The n argument is the number of words that will be used for the display file. The n argument is required only the first time you call INIT; do not supply an argument when you call INIT later in a program to reinitialize the display file.

Instructions:

Before you call INIT, you must define a COMMON block that will contain the display file. The name of this COMMON block must be DFILE. The COMMON statement must also assign an integer array to DFILE that is equal in length to the display file your program will use. The name of the integer array is not important, but the array must be large enough to accommodate all the display instructions that will be created by your program. If you run out of display-file space during the execution of your graphic program, the message

```
DISPLAY FILE FULL
```

will appear, and program execution will terminate. To correct the problem, you should either assign more display-file space to DFILE, or you should compress the display file (see Section 2.13.1).

INIT clears the display screen, sets the display parameters to their initial values, and initializes the number of display-file words equal to n in your call.

The following example illustrates the creation of a FORTRAN COMMON block and the use of a call to INIT.

```
COMMON/DFILE/I(400)
CALL INIT (400)
```

This COMMON statement assigns 400 words of memory to the area named DFILE. The INIT call then initializes 400 words of DFILE, so that the display processor will begin at the first word of the display file, I(1), and execute any graphic instructions in the display file. When 400 words of instructions and data have been processed, the display processor will return to I(1) and start over. This cyclic action, or "refreshment," draws the same picture on the screen several times a second, creating the illusion of a continuous display.

If you call INIT later in the program to reinitialize the display, do not include any argument. Using INIT in this way clears any existing display from the screen, clears all graphic instructions from the display file, and resets the display parameters to their initial values. The display processor will continue to cycle on the display file you defined at the beginning of the program.

Once you have defined and initialized the display file, calls to the image-generating subroutines will add new instructions and data to the end of the display file or will modify the existing contents of the file. Since the INIT call has started the display processor, these new instructions will be executed immediately and the new graphics will appear immediately on the screen.

Any FORTRAN program that has called INIT will not return to the monitor immediately after execution or error detection. Instead, the system will display the following message on the console:

TYPE <CR> TO EXIT

Even if an error has occurred, you will be able to view the display created by the program without having to issue a READ or PAUSE instruction.

NOTE

Do not use the FORTRAN library routine, USEREX, after a call to INIT. The INIT subroutine issues its own call to USEREX.

INIT also sets the following initial conditions for the display:

1. The standard coordinate system is enabled (see Section 2.2.3).
2. The beam is positioned at the lower left corner of the viewing area of the screen (x=0., y=0.).
3. Light-pen interaction (display parameter l) is disabled.
4. The intensity level (display parameter i) is set to 4.
5. Flash mode (display parameter f) is turned off.
6. Line type (display parameter t) is solid.
7. Characters are displayed in the normal font, not in italics or shift-out mode (see Section 2.3.7).
8. For the VS60, Scope 1 is on and Scope 2 is off (see Section 2.2.1).
9. On the VS60, any display is directed to the main viewing area, not the menu area (see Section 2.2.2).

Routines called by INIT: STOP, NOWNDW, CONT, TOSAT (h-s)

Routines that call INIT: None.

2.1.2 STOP: Stopping the Display

Form: CALL STOP

Arguments: None.

Instructions:

STOP halts the display processor and clears the display screen. The stopped display can be restored by a call to the CONT subroutine (see Section 2.1.3).

STOP also halts the transmission of interrupts from the VT11 or VS60 display processor. Because interrupts from the display processor are not being processed, the CPU executes at a significantly faster rate.

Attempting to stop a display that has already been stopped has no effect.

On the VS60, STOP clears the screens on both scopes, if both are enabled (see Section 2.2.1).

Routines called by STOP: None.

Routines that call STOP: INIT, ERAS, ERASP, ESUB, INSRT, OFF, CMPRS

2.1.3 CONT: Restoring the Display

Form: CALL CONT

Arguments: None.

Instructions:

CONT restores the display that was halted by a call to STOP. CONT restarts the display processor and the transmission of interrupts from the VT11 or VS60.

Routines called by CONT: None.

Routines that call CONT: INIT, ERAS, ERASP, ESUB, INSRT, OFF, CMPRS

2.1.4 FREE: Releasing the Display-File Area

Form: CALL FREE

Arguments: None.

Instructions:

Use FREE when you need to use the memory area allocated to the display file for another purpose. A call to FREE effectively disconnects the display file from the display processor, thus freeing the space used by the display file. FREE also clears the display screen.

After you have called FREE, no display file is available for use in graphic processing. If you need the display file later in your program, call INIT again. The COMMON DFILE definition remains in effect.

Routines called by FREE: TOSAT (h-s)

Routines that call FREE: None.

2.2 CHOOSING SCREEN AREA AND SCALING

The subroutines described in this section allow you to select the scope (VS60) and the screen area and to establish your own coordinate system for graphic displays.

2.2.1 SCOPE: Selecting a VS60 Display Scope

Form: CALL SCOPE (n)

Arguments:

The n argument is the scope number, 1 for Scope 1 and 2 for Scope 2. If n is positive, the scope will be enabled, and if n is negative, the scope will be disabled.

Instructions:

SCOPE is used in VS60 display subsystems to enable or disable either of the two scopes. After you have enabled a scope, graphic calls will display images on it.

A call to SCOPE has no effect on a VT11 subsystem.

An enabled scope remains enabled until you explicitly disable it, so that both VS60 scopes can be used at the same time. However, if a call to INIT is made during program execution, the initial scope setting will be reestablished, and only Scope 1 will be enabled.

Routines called by SCOPE: None.

Routines that call SCOPE: None.

2.2.2 AREA: Selecting the Main or Menu Area

Form: CALL AREA (n)

Arguments:

When n is 1, the display is switched to the main viewing area. When n is 2, the display is switched to the menu area. If n has any other value, the AREA call is ignored.

Instructions:

With VS60 subsystems, AREA allows you to switch subsequent grapnic displays to one of the two distinct areas of the display screen; the main viewing area or the menu area.

A call to AREA has no effect on a VT11 subsystem.

When you call AREA, the current beam position is not automatically changed, so you must reposition the beam before displaying graphics on the screen.

When you address the menu area, the x coordinate can range from 0 to 127 and the y coordinate from 0 to 1023.

It is not possible to address both the main viewing area and tne menu area at the same time.

Example:

This example (see Figure 2-1) shows a figure subpicture (see Section 2.5.3) displayed in the main viewing area and a scaled copy of the same subpicture in tne menu area:

```
      COMMON/DFILE/IBUF(1000)
      REAL ENDPNT(16)
      DATA ENDPNT/0.,100.,100.,0.,0.,-100.,-100.,0.,100.,100.,
     X -50.,50.,-50.,-50.,100.,-100./
      CALL INIT(1000)
      CALL APNT(512.,512.,,-4)
      CALL FIGR(ENDPNT,16,2000)
      CALL RPNT(0.,-50.,,-4)
      CALL TEXT(-2,'MAIN AREA')
      CALL AREA(2)
      CALL SUBP(2001)
      CALL SUBP(2002)
      CALL APNT(0.,512.,,-4)
      CALL COPY(,2000)
      CALL RPNT(0.,0.,,-2)
      CALL TEXT(1,-2,'MENU AREA')
      CALL ESUB
      CALL CVSCAL(2002,1,2)
      CALL ESUB(2001)
      CALL AREA(1)
      STOP
      END
```

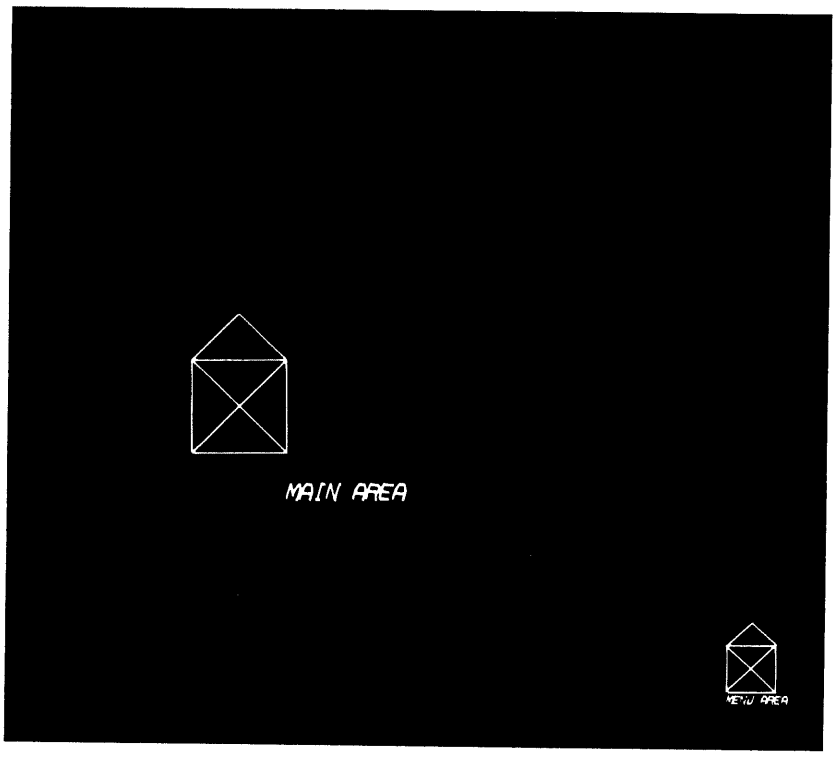

Figure 2-1 AREA Subroutine

Routines called by AREA: None.

Routines that call AREA: None.

2.2.3 WINDW: Redefining the Window

Form: CALL WINDW (x0,y0,xl,yl[,FX[,FY]])

Arguments:

The x0 and y0 arguments are the coordinates of the lower left corner of the window. The xl and yl arguments are the coordinates of the upper right corner of the window. FX and FY are the scaling factors of the x and y axes, respectively.

Instructions:

WINDW allows you to define your own window. The default window describes the main screen area as the area extending from (x=0., y=0.) to (x=1023., y=1023.). The area has a physical size of 1024 raster units on each axis, and each axis contains 1024 individually addressable points. Consecutive coordinate positions on an axis are addressed in increments of 1 (e.g., x=0., x=1., x=2.), so the default coordinate system is unit scaled.

WINDW can change this default coordinate system to almost any other scale. For example, you can specify a coordinate system containing only one fourth the number of addressable points as the default scale:

```
CALL WINDW (0.,0.,512.,512.)
```

where (0.,0.) is the lower left corner and (512.,512.) is the upper right corner of the display screen. The physical screen size remains the same (1024 raster units along each axis), but consecutive coordinate positions are now two raster units apart, not one. The scaling factor along each axis is .5.

If you make the following calls:

```
CALL WINDW (0.,0.,200.,200.)
CALL APNT (0.,0.)
CALL VECT (200.,200.)
```

the vector that is produced will extend on the main display diagonal, completely across the screen.

You can also specify one or two additional arguments in the WINDW call. These arguments return the scaling factors for the x and y axes. If FX is included in the call, the x scaling factor will be returned; if you also include FY, the y scaling factor will be returned as well.

Each new call to WINDW redefines the window only for subsequent graphic calls. Any previous windows still apply to primitives that precede the new call to WINDW.

NOTES ON WINDOWS AND COORDINATES

1. The length and coordinates specified in every graphic call must be within the current coordinate system (the default system or the system established by a call to WINDW). Any graphic call that attempts to place the beam outside the window is ignored.

2. The WINDW subroutine does not function with libraries built to take only integers as arguments (see Chapter 4 or 5).

3. Because all vector displacements and point coordinates are represented as integers in the display file, errors can accumulate when several relative vectors or points are drawn consecutively. This happens when the vector displacement or point coordinates are converted to a noninteger number of physical screen units. For instance, a scaling factor of .5 will convert the coordinate 511 to a noninteger number.

4. On VS60 subsystems, WINDW changes the coordinate range of the entire image definition area. CALL WINDW(0.,0.,1.,1.) will change the limits of the image definition area to the coordinates (-4.,-4.) and (4.,4.).

An example using WINDW is included at the end of the description of the NOWNDW subroutine (see Section 2.2.4).

Routines called by WINDW: TOSAT (h-s)

Routines that call WINDW: NOWNDW

2.2.4 NOWNDW: Restoring the Standard Coordinate System

Form: CALL NOWNDW

Arguments: None.

Instruction:

NOWNDW eliminates the user-defined window that is in effect and restores the default window. This action reestablishes the standard unit-scaled coordinate system, in which the screen is described as the area between coordinate positions (x=0., y=0.) and (x=1023., y=1023.).

Example:

This example (see Figure 2-2) draws a figure subpicture (see Section 2.5.3), changes the scaling factor to draw a larger version of the same figure, and then restores the unit-scaled coordinate system.

```
      COMMON/DFILE/IBUF(1000)
      REAL ENDPNT(16)
      DATA ENDPNT/0.,100.,100.,0.,0.,-100.,-100.,0.,100.,100.,
     X -50.,50.,-50.,-50.,100.,-100./
      CALL INIT(1000)
      CALL APNT(512.,512.,,-4)
      CALL FIGR(ENDPNT,16,100)
      CALL RPNT(125.,0.,,-4)
      CALL WINDW(0.,0.,750.,750.)
      CALL FIGR(ENDPNT,16,101)
      CALL NOWNDW
      STOP
      END
```

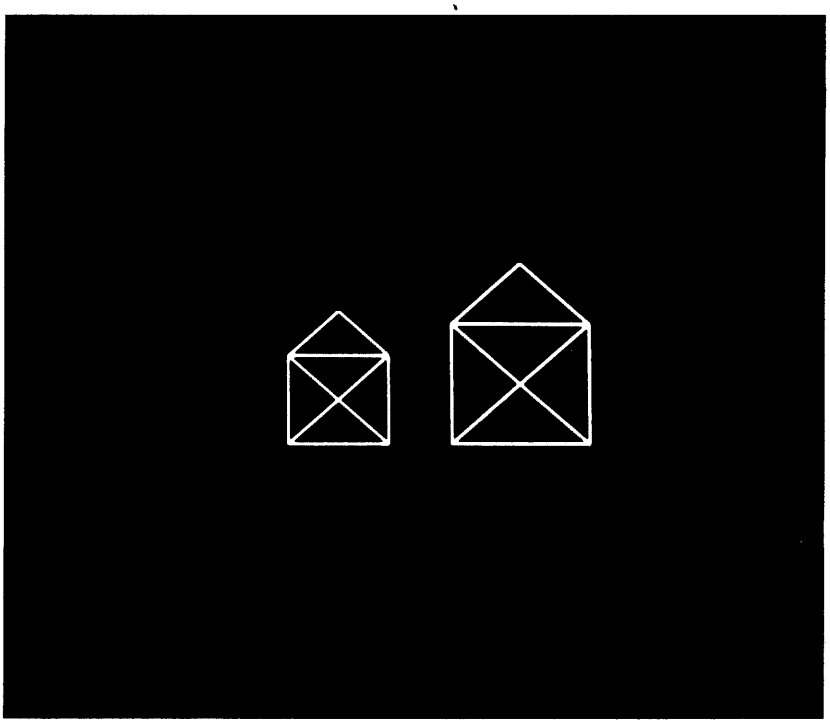

Figure 2-2 WINDW and NOWNDW Subroutines

Routines called by NOWNDW: WINDW

Routines that call NOWNDW: INIT

2.2.5 VIEWPT: Redefining the VS60 Viewport

Form: CALL VIEWPT (x,y)

Arguments:

The x and y arguments are the coordinate positions of the lower left corner of the user-defined viewport.

Instructions:

This subroutine allows you to redefine the viewport on the image definition area of the VS60. If you are using the default window, the image definition area extends from (-4095., -4095.) to (4095., 4095.) and is therefore 64 times as large as the default viewport. With VIEWPT, you can select the portion of the image definition area you will view on your display screen.

If you use WINDW to change the coordinate system of the screen, the maximum coordinates of the image definition area will be changed by the same scaling factors that apply to the viewport. For example, after the following call:

```
CALL WINDW (0.,0.,1.,1.)
```

the image definition area will extend from (-4.,-4.) to (4.,4.).

The size of the viewport is always 1024 by 1024 raster units. If the lower left corner is represented by (x,y), then the upper right corner is (x+1023., y+1023.).

If you attempt to move the viewport across a part of the image definition area that contains absolute vectors (see Section 2.3.4), the absolute vectors may disappear. The absolute vectors defined by the AVECT subroutine are the only primitives affected in this way by VIEWPT. To avoid this problem, use VECT, SVECT, or LVECT to draw vectors (all relative vectors) rather than AVECT whenever you plan also to use VIEWPT.

The VIEWPT call affects only the primitives that follow the call. Consequently, always place the VIEWPT call before the primitives that you want it to affect. A problem can occur if you include both VIEWPT and CVSCAL (see Section 2.4.8) in the same subpicture. The CVSCAL call within the subpicture places the scaling instruction before all other instructions in the subpicture, including the VIEWPT instruction. If you want to perform both a VIEWPT and a CVSCAL operation on a set of primitives, then do not include CVSCAL and VIEWPT in the same subpicture.

In the following example:

```
CALL VIEWPT (512.,512.)
```

the area of the screen shown in Figure 2-3 is the new viewport.

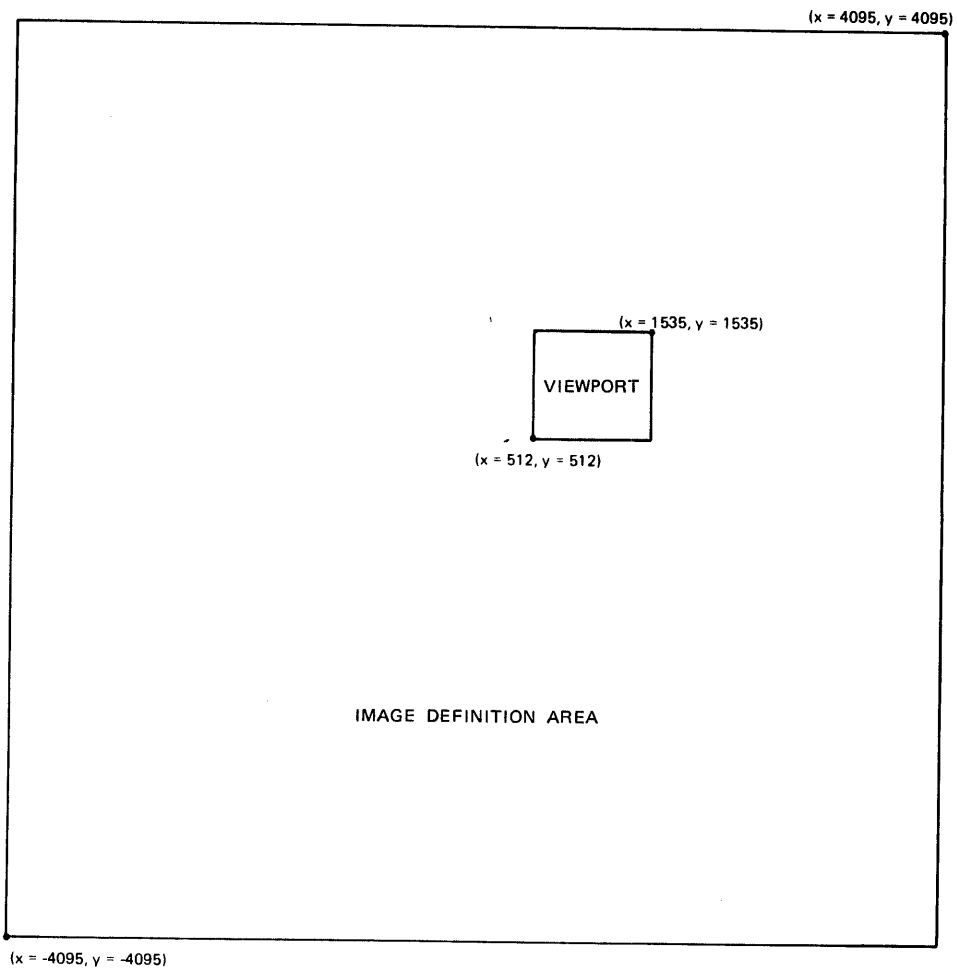

Figure 2-3 VIEWPT Subroutine

Example:

This example sets the viewport to extend from a lower left corner of (1024.,0.) to an upper right corner of (2047.,1023.).

```
      .
      .
      .
CALL SUBP (1)
CALL VIEWPT (1024.,0.)
CALL ESUB
      .
      .
      .
```

The viewing area is now changed to extend from (x,y) to (x+1023., y+1023.)

To change an existing viewport, use the CHANGE subroutine (see Section 2.6.4). When CHANGE is used for this purpose, its arguments must be the negatives of the arguments you would normally give VIEWPT. For example, to move the lower left corner of the viewport to (512.,512.), give to CHANGE the arguments (-512.,-512.).

Routines called by VIEWPT: None.

Routines that call VIEWPT: None.

2.3 CREATING GRAPHIC PRIMITIVES

This section describes the DECgraphic-11 subroutines that create graphic primitives such as points, vectors, and characters. These subroutines insert primitives in the display file and, if you so request, immediately display them on the screen.

You can use many of the subroutines in this category to change the display parameters described in Section 1.4.4.

If you are a VS60 user, remember that the primitives created by these subroutines can be placed anywhere in the image definition area.

Remember also that, although many of the subroutine arguments are shown as FORTRAN REAL variables, you can create special DECgraphic-11 libraries in which all subroutine arguments are integers. For more details, see the chapter on operating instructions for your operating system (Chapter 4 or 5).

2.3.1 APNT: Displaying an Absolute Point

Form: CALL APNT (x,y[,l,i,f,t])

Arguments:

The x and y arguments are the coordinates of the absolute point. The four optional arguments l, i, f, and t are the display parameters described in Section 1.4.4.

Instructions:

APNT moves the beam to an absolute coordinate position and creates a point at that location.

In the call to APNT, you can optionally specify new values for the display parameters l, i, f, and t. For example, to position the beam without displaying a point, use a negative value for the intensity (i) parameter.

Example:

This example (see Figure 2-4) draws four flashing points on the display screen at Intensity Level 3.

```
COMMON/DFILE/IBUF(1000)
CALL INIT(1000)
CALL APNT (100.,100.,,3,1)
CALL APNT (100.,923.)
CALL APNT (923.,100.)
CALL APNT (923.,923.)
STOP
END
```

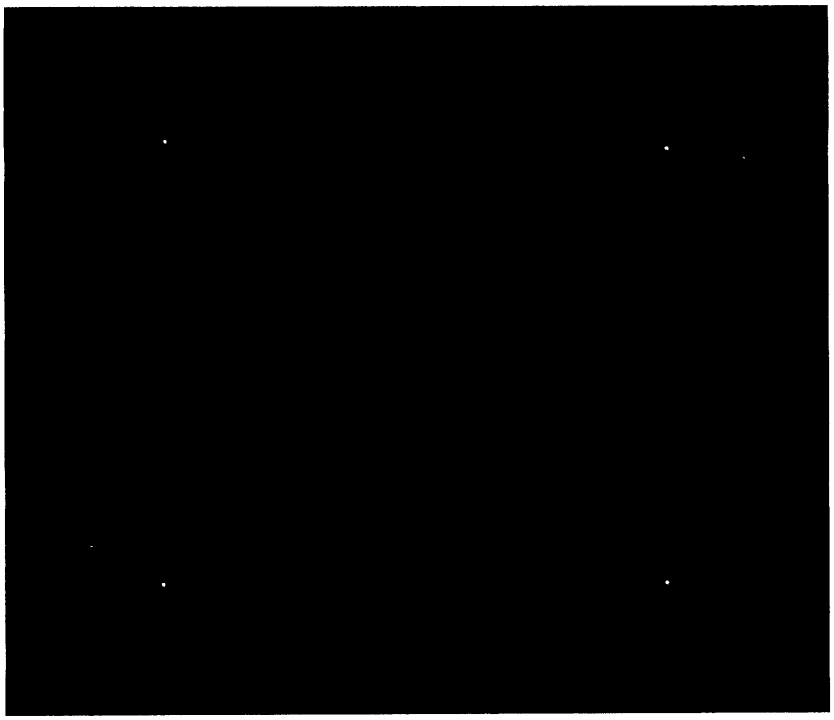

Figure 2-4 APNT Subroutine

Routines called by APNT: None.

Routines that call APNT: MENU

2.3.2 RPNT: Displaying a Relative Point

Form: CALL RPNT (x,y[,l,i,f,t])

Arguments:

The x and y values will be added to the coordinates of the current beam position. The relative point will appear at the resulting coordinates.

Instructions:

RPNT allows you to move the beam to a position relative to the current beam position, with or without displaying a new point. If the current beam position is represented by (x0,y0), then a call to RPNT will move the beam to the relative position represented by (x0+x,y0+y).

When a beam is repositioned by a call to RPNT, the resulting displacement in either the x or y direction cannot exceed 63 raster units (approximately one sixteenth of the full screen). Thus in the default (unit-scaled) coordinate system, the maximum values you can specify in (x,y) are (-63.,-63.) or (63.,63.). If your (x,y) specification exceeds 63 raster units in either the x or the y direction, it will be truncated to 63 units by RPNT.

In the call to RPNT, you can optionally specify new values for the display parameters l, i, f, and t. As with APNT, you can use RPNT to position the beam without displaying a point by specifying a negative value for the i parameter.

Example:

This example (see Figure 2-5) draws a sine wave, made up of dots, across the display screen.

```
        COMMON/DFILE/IBUF(1000)
        CALL INIT(1000)
        CALL APNT (0.,500.)
        Y0=500.
        DO 10 I=1,50
        Y=SIN(.125*I)*500.+500.
        CALL RPNT (20.,Y-Y0)
10      Y0=Y
        STOP
        END
```

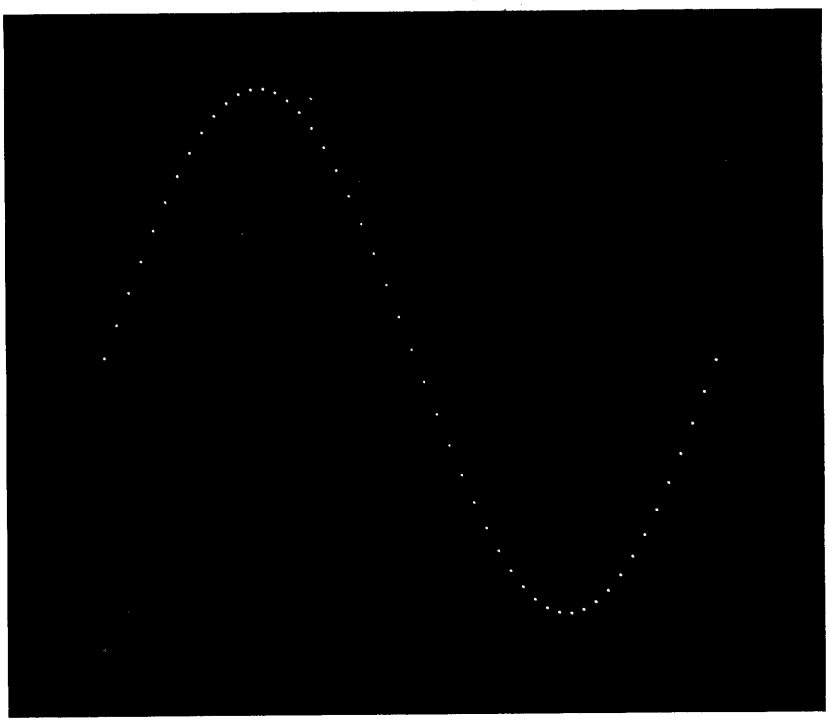

Figure 2-5 RPNT Subroutine

Routines called by RPNT: None.

Routines that call RPNT: MENU

2.3.3 VECT: Drawing a Relative Vector

Form: CALL VECT (x,y[,l,i,f,t])

Arguments:

The x and y values are added to the coordinates of the current beam position. A relative vector will be drawn to the resulting coordinate position.

Instructions:

VECT draws a relative vector from the current beam position to a point relative to the current beam position. If the current beam position is represented by (x0,y0), then a call to VECT will draw a line segment from (x0,y0) to (x0+x,y0+y).

The absolute values of the (x,y) specifications included in the VECT call must not exceed 1023 raster units.

If the displacement of a vector is equal to (0.,0.), the vector will not be visible on the screen, regardless of the intensity level established by the current value of display parameter i.

VECT uses the short-vector format (one word in the display file for each vector) whenever possible in drawing line segments. If your program requires use of the long-vector format -- for example, for "stretching" vectors with the light pen -- you should use LVECT (see Section 2.3.6).

In the VECT call, you can optionally specify new values for the display parameters l, i, f, and t.

Example:

This example draws a triangle (see Figure 2-6) in the center of the screen. The triangle's starting point is repeatedly changed by the CHANGE subroutine in such a way that the triangle rotates in a clockwise direction and then comes to rest near its starting position.

```
        COMMON/DFILE/IBUF(1000)
        CALL INIT(1000)
        CALL SUBP(100)
        CALL APNT(500.,500.,,-4)
        CALL VECT(100.,0.)
        CALL VECT(0.,100.)
        CALL VECT(-100.,-100.)
        CALL ESUB
        CALL POINTR(20,100)
        DO 100 I=1,1500
        Y=SIN(I/50.)*50.+450.
        X=COS(I/50.)*50.+450.
100     CALL CHANGE(20,X,Y)
        STOP
        END
```

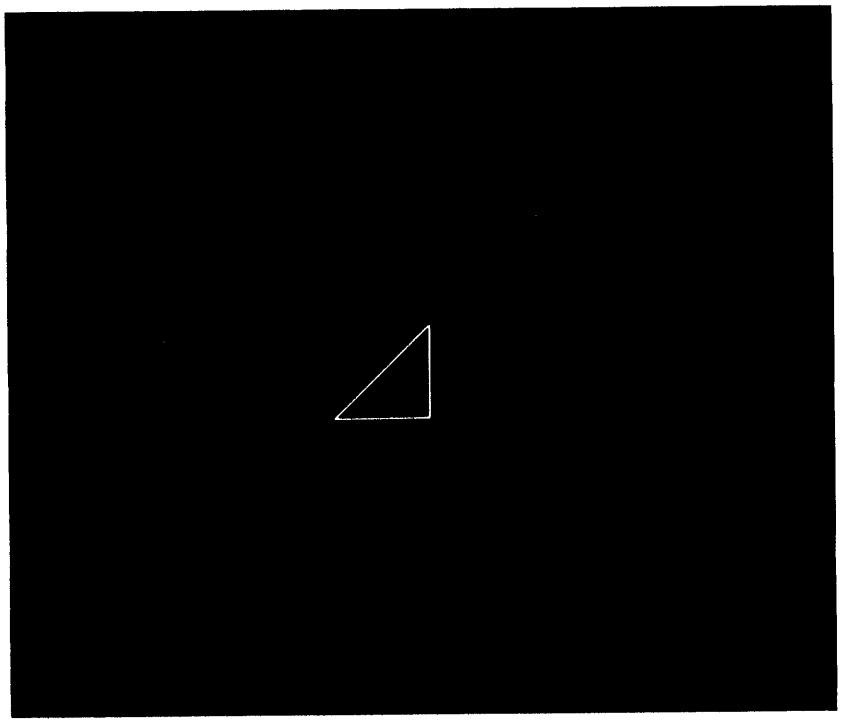

Figure 2-6 VECT Subroutine

Routines called by VECT: None.

Routines that call VECT: None.

2.3.4 AVECT: Drawing an Absolute Vector

Form: CALL AVECT (x,y[,l,i,f,t])

Arguments:

The x and y values are the coordinates of the destination of the vector. The arguments l, i, f, and t are the display parameters described in Section 1.4.4.

Instructions:

AVECT draws a vector from the current beam position to an absolute point on the image definition area, where x and y are the coordinates of the point. AVECT uses the long-vector format (two words for each vector) in drawing line segments. In the call to AVECT, you can optionally specify new values for the display parameters l, i, f, and t.

NOTE

Absolute vectors should not cross the boundaries of the viewport. If you are using AVECT together with VIEWPT, be sure to observe the cautions discussed in Section 2.2.5.

AVECT only works with the VS60 display subsystem; it is ignored by the VT11.

Example:

This example (see Figure 2-7) outlines the screen with dot-dash lines (Line Type 4) at Intensity Level 4.

```
COMMON/DFILE/IBUF(1000)
CALL INIT(1000)
CALL APNT (0.,0.,,-4,,4)
CALL AVECT (0.,1023.)
CALL AVECT (1023.,1023.)
CALL AVECT (1023.,0.)
CALL AVECT (0.,0.)
STOP
END
```

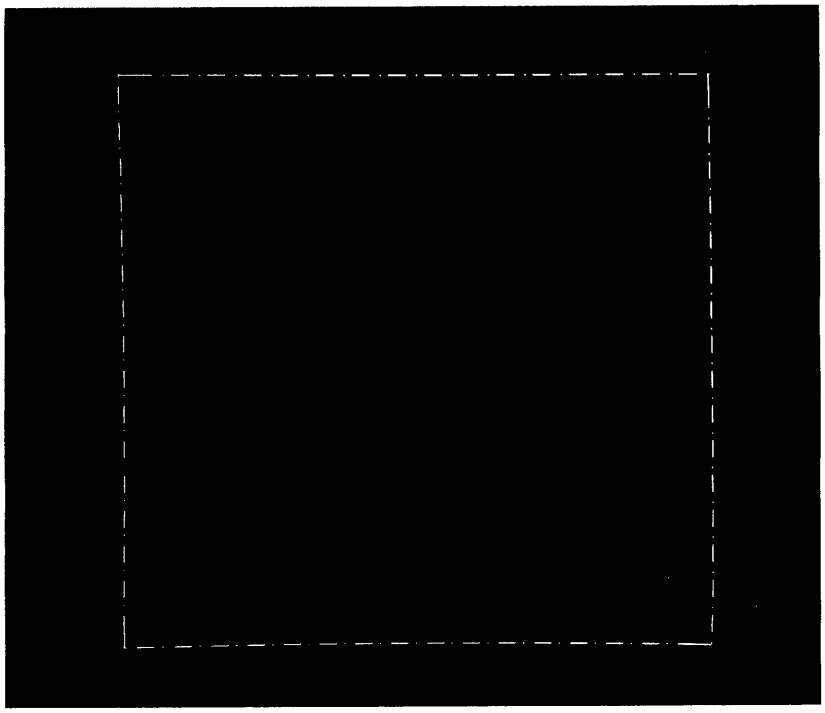

Figure 2-7 AVECT Subroutine

Routines called by AVECT: None.

Routines that call AVECT: None.

2.3.5 SVECT: Drawing a Vector in Short Format

Form: CALL SVECT (x,y[,l,i,f,t])

Arguments:

The x and y values will be added to the coordinates of the current beam position to produce the destination of the short vector; l, i, f, and t are the optional display parameters described in Section 1.4.4.

Instructions:

SVECT draws a vector from the current beam position to a point relative to the current beam position. If the current beam position is represented by (x0,y0), then a call to SVECT will draw a line segment from (x0,y0) to (x0+x,y0+y).

SVECT always uses the short-vector format (one word for each vector) in drawing line segments.

The displacement resulting from an (x,y) specification cannot exceed 63 raster units in either the x or the y direction. The maximum values you can specify in the default coordinate system are thus (-63.,-63.) or (63.,63.). If a displacement exceeds 63 raster units, it will be truncated to 63 units by SVECT. You should use VECT or LVECT rather than SVECT if the line segment to be drawn may exceed 63 raster units in either direction.

In the call to SVECT, you can optionally specify new values for the display parameters l, i, f, and t.

Figure 2-8 shows the sort of small objects for whicn SVECT is useful.

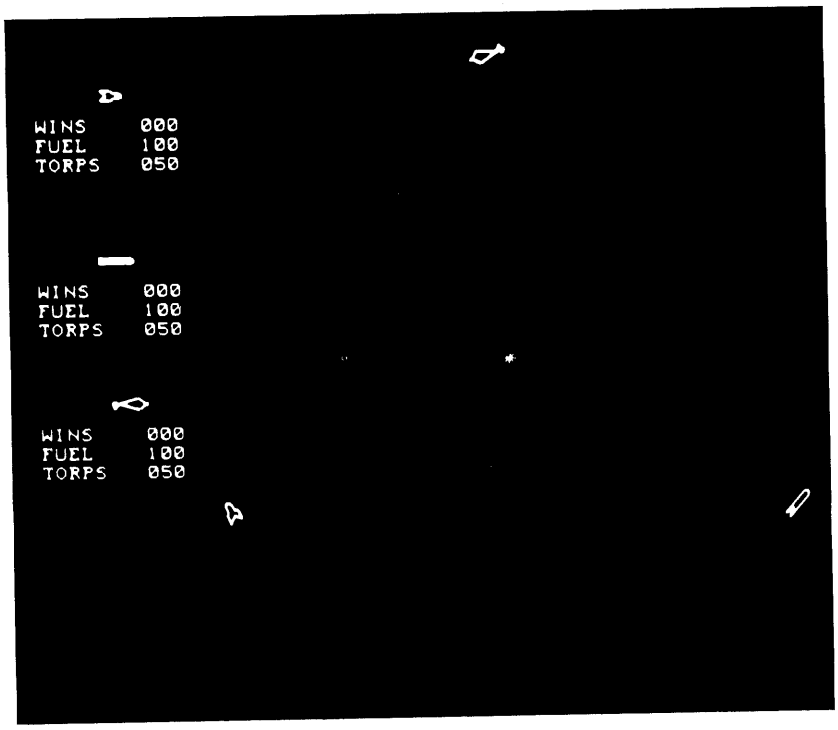

Figure 2-8 SVECT Subroutine

Routines called by SVECT: None.

Routines that call SVECT: None.

2.3.6 LVECT: Drawing a Vector in Long Format

Form: CALL LVECT (x,y[,l,i,f,t])

Arguments:

The x and y values will be added to the coordinates of the current beam position. The resulting new coordinates will be the destination of the long vector.

Instructions:

LVECT draws a vector from the current beam position to a point relative to the current beam position. If the current beam position is (x0,y0), then a call to LVECT will draw a line segment from (x0,y0) to (x0+x,y0+y).

LVECT always uses the long-vector format (two words for each vector) in drawing line segments.

The displacement resulting from an (x,y) specification cannot exceed 1023 raster units in either the x or the y direction. The maximum displacement in the default coordinate system is therefore (-1023.,-1023.) or (1023.,1023.).

You should use VECT rather the LVECT if the vector being drawn is of variable size and may be small enough for the short-vector format. The short-vector format uses only half the display-file space required by the long-vector format.

In the call to LVECT, you can optionally specify new values for the display parameters l, i, f, and t.

Figure 2-9 shows one of the features of the DRAW program described in Appendix D of this manual. The lines forming the triangle are long vectors that have been attached to the tracking object (see also Section 2.8.4 for a description of the ATTACH subroutine).

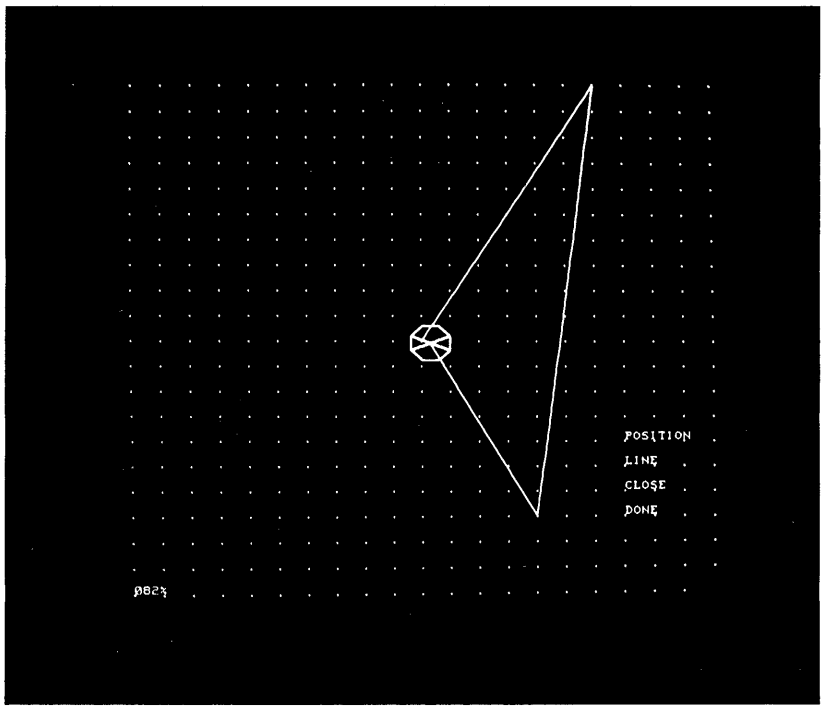

Figure 2-9 LVECT Subroutine

Routines called by LVECT: None.

Routines that call LVECT: None.

2.3.7 TEXT: Displaying a Text String

Form: CALL TEXT ([ictrl,]'a1'[[,ictrl],'a2,...'])

Arguments:

The values a1,a2,... are FORTRAN literal text strings (i.e., strings of characters enclosed within single quotation marks). The arguments represented by ictrl are optional control codes; these codes specify the mode in which the immediately following text string is to be displayed, and can also specify blank lines between text strings. If you do not include a control code, the text string will be displayed horizontally, with normal, nonitalic characters.

Instructions:

In a single call to TEXT, you can specify up to 10 arguments, including control codes. You can display more text strings by simply making multiple calls to TEXT. Each text string will be displayed with the lower left corner of the first character at the current beam position. After a string is displayed, the beam position will be the lower right corner of the last character.

Each TEXT call is a primitive. Therefore, when TEXT is called in a subpicture definition, a pointer can be set to the text primitive, which can then be changed with the CHANGT subroutine described in Section 2.6.6.

There are two basic character sets you can use for text strings:

- normal characters (a subset of ASCII printable characters)
- shift-out characters (Greek letters and special symbols)

A normal character string can consist of any number of printable characters with ASCII values greater than or equal to 40 (octal). These are the following:

- uppercase letters A through Z and lowercase letters a through z
- numbers 0 through 9
- a variety of special characters (see the examples in Section 1.1.3)

A string must be enclosed in single quotes within the TEXT call, as shown in the following example:

```
CALL TEXT ('ABCDEF')
```

The ictrl argument produces the following alternative displays to the normal text string:

- a count of blank lines to be displayed
- shift-out characters
- italics
- rotated characters (VS60 only)
- subscripts or superscripts (VS60 only)

NOTES ON SUBSCRIPTS AND SUPERSCRIPTS

1. Subscripts and superscripts are generated and positioned on the VS60 screen by a hardware character scaling feature. Each level of subscript or superscript is displayed in a character size one level smaller than the size of the characters preceding the subscript or superscript. The VS60 has four character sizes, and the normal character size is the second smallest of the four valid sizes (see the CVSCAL description, Section 2.4.8).

2. If you are displaying characters in the normal size, you can include a single subscript or superscript. In this case, the normal characters have a scaling factor of 2, and the subscript or superscript has a scaling factor of 1. If you want to specify multiple levels of subscripts or superscripts, as in the previous example, you must begin with a character size larger than the normal size of 2. For example, if characters have a scaling factor of 3, then the characters at the first level of subscripting will have a scale of 2, and the characters at the second level will have a scale of 1. Do not allow the character scaling factor to fall below 1. If you specify too many subscripts, the normal characters will be the wrong size when you return from subscript mode.

To display italic or rotated characters, include the appropriate negative code, followed by the character string to be displayed in the desired mode.

Position characters on the screen by moving the beam (for example, with APNT) to the desired coordinate position before calling TEXT. The lower left corner of the first character displayed will start at the current beam position.

Every string displayed by means of a TEXT call must be terminated with a null byte. A null character (ASCII code 0) is automatically appended to FORTRAN literal strings at compile time, but you must remember to include this byte explicitly when generating your own string data in arrays.

You cannot ordinarily display "invisible" characters on the display screen. Text output is always visible regardless of the current value of Display Parameter i, unless an entire subpicture containing characters is turned off (see Section 2.4.4).

In the TEXT call, you cannot specify new values for the display parameters l, i, f, and t, but you can call RPNT before the TEXT call, as shown here:

```
CALL RPNT (0.,0.,,-4)
CALL TEXT ('string')
```

Notice that a negative intensity level is specified in the RPNT call; this specification sets the intensity level for the text string, but does not display the point. Since the RPNT displacement is (0.,0.), the beam position will not be disturbed by the RPNT call.

Examples:

Figure 2-10 shows the TEXT features available to VT11 users.

```
COMMON/DFILE/IBUF(1000)
CALL INIT(1000)
CALL APNT(0.,600.,,-6)
CALL TEXT('NORMAL CHARACTERS: ABCDEF')
CALL TEXT(2,'ITALIC CHARACTERS:',-2,'ABCDEF')
CALL TEXT(2,'SHIFT-OUT CHARACTERS:',-1,'ABCDEF')
CALL TEXT(2,'ITALIC SHIFT-OUT:',-3,'ABCDEF')
STOP
END
```

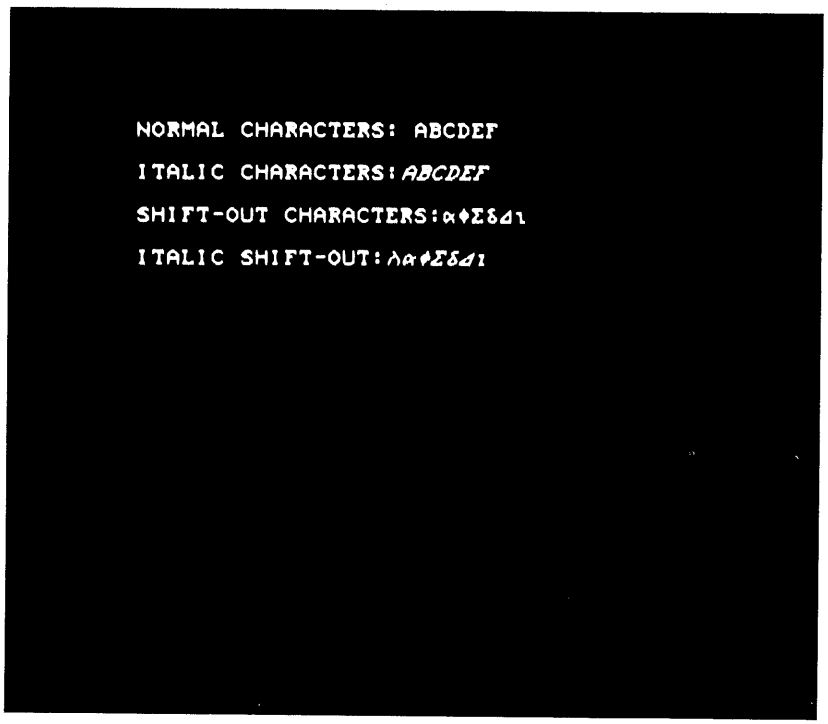

Figure 2-10 VT11 TEXT Features

The following example draws the axes for a graph and labels them in italic letters. Note that the label for the y axis is rotated (a unique VS60 feature). See Figure 2-11.

```
COMMON/DFILE/IBUF(1000)
CALL INIT(1000)
CALL APNT (100.,900.,,-4)
CALL LVECT (0.,-800.)
CALL LVECT (800.,0.)
CALL APNT (50.,200.,,-4)
CALL TEXT (-6,'POWER LOSS')
CALL APNT (200.,50.,,-4)
CALL TEXT (-2,'FREQUENCY')
STOP
END
```

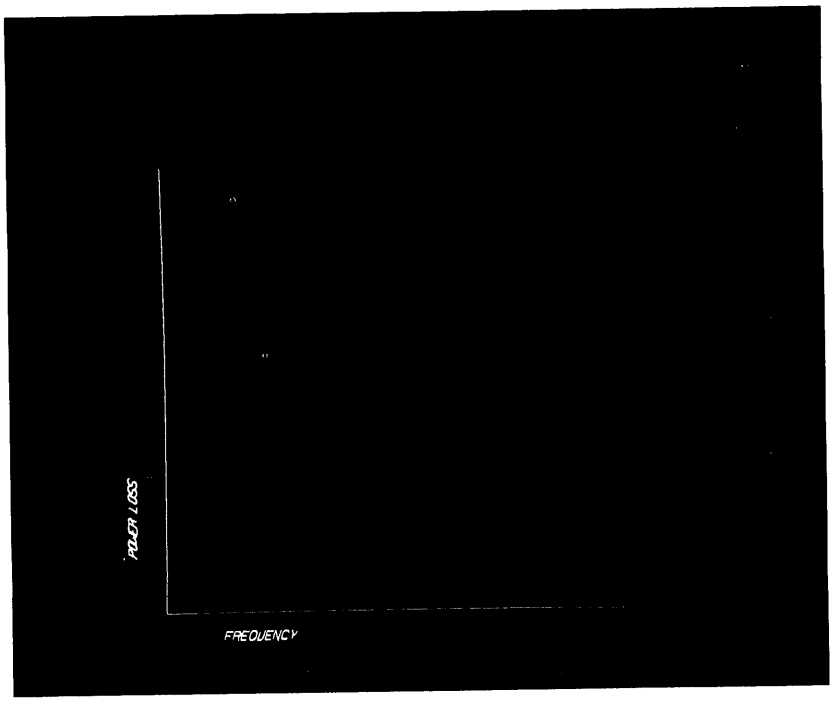

Figure 2-11 VS60 TEXT Features

Routines called by TEXT: None.

Routines that call TEXT: NMBR

2.3.8 MENU: Displaying Items in the Menu Area

Form: CALL MENU ([x0],y0,dy,m,'a1'[,'a2',...,'a10'])

Arguments:

The x0 and y0 arguments are the coordinates at which the first menu item will appear. If x0 is not supplied in your call, the first menu item will appear in the menu area at the distance y0 from the bottom of the screen. The vertical spacing between menu items is given by dy. The m argument is the subpicture tag of the first menu item. The menu items will be numbered sequentially, starting with m. The arguments a1,a2,...,a10 are FORTRAN literals, that is, character strings enclosed within single quotation marks (').

Instructions:

MENU displays a menu of light-pen-sensitive text strings on the display screen at Intensity Level 4. You can put as many as 10 menu items on the screen with a single MENU call, and you can make repeated calls to display more than 10 items.

You can also select the area of the screen in which the menu is to appear. If, for example, you supply the arguments (512.,512.) for (x0,y0), the first menu item will appear in the center of the main area. If you do not include a value for x0, the menu will automatically appear in the menu area at the right edge of the screen.

NOTES ON MENUS

1. If you decide to use "alphanumeric arrays" instead of literals for MENU arguments, the last element of each array must contain the ASCII code 0 (NUL).

2. After a MENU call, the VS60 display beam automatically returns to the main area.

The following call will produce the display shown in Figure 2-12.

```
CALL MENU (,512.,-100.,56,'ERASE','COPY','MOVE')
```

Figure 2-12 MENU Subroutine

The character strings of the items to be displayed in the menu are supplied in the a1,a2,...,a10 arguments. In the example above, the character strings 'ERASE', 'COPY', and 'MOVE' are the menu items. The dy argument represents the amount of vertical spacing between character strings displayed in the menu area. If the first item

begins at (x0,y0) and there are six items, then the sixth item will begin at (x0,y0+(6-1)*dy). In the example above, the first string is displayed starting at a y0 position of 512. Because dy is -100, the second item begins at (512+(2-1)*-100) or 412, and the third item at (512+(3-1)*-100) or 312.

The m parameter is the subpicture tag of the first item (a1) displayed in the menu area. In the example above, the tag representing character string 'ERASE' is 56. Successive character strings are tagged sequentially, so the second item, 'COPY', is 56+(2-1) or 57, and the third item, 'MOVE', is 56+(3-1) or 58.

Routines called by MENU: APNT, SUBP, ESUB, RPNT

Routines that call MENU: None.

2.4 DEFINING AND USING SUBPICTURES

The subroutines described in this section define subpictures that enable you to generate repeated images in graphics displays; to copy, erase, and move subpictures; to create special numeric subpictures; and to scale the size of characters and vectors used in subpictures.

There is no intrinsic limit on the number of primitives that you can define in a subpicture.

2.4.1 SUBP: Defining a Subpicture

Form: CALL SUBP (m1[,m2])

Arguments:

The m1 argument (1 to 32767) is the tag of the subpicture being defined by this SUBP call. If m2 is included in the call, it is the tag of a preexisting subroutine that is being assigned the new tag m1.

Instructions:

SUBP begins the definition of a new subpicture or repeats an existing subpicture. The new subpicture will have the tag m1; you can then use the tag m1 later in the program to refer to the entire group of primitives contained in the subpicture definition. For example, the entire subpicture can be turned on or off, moved from place to place on the screen, made sensitive to the light pen, or made to flash instead of being a constant image.

To define a new subpicture you must call SUBP in this form:

```
CALL SUBP (m1)
```

The graphic subroutine calls that follow this SUBP call insert the lines, points, and other images directly into the display file, making up a subpicture whose tag is m1.

All graphic subroutine calls following CALL SUBP (m1) will become part of Subpicture m1 until you call the ESUB subroutine (see Section 2.4.2) to terminate the subpicture definition. Thus, a series of subpicture definitions looks like this:

```
CALL SUBP (120)
   .
   .
   .
definition
   .
   .
   .
CALL ESUB
   .
   .
   .
CALL SUBP (130)
   .
   .
   .
definition
   .
   .
   .
CALL ESUB
```

The second, two-argument form of the SUBP call allows you to repeat an existing subpicture by referring to its tag. In the following form:

```
CALL SUBP (m1,m2)
```

The m1 argument is the tag of the new "instance" of a subpicture, and the m2 argument is the tag of the original subpicture.

Internally, the two-argument SUBP call does not actually duplicate the code of Subpicture m2 for use by Subpicture m1 (compare with the COPY subroutine described in Section 2.4.3). Instead, CALL SUBP (m1,m2) creates a "jump and return" instruction in the display file that executes the code for Subpicture m2 and gives the tag m1 to the resulting repeated image, or instance.

Because this two-argument form of SUBP does not begin a definition, do not include a corresponding call to ESUB.

You cannot address a subpicture instance with the POINTR subroutine. Instead, you must address the original subpicture.

A new subpicture is often constructed by repeating several existing subpictures. In other words, the DECgraphic-11 FORTRAN Graphics Package allows you to "nest" subpicture definitions. A single subpicture definition can nest up to seven additional subpictures. The first SUBP call in this case should be of the one-argument form; it will define the large subpicture containing the nested ones.

Be sure to include one call to ESUB for every definition call to SUBP, that is, every call that contains only a single argument. The first call to ESUB encountered in the code will end the last subpicture created, as in a typical subroutine structure.

The following segment illustrates the nesting of four subpictures (including the "main" subpicture) in a FORTRAN program.

```
CALL SUBP (201)          begins definition of 201
CALL SUBP (202,721)      repeats 721
CALL SUBP (203)          begins definition of 203
     .
     .
     .
definition
     .
     .
     .
CALL SUBP (204,771)      repeats 771
CALL ESUB                terminates 203 definition
CALL ESUB                terminates 201 definition
```

The subpictures that precede a nested subpicture are called its "precedents." In this example, the first precedent of Subpicture 204 is 203, the second precedent is 202, and the third is 201.

Nested subpictures are useful in applications in which different parts of a graphic display are frequently repeated or tested for light-pen hits, both individually and as a group. The following error message is generated if more than eight subpictures are nested:

```
MORE THAN 8 NESTED SUBP
```

Two examples of the use of SUBP are included at the end of the description of the ESUB subroutine (see Section 2.4.2).

Routines called by SUBP: TOSAT (h-s)

Routines that call SUBP: MENU, NMBR, COPY, FIGR

2.4.2 ESUB: Terminating a Subpicture

Form: CALL ESUB [(m)]

Arguments:

The optional argument m not only identifies the particular subpicture being ended but also (in VS60 subsystems) restores the display parameters that were in effect before the referenced SUBP call was issued.

The m argument can be any integer; the ESUB call will always operate on the subpicture that was defined most recently. However, it is always a good practice to match the value of m with the tag of the subpicture being terminated.

Instructions:

In VS60 subsystems, the optional argument m is useful because you will want to change the display parameters frequently even in simple graphic programs. Remember that setting one of the display parameters (l, i, f, and t), either in a primitive call like VECT or in a special subroutine like SENSE, will change the display parameter for the primitive in question and for all following primitives.

It is unlikely that you will want all the primitives in a subpicture, or even most of them, to have the same display parameters. For instance, many interactive programs have only a minority of the vectors and menu items sensitized to the light pen. With a VS60 display subsystem, ESUB can thus be used to confine the effect of a certain display-parameter setting to a single subpicture, which could contain only one primitive or could define a complex picture. Such a subpicture is terminated with an ESUB call that includes the m argument. Since this type of ESUB call restores the display parameters to their prior settings, it is called a "restoring ESUB." Especially in combination with the nested subpicture feature, the restoring ESUB is a powerful tool for graphic programming.

VT11 users can accomplish the same effect as a restoring ESUB by remembering to change the display parameters back to the proper values at critical points in the program. One way to do so is with an "invisible point," as follows:

```
CALL RPNT(0.,0.,1,-4,1,1)
```

Because the RPNT call has a 0 displacement, it will not affect the position of the beam. The -4 value for i means that the relative point will be invisible but the i value for all subsequent primitives will be +4. The RPNT call also has set l=1, f=1, and t=1. These new display parameters will stay in effect until you change them again. Although this method is more complicated than a restoring ESUB, it accomplishes the same effect, since in a program for a VT11 subsystem the RPNT call can be placed immediately after an ESUB. You will have to remember the values to which the display parameters must be restored.

This example shows repeated subpictures (see Figure 2-13). Note that any change to the original subpicture will affect any instance of the subpicture.

```
      COMMON/DFILE/IBUF(1000)
      CALL INIT(1000)
      CALL APNT(100.,512.,,-4)
      CALL SUBP(100)
      CALL VECT(100.,0.)
      CALL VECT(0.,100.)
      CALL VECT(-100.,-100.)
      CALL ESUB
      CALL SUBP(1100)
      CALL RPNT(150.,0.,,-4)
      CALL SUBP(1101,100)
      CALL RPNT(150.,0.,,-4)
      CALL SUBP(1102,100)
      CALL RPNT(150.,0.,,-4)
      CALL SUBP(1103,100)
      CALL APNT(900.,512.,,-4)
      CALL SUBP(1104,100)
      CALL ESUB
      STOP
      END
```

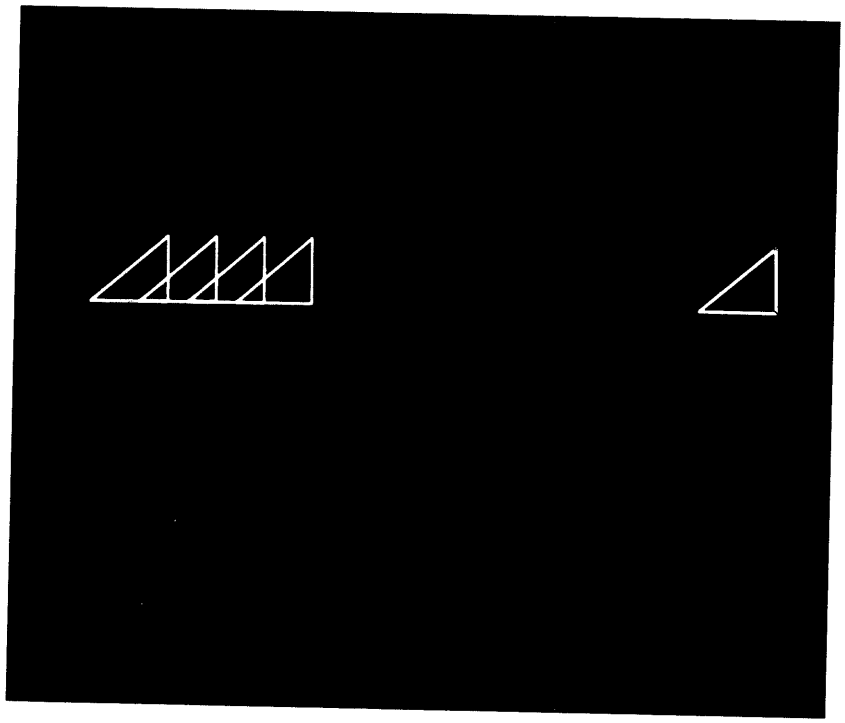

Figure 2-13 Repeated Subpictures

This example (see Figure 2-14) shows the distinction between a regular ESUB call and a restoring ESUB call on the VS60. The bright figure (third from the left) is at a higher intensity level than the previous figure, but the bright figure is also nested in a subpicture definition that ends with a restoring ESUB. Thus, the fourth figure is restored to the lower intensity level.

```
      COMMON/DFILE/IBUF(1000)
      REAL ENDPT(16)
      DATA ENDPT/0.,100.,100.,0.,0.,-100.,-100.,0.,100.,100.,
     X  -50.,50.,-50.,-50.,100.,-100./
      CALL INIT(1000)
      CALL SUBP(1)
      CALL APNT(100.,512.,,-4)
      CALL FIGR(ENDPNT,16,100)
      CALL ESUB
      CALL SUBP(2)
      CALL APNT(250.,512.,,-4)
      CALL SUBP(101,100)
      CALL ESUB
      CALL SUBP(3)
      CALL APNT(400.,512.,,-4)
      CALL SUBP(102,100)
      CALL ESUB(3)
      CALL APNT(550.,512.,,-10)
      CALL FIGR(ENDPNT,16,200)
      CALL POINTR(1,3)
      CALL INTENS(1,8)
      STOP
      END
```

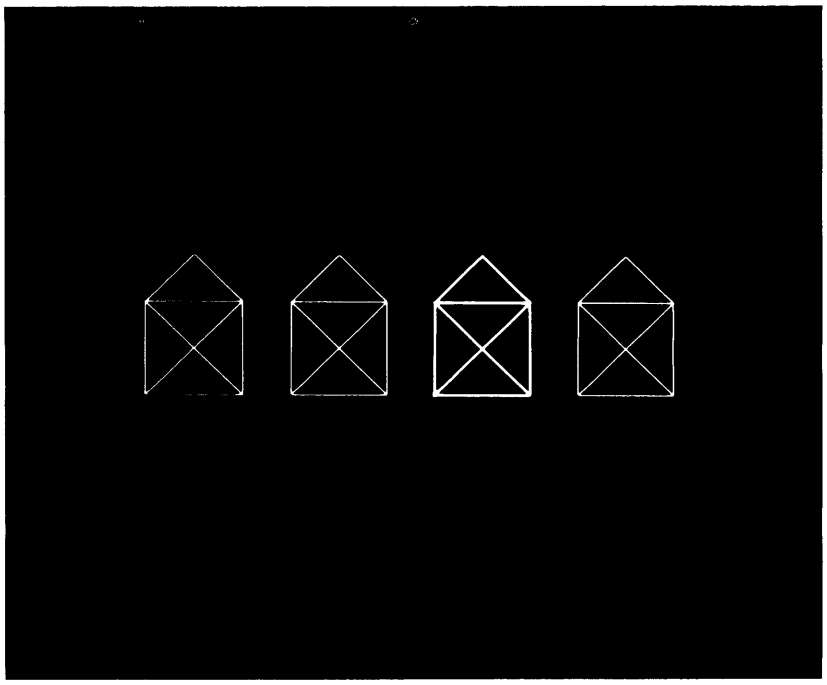

Figure 2-14 Restoring vs. Nonrestoring ESUB

Routines called by ESUB: STOP, CONT, TOSAT (h-s)

Routines that call ESUB: MENU, NMBR, COPY, FIGR

2.4.3 COPY: Copying a Subpicture

Form: CALL COPY ([m1],m2)

Arguments:

The argument m2 is the tag of an existing subpicture. If m1 is included, it is the tag assigned to the copy of Subpicture m2. If m1 is omitted from the call, Subpicture m2 is copied to the currently open subpicture.

Instructions:

COPY creates a duplicate of Subpicture m2. This duplicate subpicture is different from the repeated instance of an existing subpicture, created by a two-argument SUBP call. A duplicate of a subpicture, created by COPY, will not be affected when you change a primitive or display parameter in the original subpicture. In contrast, a change to a subpicture will also change all instances of that subpicture created by a two-argument SUBP. A new subpicture created by COPY can be addressed by the POINTR subroutine; the subpicture instances created by SUBP cannot.

The COPY call can either assign a new tag to the copied subpicture or, if COPY is included within a subpicture definition, can let the defining subpicture's tag be used. For example, in the following:

```
CALL SUBP (122)
CALL COPY (,701)
CALL ESUB
```

Subpicture 122 has been "opened" for definition, and Subpicture 701 is copied and assigned the tag 122. An alternative way of expressing this operation is:

```
CALL COPY (122,701)
```

NOTE

As with SUBP, coordinate references in the subpicture being copied should be to relative, not absolute, positions on the display screen. Avoid using APNT and AVECT in subpictures to be copied.

Example:

This example shows an original subpicture and four copies. Notice that each of the copied subpictures can be addressed by the POINTR subroutine, and so each can be given its own line type. (The original subpicture is the one in the center of the screen; see Figure 2-15.)

```
      COMMON/DFILE/IBUF(1000)
      REAL ENDPT(16)
      DATA ENDPNT/0.,100.,100.,0.,0.,-100.,-100.,0.,100.,100.,
     X  -50.,50.,-50.,-50.,100.,-100./
      CALL INIT(1000)
      CALL APNT(512.,512.,,-4)
      CALL FIGR(ENDPNT,16,100)
      CALL APNT(0.,850.,,-4)
      CALL COPY(101,100)
      CALL APNT(900.,850.,,-4)
      CALL COPY(102,100)
      CALL APNT(900.,0.,,-4)
      CALL COPY(103,100)
      CALL APNT(0.,0.,,-4)
      CALL COPY(104,100)
      PAUSE 'TYPE <RET> TO CHANGE LINE TYPES'
      CALL POINTR(14,101)
      CALL LINTYP(14,1)
      CALL POINTR(15,102)
      CALL LINTYP(15,2)
      CALL POINTR(16,103)
      CALL LINTYP(16,3)
      CALL POINTR(17,104)
      CALL LINTYP(17,4)
      CALL POINTR(18,100)
      CALL FLASH(18)
      CALL FLASH(14,-1)
      STOP
      END
```

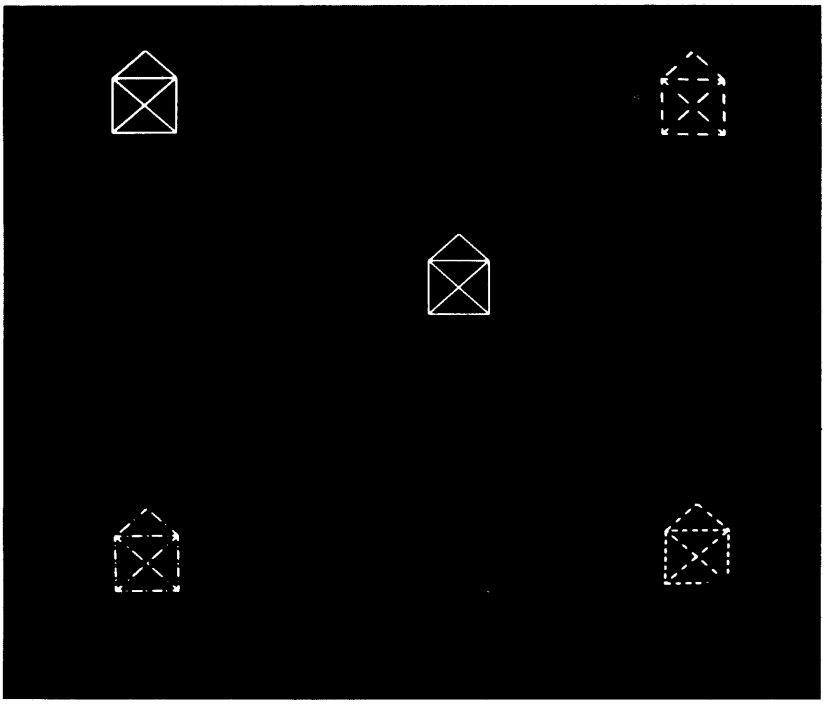

Figure 2-15 Copied Subpictures

Routines called by COPY: SUBP, ESUB, TOSAT (h-s)

Routines that call COPY: None.

2.4.4 OFF: Turning Off a Subpicture

Form: CALL OFF (m)

Arguments:

The m argument is the tag of the subpicture being turned off.

Instructions:

OFF turns off the subpicture whose tag is m. When a subpicture is turned off, there will be two effects on the display:

1. The graphic calls in the subpicture will not produce any output on the screen.

2. If Subpicture m is terminated by a nonrestoring ESUB (i.e., one without an m parameter), the display parameters (e.g., light-pen interaction, intensity level, line type) will have the same values as before the subpicture was defined. The beam will also be returned to its former position.

Because of the second condition, any call to OFF should be followed by a call to RPNT or APNT to set the beam position and change line type and other display parameters, unless the subpicture was ended with a restoring ESUB (VS60 only) or unless it is desirable to have these changes occur.

When a subpicture is turned off, the definition of the subpicture remains in the display file; therefore, you can copy the subpicture even though it is turned off. A turned-off subpicture can be turned on again by the ON subroutine (see Section 2.4.5). Attempting to turn off a subpicture that is already turned off has no effect.

To increase system speed, you can turn off a subpicture before it has been completely defined, that is, between the SUBP and ESUB calls that define it. This action is often useful when building a display that should be seen only when its construction is complete.

An example of using OFF is included in the description of the ON subroutine (see Section 2.4.5).

Routines called by OFF: STOP, CONT, TOSAT (h-s)

Routines that call OFF: None.

2.4.5 ON: Turning On a Subpicture

Form: ON (m)

Arguments:

The m argument is the tag of the subpicture to be turned on.

Instructions:

ON turns on the subpicture whose tag is m, and will therefore redisplay a subpicture that has been turned off by the OFF subroutine. Attempting to turn on a subpicture that is already on has no effect.

Example:

This example builds Subpicture 1500 from the data in arrays x and y. The subpicture is turned off until the entire picture has been constructed, and then the display is turned on. This technique gives you the facility to display "all-at-once" pictures, as well as to show pictures "growing" as new primitives are inserted in the display file.

```
        .
        .
        .
     CALL APNT (0.,500.,,-4)
     CALL SUBP (1500)
     CALL OFF (1500)
     DO 100 I=1,50
100  CALL VECT (X(I),Y(I))
     CALL ESUB (1500)
     CALL ON (1500)
        .
        .
        .
```

Routines called by ON: TOSAT (h-s)

Routines that call ON: None.

2.4.6 ERAS: Erasing a Subpicture

Form: CALL ERAS [(m)]

Arguments:

The optional argument m is the tag of the subpicture that will be erased. If m is not supplied, ERAS will erase only the tracking object (see Section 2.8) and will not affect any subpicture.

Instructions:

ERAS permanently erases Subpicture m from the screen. Note the difference from OFF, which turns off the subpicture, but only temporarily.

Since ERAS effectively eliminates the definition using tag m from the display file, tag m is again available for use in new definitions. However, erasing a subpicture does not automatically reclaim the space in the display file that was used for the subpicture definition. You can reclaim this space by compressing the display file; use either CMPRS (see Section 2.13.1) or SAVE (see Section 2.13.2).

NOTE

Erasing a subpicture also erases all instances of it made with two-argument SUBP calls.

Example:

This example creates a new subpicture whose tag is 25, copies into it the contents of Subpictures 100 and 200, and then erases 100 and 200. This action allows the space used by 100 and 200 to be reclaimed by the CMPRS subroutine. Note that Subpicture 25 is turned off until Subpictures 100 and 200 are erased; this action will prevent the two subpictures from appearing overly bright because of the display of multiple images at the same screen location.

```
      .
      .
      .
CALL SUBP (25)
CALL OFF (25)
CALL COPY (,100)
CALL COPY (,200)
CALL ESUB (25)
CALL ERAS (100)
CALL ERAS (200)
CALL ON (25)
      .
      .
      .
```

Routines called by ERAS: STOP, CONT, TOSAT (h-s)

Routines that call ERAS: None.

2.4.7 NMBR: Creating a Numeric Subpicture

Form: CALL NMBR (m,var[,n,'format'])

Arguments:

The m argument is the tag that will be assigned to the numeric subpicture, and var is the name of the FORTRAN REAL variable to be displayed. The optional arguments n and format give the field width and FORTRAN field descriptor (e.g., F7.2), respectively, for the variable to be displayed. The format argument is a FORTRAN "literal" string, and must therefore be enclosed in single quotation marks ('). The format argument must consist of every character that normally follows the word FORMAT in a FORMAT statement, including parentheses. Thus, the statement

```
CALL NMBR(100,XVALUE,7,'(F7.2)')
```

is a legal call to NMBR; the statement

```
CALL NMBR(100,INTER,4,'I4')
```

is illegal because there are no parentheses in the format argument. The default format for NMBR is F16.8.

Instructions:

NMBR creates a special numeric subpicture that can be displayed on the screen in any FORTRAN REAL format and can be updated in "odometer" fashion. The number is displayed with a maximum field width of 16, in any legal real format that you specify.

The NMBR call can have either two or four arguments. The simpler form of the call:

```
CALL NMBR (m,var)
```

creates a numeric subpicture with tag m and displays the value of var in the default format. This default is originally F16.8, but the default will be changed if you call NMBR with all four arguments. For integer-only libraries, the default format of NMBR is I6.

When you use all four arguments, the format specification is a literal string:

```
CALL NMBR (200,X2(I),8,'(F8.2)')
```

If you do not provide the format specification in an NMBR call, the number is displayed in the format last specified in an NMBR call in the program. The second NMBR call below will display the value of B in format F10.4.

```
CALL NMBR (101,A,10.'(F10.4)')
   .
   .
   .
CALL NMBR (102,B)
```

You can update the number displayed by NMBR by calling NMBR a second time with only the m and var arguments. In this case, m is the tag of

the same NMBR subpicture that was created previously; var is the same variable name as before, with a new value having been assigned to it. However, see the alternative technique described in Section 3.1.4.

As with other types of subpictures, you can position the numeric subpicture on the screen by moving the beam to the desired location (with APNT, for example) before calling NMBR. The lower left corner of the first digit displayed will start at the current beam position.

This example (see Figure 2-16) shows a typical NMBR subpicture.

```
        COMMON/DFILE/IBUF(1000)
        DATA PI/3.14159/
        REAL R(51)
        DO 1 I=1,51
1       R(I)=200.+200.*SIN(PI*(I-1)/25.)
        CALL INIT(1000)
        CALL APNT(512.,512.,,-6)
        CALL NMBR(100,R(1),7,'(F7.2)')
        DO 700 I=1,50
700     CALL NMBR(100,R(I+1))
        STOP
        END
```

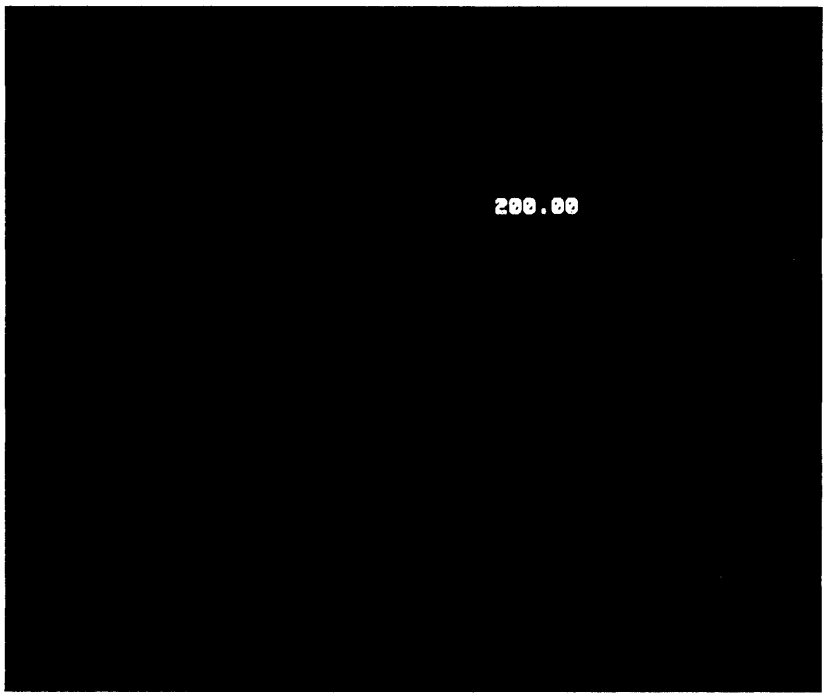

Figure 2-16 NMBR Subroutine

A further example of using NMBR is included in the description of the CMPRS subroutine (see Section 2.13.1).

Routines called by NMBR: SUBP, TEXT, ESUB, POINTR, CHANGT, TOSAT (h-s), FRSAT (h-s)

Routines that call NMBR: None.

2.4.8 CVSCAL: Scaling Subpicture Characters and Vectors

Form: CALL CVSCAL (m[,ifc,ifv])

Arguments:

The m argument is the tag of the VS60 subpicture to be scaled. If supplied, ifc (1 to 4) and ifv (1 to 15) are the scaling factors for characters and vectors, respectively. The ifc and ifv arguments do not have to be supplied in the same call, but CVSCAL will have no effect if both are omitted.

Instructions:

CVSCAL calls can be included in VT11 programs, but they will have no effect.

This subroutine allows you to enlarge or contract the characters and vectors on the display screen. This feature is helpful in creating charts or figures whose components can be enlarged for more detailed examination, or in creating and labeling pictures that will then be "shrunk" for inclusion in a larger structure.

The value of ifc must be an integer in the range 1 through 4, where each number changes the character size by a factor of 2. The normal size of characters corresponds to ifc=2.

ifc Value	Character Size
1	1/2 normal size
2	normal size
3	1 1/2 normal size
4	2 times normal size

The ifv argument is the vector scaling factor. The value of ifv must be an integer in range 1 through 15, and vectors are scaled in increments of one quarter. The normal vector size corresponds to ifv=4.

ifv Value	Vector Size
1	1/4 normal size
2	1/2 normal size
3	3/4 normal size
4	normal size
5	1 1/4 normal size
6	1 1/2 normal size
7	1 3/4 normal size
8	2 times normal size
9	2 1/4 normal size
10	2 1/2 normal size
11	2 3/4 normal size
12	3 times normal size
13	3 1/4 normal size
14	3 1/2 normal size
15	3 3/4 normal size

In addition to scaling the characters or vectors in the specified subpicture, CVSCAL will scale the rest of the display file, unless you terminate the subpicture definition with a restoring ESUB call (see Section 2.4.2).

A problem can occur if you include both VIEWPT (see Section 2.2.5) and CVSCAL in the same subpicture. The CVSCAL call within the subpicture places the scaling instruction before all other instructions in the subpicture, including the VIEWPT instruction. If you want to perform both a VIEWPT and a CVSCAL operation on a set of primitives, then do not include CVSCAL and VIEWPT in the same subpicture.

NOTE

When you use CVSCAL to scale the characters or vectors within a subpicture, the display screen may become blank for a moment, and then reappear with the changed characters or vectors. A technique for avoiding this occurrence is given in Section 3.2.4.

Example:

This example (see Figure 2-17) displays the string 'COMMANDS:' in characters twice the normal size. It then displays 'SAMPLE', 'ANALYZE', 'PLOT', and 'STOP' in a column beneath 'COMMANDS:' in characters of normal size.

```
      COMMON/DFILE/IBUF(1000)
      CALL INIT(1000)
      CALL APNT (0.,800.,,-4)
      CALL SUBP (10)
      CALL TEXT ('COMMANDS:',1)
      CALL ESUB (10)
      CALL CVSCAL (10,4)
      CALL TEXT (4,'SAMPLE',1,'ANALYZE',1,
     X 'PLOT',1,'STOP')
      STOP
      END
```

Figure 2-17 CVSCAL Subroutine

Routines called by CVSCAL: POINTR, INSRT, TOSAT (h-s)

Routines that call CVSCAL: None.

2.5 DISPLAYING GRAPHS AND FIGURES

The subroutines described in this section allow you to make single calls that generate more complex structures on the display screen than the primitive elements described in Section 2.3. These subroutines generate subpictures that contain entire graphs and figures. Because these graphs and figures are subpictures, you can erase them and turn them on and off. You can also modify a displayed graph or figure by changing the values of certain primitives defined in the special subpictures.

2.5.1 XGRA: Displaying an X-Value Graph

Form: XGRA (dy,ARRAY,n,m[,l,i,f,t])

Arguments:

The dy argument is the vertical spacing between points on the graph. The maximum value for dy is 64 raster units. Be sure to adjust the value of dy to agree with your current scaling factor. The array contains real-valued x offsets that will be converted to the x coordinates of points. ARRAY is the name of any legally dimensioned FORTRAN array. The first n elements from ARRAY are used. The entire graph is a subpicture with tag m. You can optionally set new values for the display parameters l, i, f, and t.

Instructions:

XGRA creates a special graph subpicture (with tag m) that consists of a series of points. Each point is a primitive. XGRA allows you to display a graph whose y coordinates are evenly spaced on the display screen.

The array must be a one-dimensional real array.

The y coordinates of the graph displayed by XGRA are generated from integral multiples of dy. The x coordinate of the first point is the first element of the array plus the x value of the current beam position; the first y coordinate is at the current y value plus the value of dy.

The subpicture created by XGRA can be changed with the POINTR, CHANGE, and CHANGA subroutines (see Section 2.6). The y coordinate in a CHANGE or CHANGA call is ignored when these subroutines are used on an XGRA primitive. The GET subroutine can return the coordinates of individual points.

When a graph subpicture is created by means of XGRA, the array elements are converted to x coordinates before the subpicture is entered in the display file, so that the array is reusable.

Example:

Figure 2-18 shows a sine function plotted with XGRA.

```
        COMMON/DFILE/IBUF(1000)
        REAL R(51)
        DATA PI/3.14159/
        DO 1 I=1,51
1       R(I)=200.+200.*SIN(PI*(I-1)/25.)
        CALL INIT(1000)
        CALL APNT(512.,0.,,-4)
        CALL XGRA(20.,R,51,1000)
        STOP
        END
```

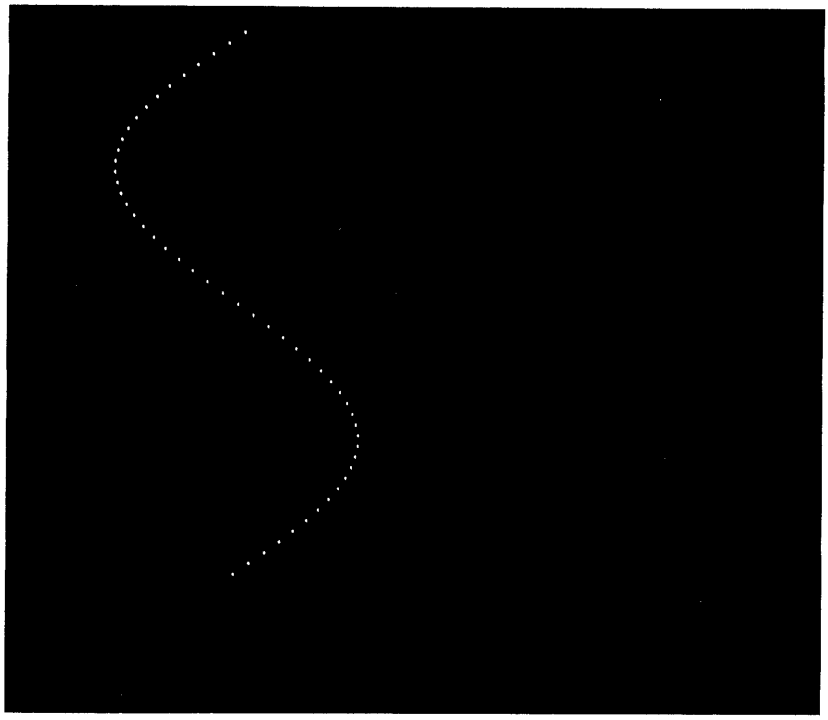

Figure 2-18 XGRA Subroutine

Routines called by XGRA: None.

Routines that call XGRA: None.

2.5.2 YGRA: Displaying a Y-Value Graph

Form: CALL YGRA (dx,ARRAY,n,m[,l,i,f,t])

Arguments:

The dx argument is the horizontal spacing between points of the graph. The maximum value for dx is 64 raster units. Be sure to adjust the value of dx to agree with your current scaling factor. ARRAY is the name of any legally dimensioned FORTRAN array. The graph is constructed by using the first n elements of the array as y offsets. As with XGRA, the entire graph is a subpicture with tag m, and you can optionally set new values for the display parameters l, i, f, and t.

Instructions:

YGRA creates a special graph subpicture with tag m, which consists of a series of points. Each point is a primitive. YGRA allows you to display a graph whose x coordinates are evenly spaced on the display screen.

The y coordinates are computed by adding the values of the array elements to the y coordinates of the current beam position. As with XGRA, the array is reusable since its elements are not changed by the call.

The array must be a one-dimensional real array.

The x coordinates of the graph displayed by YGRA are generated from integral multiples of dx. The first x coordinate is at the x coordinate of the current beam position plus dx.

Like XGRA, YGRA creates a subpicture whose primitives (the individual points) can be examined and changed by the GET, CHANGA, and CHANGE subroutines. The x coordinate in a CHANGE or CHANGA call is ignored when these subroutines are used on a YGRA subpicture.

Example:

This example draws a sine wave across the entire screen using 51 points (see Figure 2-19).

```
      COMMON/DFILE/IBUF(1000)
      REAL R(51)
      DATA PI/3.14159/
      DO 1 I=1,51
1     R(I)=200.+200.*SIN(PI*(I-1)/25.)
      CALL INIT(1000)
      CALL YGRA(20.,R,51,1000)
      STOP
      END
```

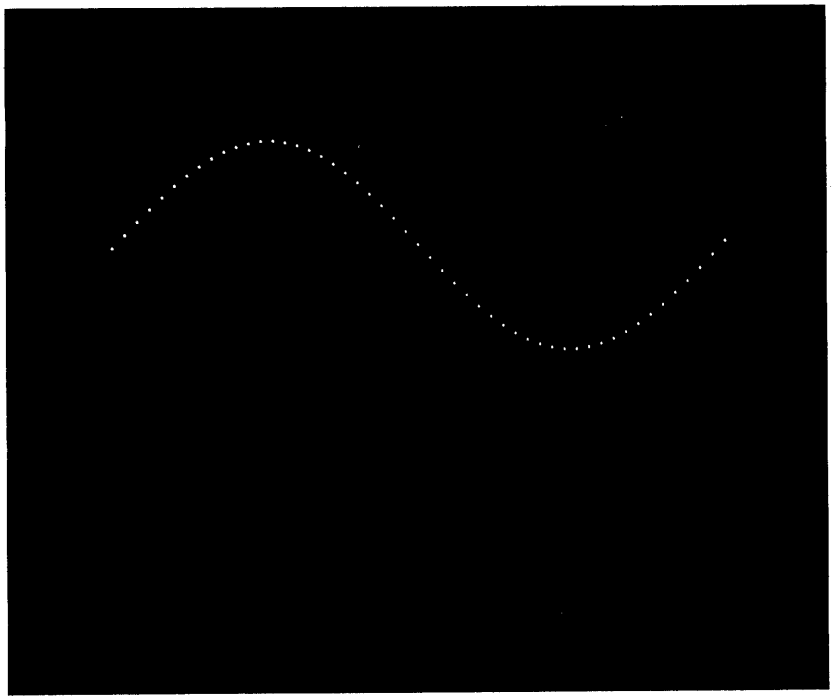

Figure 2-19 YGRA Subroutine

Routines called by YGRA: None.

Routines that call YGRA: None.

2.5.3 FIGR: Displaying a Figure

Form: CALL FIGR (ARRAY,n,m[,l,i,f,t])

Arguments:

The array contains real values that, when added to the current beam position, form the endpoints of relative (long) vectors in the figure subpicture. The first n elements of the array are used for the figure. The tag of the figure subpicture is m. You can optionally set new values for the display parameters l, i, f, and t.

Instructions:

FIGR creates a special figure subpicture with tag m from an array of (x,y) pairs. If, for example, the array name is A, the first (x,y) pair is formed by A(1) and A(2). If the starting beam position is (x0,y0), the first vector is drawn from (x0,y0) to (A(1)+x0,A(2)+y0); this endpoint is also the new beam position and will therefore be the starting point of the next vector.

FIGR uses the long-vector format (two words for each vector).

The individual primitives in the FIGR subpicture can be examined and changed by the GET, CHANGA, and CHANGE subroutines (see Section 2.6). For example, the eighth (x,y) pair is Primitive 8.

Example:

This example (see Figure 2-20) draws a crossed box in the center of the screen.

```
      COMMON/DFILE/IBUF(1000)
      REAL ENDPNT(16)
      DATA ENDPNT/0.,100.,100.,0.,0.,-100.,-100.,0.,100.,100.,
     X -50.,50.,-50.,-50.,100.,-100./
      CALL INIT(1000)
      CALL APNT(512.,512.,,-4)
      CALL FIGR(ENDPNT,16,100)
      STOP
      END
```

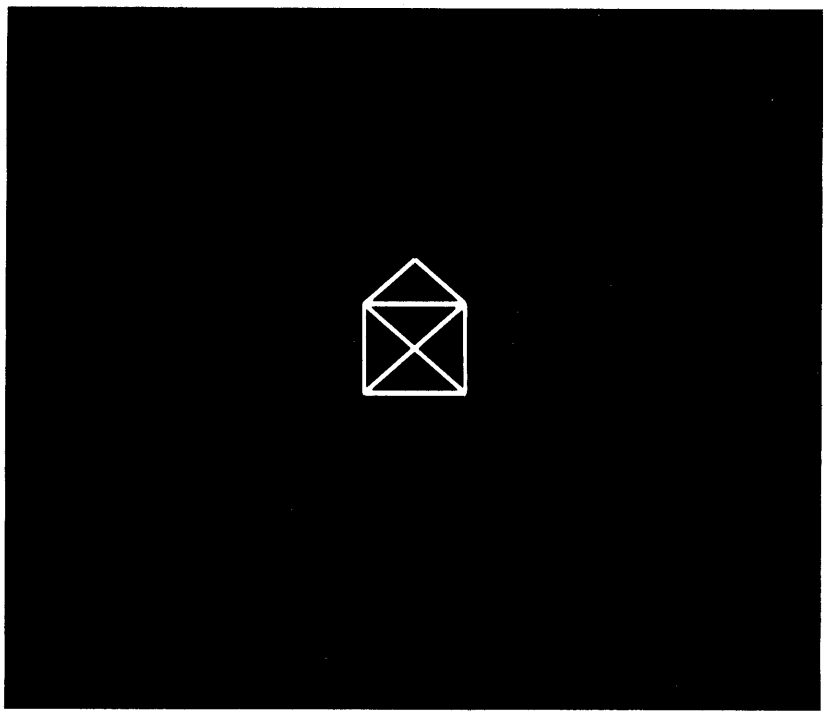

Figure 2-20 FIGR Subroutine

Routines called by FIGR: SUBP, ESUB

Routines that call FIGR: None.

2.6 USING DISPLAY-FILE POINTERS

The subroutines described in this section allow you to use pointers to identify individual primitives in a subpicture.

You can then change the values of the primitives referenced by these pointers. The first three subroutines in this category, POINTR, ADVANC, and GET, do not display graphics on the screen. They allow you to identify, skip, and return information on primitives.

CHANGE, CHANGA, CHANGT, INSRT, and ERASP let you change, add, and erase primitives that have been identified by a pointer. They will have an immediate effect on the display, making primitives appear or disappear instantly.

CAUTION

There are 21 distinct pointers in the DECgraphic-11 system. However, only Pointers 1-20 can be used in your subroutine calls. Pointer 21 is the system pointer and should never be used.

2.6.1 POINTR: Setting Up a Pointer

Form: CALL POINTR (k,m[,j])

Arguments:

The k argument is the pointer number (1-20), m is the tag of the subpicture containing the target primitive, and j (optional) is the number of the primitive within the subpicture. If you omit j, the pointer will go to the first primitive in the subpicture. If j is greater than the number of primitives in the subpicture, the pointer will point to the position following the last primitive in the subpicture.

Instruction:

POINTR allows you to identify an individual primitive by its subpicture tag and numerical order within the subpicture. Once you have singled out a primitive in this way, you can examine its coordinates (the GET subroutine), change the coordinates (CHANGE and CHANGA), insert new primitives (INSRT), or erase the primitive (ERASP). If the primitive is a TEXT call, you can change its value with CHANGT. You can also advance the pointer from one primitive to the next with ADVANC.

NOTE

Pointers never point to erased subpictures, nested SUBP calls, or erased primitives. When these elements are encountered in the display file, the pointers skip them automatically.

One example of using POINTR is included below. Others are given in the descriptions of GET (see Section 2.6.3) and GRID (see Section 2.8.6).

Example:

This example draws a vector that continuously sweeps out a circle in the center of the screen. The vector moves in a counterclockwise direction.

```
        .
        .
        .
      CALL APNT (512.,512.,,-4)
      CALL SUBP (100)
      CALL VECT (500.,0.)
      CALL ESUB
        .
        .
        .
      CALL POINTR (3,100)
      ANGLE=0.
10    CALL CHANGE (3,500.*COS(ANGLE),500.*SIN(ANGLE))
C
C     SLOW IT DOWN
C
      DO 20 I=1,100
20    CONTINUE
C
C     PROCEED
C
      ANGLE=AMOD (ANGLE+.01,6.28)
      GO TO 10
        .
        .
        .
```

Routines called by POINTR: ADVANC, TOSAT (h-s)

Routines that call POINTR: NMBR, CVSCAL, LPEN

2.6.2 ADVANC: Advancing a Pointer

Form: CALL ADVANC (k[,n])

Arguments:

The k argument is the pointer number (1 to 20), and the optional n argument is the number of primitives to advance from the pointer's present position (n must be a positive integer). If you omit n, the pointer will advance to the next primitive.

Instructions:

A pointer cannot be advanced beyond the end of a subpicture definition in the display file. If an ADVANC call attempts to move a pointer beyond the subpicture boundary, the pointer will point to the position following the last primitive in the subpicture.

ADVANC will skip any nested subpicture definitions or erased primitives in the subpicture being examined.

An example using ADVANC is included in the description of the GET subroutine (see Section 2.6.3).

Routines called by ADVANC: TOSAT (h-s)

Routines that call ADVANC: POINTR, CHANGA, LPEN

2.6.3 GET: Returning the Coordinates of a Primitive

Form: CALL GET (k,X,Y)

Arguments:

The k argument is the pointer number (1 to 20). X and Y are REAL dummy variables that will return the coordinates of the primitive pointed to by Pointer k.

Instructions:

One common use of GET is to dynamically change the appearance of a display by retrieving (GET) and changing (CHANGE or CHANGA) the coordinates of certain primitives.

Example:

This example draws a sine wave of dots and then shrinks it one point at a time, using Pointer 4 to move through the subpicture definition. (See Figure 2-21.)

```
        COMMON/DFILE/IBUF(1000)
        CALL INIT(1000)
        CALL SUBP (1000)
        X=0
        DO 1000 I=1,51
        CALL APNT (X,SIN(.125*(I-1))*500.+500.)
 1000   X=X+20.
        CALL ESUB
        CALL POINTR (4,1000)
        DO 2000 I=1,51
        CALL GET (4,X,Y)
        CALL CHANGE (4,X,Y/2.)
 2000   CALL ADVANC (4)
        STOP
        END
```

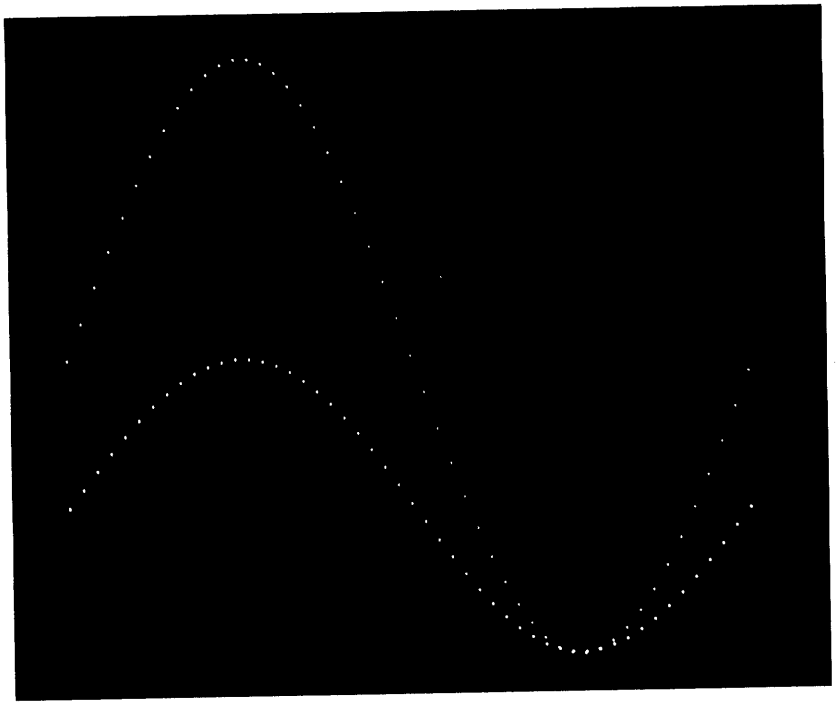

Figure 2-21 GET Subroutine

Routines called by GET: TOSAT (h-s) FRSAT (h-s)

Routines that call GET: CHANGA

2.6.4 CHANGE: Changing the Coordinates of a Primitive

Form: CALL CHANGE (k,x,y)

Arguments:

This subroutine operates on the primitive identified by Pointer k. The primitive's coordinates are changed to x and y.

Instructions:

This subroutine is usually called to change a primitive defined by the VECT, SVECT, LVECT, RPNT, APNT, or VIEWPT subroutines, or a primitive in a graph or figure subpicture created by XGRA, FIGR, or YGRA.

When CHANGE is used on a graph subpicture created by XGRA or YGRA, the y value is ignored in the XGRA call, and the x value is ignored in the YGRA call. Remember that the primitives in subpictures created by XGRA and YGRA are the individual x and y values, respectively. With POINTR, you can set Pointer k to the individual element of the XGRA or YGRA array that you want to change.

You can also use CHANGE to change the displacements of vectors in an FIGR subpicture. In this case, each pair of values in the FIGR array (see Section 2.5.3) is a primitive that Pointer k will identify. The x and y arguments in the CHANGE call will be the new displacements.

The descriptions of FIGR, CHANGA, POINTR, and GET have examples of CHANGE.

NOTE

Do not use CHANGE to update an NMBR subpicture. The subroutine CHANGT is useful with NMBR is some cases; see Section 3.1.4.

Routines called by CHANGE: TOSAT (h-s)

Routines that call CHANGE: CHANGA

2.6.5 CHANGA: Changing a Primitive and Adjusting the Next Primitive

Form: CALL CHANGA (k,x,y)

Arguments:

This subroutine operates on the primitive identified by Pointer k. The primitive's coordinates are changed to x and y.

Instructions:

CHANGA changes the coordinates of the primitive referenced by Pointer k to the new values given by x and y. Unlike the CHANGE subroutine (see Section 2.6.4), it also adjusts coordinates of the next primitive in the display file so that the rest of the primitives will be at the same absolute screen positions.

The value of k must be in the range 1 through 20.

This subroutine may be called to change a primitive defined by the VECT, SVECT, LVECT, RPNT, APNT, or VIEWPT subroutines, or a graph subpicture created by XGRA or YGRA. When a CHANGA call is specified for a graph created by XGRA or YGRA, the y value is ignored in the XGRA call, and the x value is ignored in the YGRA call. When CHANGA is used on an FIGR subpicture, x and y will be the new displacements for the vector identified by Pointer k.

NOTE

Do not use CHANGA to update the NMBR subpicture. The subroutine CHANGT is useful with NMBR in some cases; see Section 3.1.4.

Example:

This example draws a crossed box on the screen and then changes the endpoint of the next-to-last vector, first with the CHANGE subroutine and then with CHANGA. The difference is clear from Figure 2-22: the CHANGA subroutine will leave the endpoint of the last vector unaffected.

```
      COMMON/DFILE/IBUF(1000)
      REAL ENDPNT(16)
      DATA ENDPNT/0.,100.,100.,0.,0.,-100.,-100.,0.,100.,100.,
     X  -50.,50.,-50.,-50.,100.,-100./
      CALL INIT(1000)
      CALL APNT(512.,512.,,-4)
      CALL FIGR(ENDPNT,16,100)
      PAUSE 'READY FOR CHANGE? TYPE <RET>'
      CALL POINTR(20,100,7)
      CALL CHANGE(20,-400.,-500.)
      PAUSE 'FIGURE 2-22 (    ): CHANGE'
      CALL INIT
      CALL APNT(512.,512.,,-4)
      CALL FIGR(ENDPNT,16,100)
      PAUSE 'READY FOR CHANGA? TYPE <RET>'
      CALL POINTR(20,100,7)
      CALL CHANGA(20,-400.,-500.)
      PAUSE 'FIGURE 2-22 (    ): CHANGA'
      STOP
      END
```

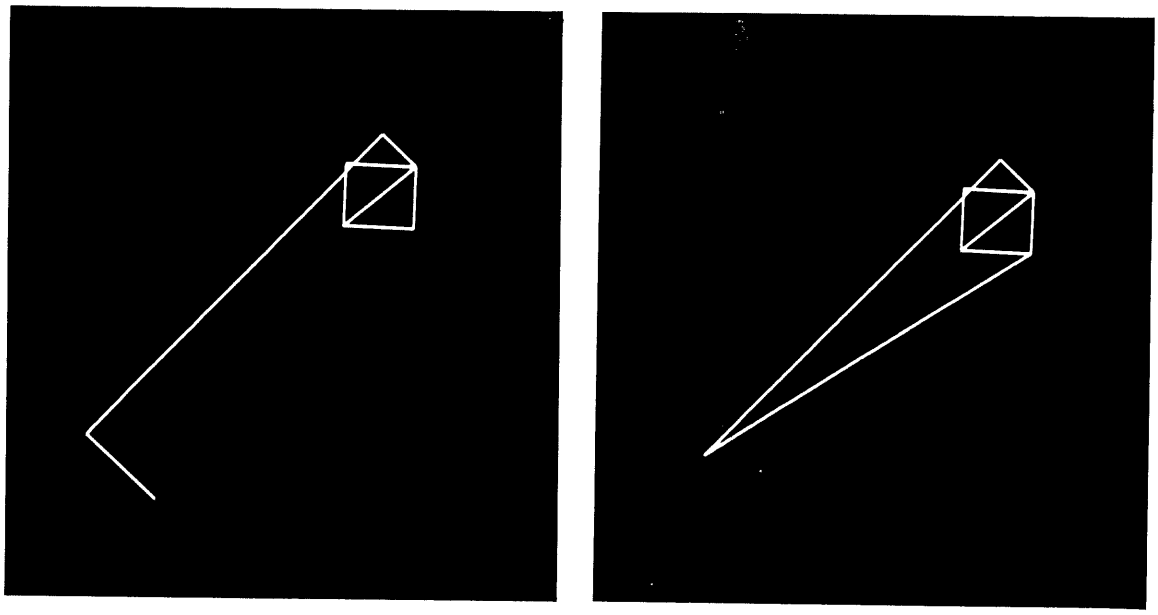

Figure 2-22 CHANGE and CHANGA Subroutines

Routines called by CHANGA: GET, ADVANC, CHANGE, TOSAT (h-s)

Routines that call CHANGA: None.

2.6.7 INSRT: Inserting Primitives in the Display File

Form: CALL INSRT [(k)]

Arguments:

The optional argument k is the pointer number (1 to 20).

Instructions:

INSRT allows you to insert primitives in an existing subpicture just before the position of the primitive referenced by Pointer k. In other words, you can add new primitives to the subpicture, or you can modify existing primitives by changing their relative positions in the subpicture (with relative vectors, for example).

When you include the k argument, an INSRT call effectively reopens the subpicture for input, so that primitives created by the graphic subroutine calls following INSRT will be entered in the subpicture definition.

When you call INSRT a second time and omit k, the insert operation is terminated, the subpicture is effectively closed, and any subsequently defined primitives will be inserted at the end of the display file in the ordinary way.

CAUTION

After you have called INSRT to reopen a subpicture, you should not call CMPRS, SAVE, RSTR, SUBP, SENSE, INTENS, FLASH, LINTYP, CVSCAL, or ESUB until you have called INSRT the second time (with no argument) to close the subpicture again. If you try to do so, the error message (Number 16) ILLEGAL DURING INSRT will appear on your terminal.

An example using INSRT is included in the description of the ERASP subroutine (see Section 2.6.8).

Routines called by INSRT: STOP, CONT, TOSAT (n-s)

Routines that call INSRT: CVSCAL, SENSE

2.6.8 ERASP: Erasing a Primitive

Form: CALL ERASP (k)

Arguments:

The optional argument k is a pointer number (1 to 20).

Instructions:

ERASP erases the primitive referenced by Pointer k. After erasing the referenced primitive, ERASP positions k at the next primitive in the file.

NOTE

Because primitive numbers are relative, erasing a primitive changes the numbers of all subsequent primitives in the subpicture. The numbers are reduced by 1.

Example:

This example waits for a light-pen hit on a subpicture and then erases the primitive identified by the hit. A vector is inserted at this point and attached to the tracking object. The program then waits for a carriage return from the keyboard before adjusting the vector. (For descriptions of LPEN and TRAK, see Section 2.8.)

```
        .
        .
        .
100   CALL LPEN (IH,IT,X,Y,IP)
      IF (IH.EQ.0) GO TO 100
      CALL POINTR (2,IT,IP)
      CALL ERASP (2)
      CALL INSRT (2)
      CALL LVECT (0.,0.)
      CALL INSRT
      CALL TRAK (X,Y)
      CALL POINTR (2,IT,IP)
      CALL ATTACH (2)
      WRITE (5,101)
101   FORMAT ('  REPOSITION TRACKING OBJECT,
     X TYPE <CR> WHEN DONE')
      READ (5,110) I
110   FORMAT (A1)
      CALL GRID (50.,50.)
        .
        .
        .
```

Routines called by ERASP: STOP, CONT, TOSAT (n-s)

Routines that call ERASP: None.

2.7 CHANGING DISPLAY PARAMETERS

The subroutines described in this section provide a second way of setting the display parameters l, i, f, and t. The first way is by including them as optional arguments in calls to the graphic subroutines APNT, AVECT, FIGR, LVECT, RPNT, SVECT, VECT, XGRA, and YGRA.

The second way to change the display parameters is with the subroutines SENSE, INTENS, FLASH, and LINTYP. Each of these subroutines changes the l, i, f, or t parameter associated with one of the graphic subroutines. The connection between these subroutines and the display parameters is as follows:

- light-pen sensitivity; l parameter; changed by SENSE
- intensity level; i parameter; changed by INTENS
- flash mode; f parameter; changed by FLASH
- line type; t parameter; changed by LINTYP

All four of these subroutines are used with pointers. That is, first use the POINTR subroutine to identify a single primitive in some subpicture, and then use one or more of the subroutines in this section to change that primitive's display parameter(s).

2.7.1 SENSE: Setting the Light-Pen Parameter

Form: CALL SENSE (k[,l,s])

Arguments:

The k argument is a pointer number (1 to 20). The l argument is the new light-pen-sensitivity parameter; if it is not included, the primitive identified by Pointer k becomes sensitive to the light pen.

If you have a VS60 with two display scopes, s is the number of the scope being addressed (either 1 or 2). If you have a one-scope VS60 or a VT11, the s argument will be ignored.

Instructions:

SENSE enables or disables light-pen sensitivity for the primitive identified by Pointer k.

Remember that when a primitive is sensitized to the light pen, all the primitives that follow it in the display file will also be sensitized, unless

1. The primitive is in a subpicture terminated by a restoring ESUB (see Section 2.4.2); or
2. Another call disables the sensitivity again, either a later call within the subpicture or a second call to SENSE with l=-1.

The s parameter is included in VS60 subsystems to indicate the scope to which the light-pen reference applies; legal values are 1 and 2. If the s parameter is omitted, Scope 1 is assumed.

NOTE

The first time you use SENSE within a subpicture to change the light-pen-sensitivity parameter (l), the display screen may become blank for a moment, and then reappear. A technique for avoiding this occurrence is given in Section 3.2.4.

Routines called by SENSE: INSRT, TOSAT (h-s)

Routines that call SENSE: None.

2.7.2 INTENS: Setting the Intensity Parameter

Form: CALL INTENS (k[,i])

Arguments:

The k argument is a pointer number (1 to 20). The optional i argument is a new intensity level; if i is omitted, the INTENS call will effectively erase the minus sign from the primitive's intensity level; the previously invisible primitive will appear on the screen with its new intensity level equal to the absolute value of its old level.

Instructions:

If i is in the range 1 through 8, the intensity level of the primitive is changed to the value of i; 1 is the faintest intensity and 8 is the brightest.

If i is 0, omitted from the call, or greater than 8, the primitive appears on the screen if it was formerly invisible, but the overall intensity level does not change.

If i is in the range -1 through -8, the primitive will be invisible, but the intensity level of subsequent primitives will be changed to the absolute value of i.

If i is less than -8, the primitive will be invisible, without any change in the overall intensity level.

When you set the intensity level for a certain primitive to a value from 1 to 8 or from -1 to -8, you are also setting the intensity level for all the primitives that follow. The new intensity level will remain in effect unless

1. The referenced primitive is part of a subpicture that ends with a restoring ESUB (see Section 2.4.2) -- a VS60-only feature; or
2. A later call to INTENS or to one of the graphic subroutines changes the intensity level again.

NOTE

The first time you use INTENS within a subpicture to change the intensity parameter (i), the display screen may become blank for a moment, and then reappear. A technique for avoiding this occurrence is given in Section 3.2.4.

Example:

This example (see Figure 2-24) displays three light buttons in the menu area. It then waits for a light-pen hit on one of the light buttons. When the light pen is pointed at one of the words on the screen, the word's intensity is increased. If the example is run on a VS60, the program will also wait for a tip-switch hit (the ITIP argument) as confirmation of the menu selection.

```
      COMMON/DFILE/IBUF(1000)
      CALL INIT(1000)
      CALL MENU(,512.,-100.,56,'ERASE','COPY','MOVE')
600   CALL LPEN(IH,IT,,,,,,ITIP)
      IF(IH.EQ.0.OR.IT.GT.58.OR.ITIP.EQ.0) GO TO 600
      CALL POINTR(7,IT)
      CALL INTENS(7,8)
      STOP
      END
```

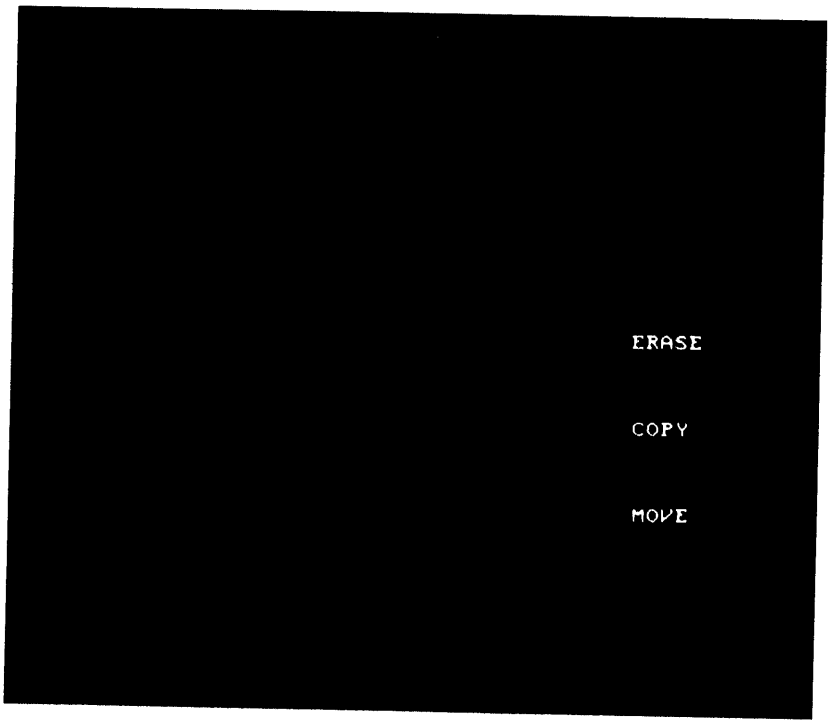

Figure 2-24 INTENS Subroutine

Routines called by INTENS: TOSAT (h-s)

Routines that call INTENS: None.

2.7.3 FLASH: Setting the Flash-Mode Parameter

Form: CALL FLASH (k[,f])

Arguments:

The k argument is a pointer number from 1 to 20. The optional f argument is a new value for the flash-mode display parameter. If you omit f, the primitive identified by Pointer k will flash.

Instructions:

FLASH enables or disables flash mode for the primitive identified by Pointer k.

If the f argument in the FLASH call is positive, of if f is omitted from the call, flash mode is enabled. The referenced primitive will flash on and off. In addition, all subsequent primitives will flash unless

1. The referenced primitive is part of a subpicture that ends with a restoring ESUB (see Section 2.4.2) -- a VS60-only feature; or

2. A later call to FLASH or to one of the graphic subroutines changes the f parameter again.

If the f parameter is negative, flash mode is disabled for the primitive referenced by k and for subsequent primitives.

If f is 0, the FLASH call has no effect.

An example using FLASH is included in the description of LPEN (see Section 2.8.1).

NOTE

The first time you use FLASH within a subpicture to change the flash-mode parameter (f), the display screen may become blank for a moment, and then reappear. A technique for avoiding this occurrence is given in Section 3.2.4.

Routines called by FLASH: TOSAT (h-s)

Routines that call FLASH: None.

2.7.4 LINTYP: Setting the Line-Type Parameter

Form: CALL LINTYP (k[,t])

Arguments:

The k argument is a pointer number from 1 to 20. The optional t argument is a new value for the line-type display parameter. If you omit t, the LINTYP call has no effect.

Instructions:

LINTYP allows you to specify the line type of the vector identified by Pointer k.

If the t argument in the LINTYP call is in the range 1 through 4, the line type is changed for the referenced vector. The new line type will also apply to all subsequent vectors unless

1. The vector is part of a subpicture that ends with a restoring ESUB (see Section 2.4.2) -- a VS60-only feature; or
2. A later call to LINTYP or to one of the graphic subroutines changes the line type again.

Vectors are drawn according to the following conventions:

t Parameter	Vector Type
1	Solid line
2	Long-dash line
3	Short-dash line
4	Dot-dash line

If t is 0, omitted from the call, negative, or greater than 4, the current line type remains unchanged.

NOTE

The first time you use LINTYP within a subpicture to change the line-type parameter (t), the display screen may become blank for a moment, and then reappear. A technique for avoiding this occurrence is given in Section 3.2.4.

Routines called by LINTYP: TOSAT (h-s)

Routines that call LINTYP: None.

2.8 INTERACTING WITH THE DISLAY

This section describes several subroutines that you can use to write interactive programs.

To write interactive programs in DECgraphic-11 FORTRAN, use the following procedure:

1. Choose a primitive or primitives that will be sensitive to the light pen; or associate any of the 16 buttons on the optional LK-11 pushbutton box with a graphic function; or associate particular keyboard characters with a set of graphic functions.

2. Use one of several subroutines (LPEN, PBS, PBH, KBC, or GRATTN) to "poll" the interactive devices; the subroutine will tell you when a graphic attention, or "hit," has occurred and will also return information about the source of the hit (for example, the subpicture in which the hit occurred). Except for GRATTN, each of the polling subroutines polls a single device. GRATTN can optionally poll all three devices and can suspend the program until a hit occurs. GRATTN should usually be followed by a call to the subroutine that is specific to the given device (e.g., PBH or PBS for the pushbutton box).

3. Then use the information returned by your polling call to branch to the appropriate section of the program; in that section you can change a display parameter, insert a new primitive, or make whatever change is necessary.

The description of each subroutine explains its use in detail.

This section also describes two subroutines, TRAK and TRAKXY, that are used with the tracking object. TRAK will display the tracking object on the screen, and TRAKXY returns the coordinates of the tracking object. The tracking object will center itself on the light pen when the light pen touches it. You can also attach certain primitives, usually vectors, to the tracking object so that they will "follow" it on the screen as you move the light pen. Use the subroutines ATTACH and DETACH to move primitives in this way.

Finally, the subroutine GRID described in this section creates an invisible dot pattern on the screen, each dot having a known coordinate position. As a final step in programs that use the light pen and tracking object, the GRID subroutine will move the tracking object (and any attached primitives) to the nearest point on the invisible grid. GRID is therefore useful when you are making rectilinear drawings, since it tends to compensate for the minor errors you may make in moving objects to an exact position with the light pen.

2.8.1 LPEN: Recording a Light-Pen Hit

Form: CALL LPEN (IH,IT[,X,Y,IP,IA,IM,IT1,IT2])

Arguments:

The IH and IT arguments must be included. If the value returned by IH is 0, no light-pen hit occurred since the last LPEN call. If the value is 1, a hit occurred on Scope 1. If the value is 2, a hit occurred on Scope 2. (On one-scope subsystems, a hit is indicated by IH=1.) If a hit did occur, IT returns the tag of the subpicture in which the hit occurred.

The optional arguments return more detailed information: the coordinates of the hit (X and Y), the primitive number that was hit (IP), the precedents of a nested subpicture (array IA), the screen area of the hit (IM=1: main viewing area; IM=2: menu area), and for VS60 scopes, the status of the light-pen tip switches for Scope 1 and Scope 2 (ITn=0: off; ITn=1: on).

Instructions:

LPEN tests, or "polls," for a light-pen hit on the display screen. You can test for a hit on any of the primitives that have been made sensitive to the light pen (e.g., with the SENSE subroutine; see Section 2.7.1). If you point the light pen at a primitive that is not sensitive, nothing will happen; if the primitive is sensitive, a graphic attention, or light-pen hit, will occur.

LPEN does not suspend the program, or "wait," until a light-pen hit occurs. If you want to continually poll a particular primitive, for example, put the LPEN call in a programmed loop. See also the description of GRATTN (Section 2.9.1). GRATTN can make the program wait until a light-pen hit occurs; then you can follow immediately with an LPEN call to get all the details on the hit.

One of the noticeable properties of the light-pen device and the LPEN subroutine is the extreme rapidity with which they generate and detect light-pen attentions. Consider the following sample program segment;

```
      COMMON/DFILE/I(500)
      CALL INIT(500)
      CALL SUBP(2)
      CALL APNT(512.,512.,1,-4)
      CALL VECT (0.,100.,,1)
      CALL RPNT(40.,0.,,-4)
      CALL VECT (0.,100.,,2)
      CALL RPNT(60.,0.,,-4)
      CALL VECT(0.,100.,,4)
      CALL RPNT(80.,0.,,-4)
      CALL VECT(0.,100.,,6)
      CALL RPNT(100.,0.,,-4)
      CALL VECT(0.,100.,,8)
100   CALL LPEN(IH,IT,X,Y,IP,,IM)
      IF(IH.EQ.0) GO TO 100
      IF(IP.EQ.10) GO TO 200
      WRITE(5,110) IT,X,Y,IP,IM
      GO TO 100
200   WRITE(5,120)
110   FORMAT(' ','HIT ON SUBPICTURE ',I2,',
     X   COORDINATES ',F7.2.','
     X   ,F7.2,', PRIMITIVE ',I2,'; IM=',I2)
120   FORMAT(' HIT ON PRIMITIVE 10; STOP')
      STOP
      END
```

Write up, compile, link, and run this short program. It will display five vertical vectors on the screen, with intensities 1, 2, 4, 6, and 8. Since the APNT call sets l=1, all the primitives in the subpicture are light-pen sensitive.

NOTE

If you run this program, adjust the BRIGHTNESS (or INTENSITY) knob on your scope so that the first vector, at Intensity Level 1, is barely visible.

First of all, you will notice that nothing happens when you point the light pen at the first two vectors; they are too faint to be detected if your scope is properly adjusted.

However, the three brightest vectors are bright enough, and an attention will occur when you touch them with the light pen. Note that LPEN is in a loop that will continue until you hit Primitive 10, the brightest vector.

If you touch the other bright vectors with the light pen, you will notice that unless you sweep the pen across the vector very quickly, LPEN will record more than one hit. When you touch a sensitive primitive with the light pen, several attentions are often generated before you can move the pen again.

If the generation of these multiple attentions creates a problem for your particular application, one solution is to make LPEN count the number of hits on a particular primitive:

```
                .
                .
                .
        IHITS=0
100     CALL LPEN(IH,IT,IP)
        IF(IH.EQ.0.OR.IT.NE.2.OR.IP.NE.6) GO TO 100
        IHITS=IHITS+1
        IF(IHITS.LT.10) GO TO 100
                .
                .
                .
```

This LPEN segment will make the program pause until it gets 10 hits on Primitive 6 of Subpicture 2. It will ignore 'accidental' hits on other primitives, and it will, in essence, demand that you deliberately select Primitive 6. If you replace the LPEN section of the previous sample program with this fragment, you will notice that the message printed on the terminal will be better synchronized with the act of touching the primitive. In other words, the illusion of "real-time" interaction is more convincing.

The dummy variable IA must be dimensioned to 8 in a DIMENSION or INTEGER statement when you use it. This variable tells whether a particular primitive is in a nested subpicture. If the primitive's subpicture, IT, is nested, IA will contain the tags of the "precedents" of IT, or in other words, the tags of the subpictures within which IT is nested. IA(1) will contain the tag of Precedent 1, the innermost subpicture. Since a subpicture cannot have more than seven precedents, the last element of IA will always equal -2. If there are fewer than seven precedents, the element of IA following the last precedent tag will equal -2. For instance, if there are three

precedents, IA(4)=-2. If Subpicture IT is not a nested subpicture (no precedents), IA(1)=-2.

Example:

This example displays a menu of program options. It then waits for a light-pen hit in the menu area, flashes the option selected by the hit, and branches to the appropriate section of the program.

```
        .
        .
        .
     CALL MENU (,800.,-50.,100,
    X 'SAMPLE','FFT','PLOT','STOP')
     CALL POINTR(20,100)
     CALL SENSE(20)
        .
        .
        .
100  CALL LPEN (IH,IT)
     IF (IH.EQ.0.OR.IT.LT.100.OR.IT.GT.103) GO TO 100
     CALL POINTR (11,IT)
     CALL FLASH (11)
     GO TO (100,200,300,400), IT-99
        .
        .
        .
```

Routines called by LPEN: POINTR, ADVANC, TOSAT (h-s), FRSAT (h-s)

Routines that call LPEN: None.

2.8.2 TRAK: Placing a Tracking Object on the Screen

Form: CALL TRAK (x,y[,s])

Arguments:

You must supply the x and y arguments; they give the coordinate position at which the tracking object will appear.

The optional s argument (1 or 2) is for two-scope VS60 subsystems and identifies the scope on which the tracking object will appear. The s argument is ignored on VT11 subsystems and on one-scope VS60 subsystems. If you omit s on a two-scope subsystem, the tracking object will appear on Scope 2.

Instructions:

TRAK allows you to position an octagonal tracking object on the display screen. The tracking object is positioned initially at the coordinate position (x,y). When you touch the tracking object with the light pen, it will center itself on the light-pen coordinates, and it will then "follow" the light pen as you move it across the screen. When the tracking object moves, so will any primitives that are attached to it (see ATTACH, Section 2.8.4).

The tracking object cannot be positioned in the menu area of the VS60 screen. Note also that the tracking object is not moved by either a call to CVSCAL or a call to VIEWPT.

NOTE

On VS60 scopes, the tracking object will follow the light pen only when the light-pen tip switch is "on" (pressed against the surface of the display screen).

You can remove the tracking object from the screen by calling ERAS with no argument (see Section 2.4.6).

An example using TRAK is included in the description of GRID (see Section 2.8.6).

Routines called by TRAK: TOSAT (h-s)

Routines that call TRAK: GRID

2.8.3 TRAKXY: Returning the Coordinates of the Tracking Object

Form: CALL TRAKXY (X,Y[,s])

Arguments:

The REAL dummy variables X and Y return the current coordinates of the tracking object. The optional argument s indicates, for two-scope VS60 subsystems, whether the tracking object being addressed is on Scope 1 (s=1) or Scope 2 (s=2). If you omit s, Scope 1 is implied on a two-scope subsystem. On one-scope VS60 subsystems and VT11 subsystems, s is ignored.

Instructions:

This example (see Figure 2-25) shows TRAKXY being used to retrieve the starting coordinates of a figure subpicture. The coordinates are displayed on the screen by the TEXT and CHANGT subroutines, in "odometer" fashion. This example uses a special subroutine, ITOA, to convert integers to ASCII codes. See Section 3.1.4 for a description of ITOA.

```
      COMMON/DFILE/IBUF(1000)
      REAL ENDPNT(16)
C
C     LOGICAL ARRAYS FOR COORDINATE
C     CHARACTER STRINGS
C
      LOGICAL*1 COORDX(8),COORDY(8)
      DATA ENDPNT/0.,100.,100.,0.,0.,-100.,-100.,0.,100.,100.,
     X -50.,50.,-50.,-50.,100.,-100./
      CALL INIT(1000)
C
C     DISPLAY BOX FIGURE (SUBP. 101)
C
      CALL SUBP(101)
      CALL APNT(512.,512.,,-4)
      CALL FIGR(ENDPNT,16,100)
      CALL TRAK(512.,512.)
      CALL ESUB(101)
C
C     ATTACH FIGURE TO TRACKING OBJECT
C
      CALL POINTR(15,101)
      CALL ATTACH(15)
C
C     DISPLAY COORDINATE STRINGS (SUBP. 111)
C
      CALL APNT(0.,100.,,-4)
      CALL SUBP(111)
      CALL TEXT('MOVE THE TRACKING
     X OBJECT WITH THE LIGHT PEN')
      CALL TEXT(1,-2,'NEW COORDINATES:',' X=')
      CALL TEXT('00000')
      CALL TEXT(', Y=')
      CALL TEXT('00000')
      CALL ESUB(111)
C
C     SET POINTERS TO X AND Y COORDINATE STRINGS
C
      CALL POINTR(1,111,3)
      CALL POINTR(2,111,5)
C
C     GET TRACKING OBJECT'S COORDINATES,
```

```
C       CONVERT THEM TO INTEGERS (IXCORD, IYCORD),
C       AND CONVERT INTEGERS TO ASCII (ITOA)
C
C       COORDINATE DISPLAY IS UPDATED IN LOOP 500
C
500     CALL TRAKXY(XCORD,YCORD)
        IXCORD=INT(XCORD)
        IYCORD=INT(YCORD)
C       [STOP PROGRAM WHEN X COORDINATE < 35]
        IF(IXCORD.LT.35) GO TO 501
        CALL ITOA(IXCORD,5,COORDX)
        CALL ITOA(IYCORD,5,COORDY)
C
C       APPEND NULL BYTE TO END OF
C       5-CHARACTER COORDINATE STRINGS
C
        COORDY(6)=0
        COORDX(6)=0
C
C       UPDATE COORDINATE STRINGS
C
        CALL CHANGT(1,COORDX)
        CALL CHANGT(2,COORDY)
        GO TO 500
501     STOP
        END
```

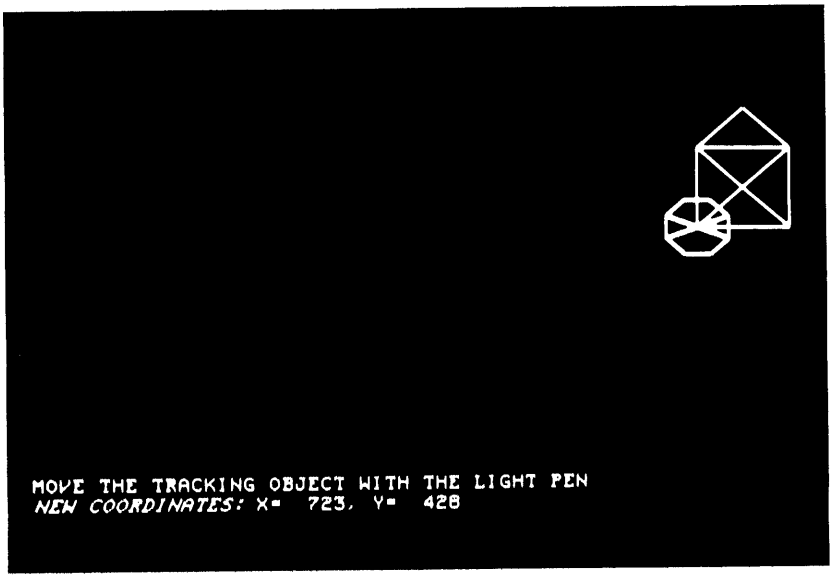

Figure 2-25 TRAKXY Subroutine

Another example using TRAKXY is included in the description of GRID (see Section 2.8.6).

Routines called by TRAKXY: TOSAT (h-s), FRSAT (h-s)

Routines that call TRAKXY: GRID

2.8.4 ATTACH: Attaching a Primitive to the Tracking Object

Form: CALL ATTACH (k[,n,s])

Arguments:

The k argument, which you must supply, is a pointer number from 1 to 20. The optional argument n indicates which end of an attached vector will move with the tracking object. If you omit n, the vector's origin is stationary and its destination point moves.

The s argument indicates which scope of a two-scope VS60 subsystem is being addressed; making s=1 attaches primitives to the tracking object on Scope 1; s=2 attaches the primitives on Scope 2 to that scope's tracking object; if you omit s, Scope 1 is implied. The s argument is ignored on one-scope VS60 subsystems and on VT11 subsystems.

Instructions:

ATTACH allows you to attach the primitive identified by Pointer k to the tracking object. The primitive will not necessarily be superimposed on the tracking object, but it will still move the same distance in the same direction as the tracking object.

Primitives that can be attached in this fashion include the long vector (LVECT), the absolute point (APNT), and the absolute vector (AVECT). If you specify a pointer to any other kind of primitive, the ATTACH call will be ignored.

By attaching a primitive to the tracking object, you can easily change the coordinates of a point or vector by moving it to another part of the display screen and then detaching it (see the description of DETACH in the next section).

If the primitive to be attached is a long vector, the vector will follow the tracking object as it moves on the display screen. The optional n argument determines which end of the vector will move and which end will be stationary. If n is positive or omitted from the call, the vector's origin is stationary and its destination will move. If n is negative, the vector's destination is stationary and its origin will move.

NOTE

If you try to use ATTACH to "stretch" a vector more than 1023 raster units, the vector will not move along reliably with the tracking object.

You do not need the n argument if the attached object is an absolute point or an absolute vector. If an absolute point or vector is attached to the tracking object, the point or vector end point will be moved the same distance as the tracking object.

You can attach as many as 40 primitives to the tracking object on the display screen by simply calling ATTACH repeatedly.

An example using ATTACH is included in the description of GRID (see Section 2.8.6).

Routines called by ATTACH: TOSAT (h-s)

Routines that call ATTACH: None.

2.8.5 DETACH: Detaching Primitives from the Tracking Object

Form: CALL DETACH [(s)]

Arguments:

The optional s argument indicates which scope of a two-scope VS60 subsystem is being addressed (s=1: Scope 1; s=2: Scope 2). If you omit s, Scope 1 is implied. The s argument is ignored on one-scope VS60 subsystems and VT11 subsystems.

Instructions:

DETACH detaches all currently attached primitives from the tracking object.

Routines called by DETACH: TOSAT (h-s)

Routines that call DETACH: None.

2.8.6 GRID: Positioning the Tracking Object on the Grid

Form: CALL GRID (gx,gy[,s])

Arguments:

The arguments gx and gy give the spacing between the individual points of the grid.

The optional s argument identifies the scope (1 or 2) to which the GRID call applies. If you omit s, Scope 1 is implied. The s argument is ignored on one-scope VS60 subsystems and on VT11 subsystems.

Instructions:

GRID defines on the display screen the coordinates of a grid of evenly spaced invisible dots, and moves the tracking object to the nearest point on the grid. GRID is useful when you are drawing with the light pen or positioning objects on the screen.

When GRID moves the tracking object on the screen, it also makes necessary adjustments to the coordinate positions of any points and vectors that are attached to the object. An automatic DETACH is performed after the adjustment.

Example:

This example uses input from the light pen to build a subpicture consisting of an absolute point and a long vector. The starting of the line is initially positioned by moving the tracking object and typing a carriage return on the terminal when finished. The GRID subroutine is used to force the starting point to lie on the invisible grid whose (x,y) spacing is (50.,50.). A long vector is then drawn and attached to the tracking object, which can be moved with the light pen. The vector is then effectively "stretched" with the tracking object to the desired endpoint. GRID is once again called to "normalize" the endpoint by making it lie exactly on the grid.

```
          .
          .
          .
      CALL TRAK (500.,500.)
      WRITE (5,10)
10    FORMAT (' POSITION TRACKING OBJECT, TYPE <CR> WHEN DONE')
      READ (5,20) I
20    FORMAT (A2)
      CALL GRID (50.,50.)
      CALL TRAKXY (X0,Y0)
      CALL SUBP (1500)
      CALL APNT (X0,Y0,,-4)
      CALL LVECT (0.,0.)
      CALL ESUB (1500)
      CALL POINTR (20,1500,2)
      CALL ATTACH (20)
      WRITE (5,30)
30    FORMAT (' DRAW LINE, TYPE <CR> WHEN DONE')
      READ (5,20) I
      CALL GRID (50.,50.)
          .
          .
          .
```

Routines called by GRID: TRAK, TRAKXY, TOSAT (h-s)

Routines that call GRID: None.

2.9 POLLING INTERACTIVE DEVICES

As you have seen with the light-pen subroutines (Section 2.8), a graphic program can be made interactive by the use of graphic attentions created when the light pen detects a primitive on the screen. The DECgraphic-11 FORTRAN Graphics Package also allows you to use a programmable pushbutton box (Section 2.10) and the keyboard (Section 2.11) as interactive devices. Each of these three devices has its own set of subroutines for detecting graphic attentions, such as the LPEN subroutine for the light pen and the PBH subroutine for the pushbutton box.

Because there are three devices that can be used for interactive graphics, you can write programs that make use of graphic attentions from all three, thereby extending the control you have over the display. In a program that uses more than one interactive device, it is convenient to have a simpler way of processing graphic attentions than is provided by the individual subroutines like LPEN and PBH. That is, you need a subroutine that can

1. Wait for a graphic attention to occur;
2. Tell you when a graphic attention does occur;
3. Tell you the source of the graphic attention; and
4. Tell the program where to look in its code for an attention-processing routine, once it knows the source of the graphic attention.

The subroutine GRATTN fulfills these requirements. It is described in Section 2.9.1.

Remember that when using GRATTN or other attention-polling subroutines, for all practical purposes FORTRAN is a synchronous language. Synchronous, in this case, means that your program can look for or respond to graphic attentions only at the precise moment that you tell it to do so. For instance, the example in Section 2.8.1 for LPEN shows that this subroutine is usually placed in a loop that repeatedly looks for light-pen hits (i.e., graphic attentions):

```
100 CALL LPEN(IH,IT)
    IF(IH.EQ.0.OR.IT.LT.100.OR.IT.GT.103) GO TO 100
```

The program will stay in this loop until it gets a light-pen hit from subpictures 100 to 103. Once it does get outside the loop, however, the program cannot detect light-pen hits any more. That is what is meant by synchronous; if FORTRAN were asynchronous, a subroutine like LPEN could respond to a light-pen hit no matter when it occurred.

2.9.1 GRATTN: Graphic-Attention Handling

Form: CALL GRATTN(iwait,IRETRN,idev1[,idev2,idev3,...,idevn])

Arguments:

The iwait argument tells GRATTN either to return after one poll of the interactive devices (iwait=0) or to wait until a graphic attention occurs (iwait=1).

The IRETRN argument is the "hit indicator." It will be 0 when GRATTN returns if there was no graphic attention. If a graphic attention occurred, IRETRN will be an index to the list of device numbers (idevn). That is, IRETRN will be either 1, 2, or 3, if three device numbers are supplied.

The idev1-idevn arguments list device numbers that stand for the interactive devices in your display subsystem. At present, the DECgraphic-11 FORTRAN Graphics Package recognizes three interactive devices, with the following device numbers:

1 = Light pen

2 = Pushbutton box

3 = Keyboard

Instructions:

The three device numbers can be listed in any order; IRETRN will point to the first, second, or third position in the list of device numbers. Notice also that only one device number is required, so that GRATTN can poll a single device instead of several.

For example, the LPEN subroutine always returns after a single check for a light-pen attention with either a "hit" or a "no hit" indication. That is why LPEN is usually put in a programmed loop. As an alternative, consider the following call to GRATTN:

```
CALL GRATTN(1,IRETRN,1)
```

When you make this call, the program will wait until a light-pen hit occurs. You then can follow up with a call to LPEN to get detailed information about the light-pen hit, as described in Section 2.8. To poll the light pen repeatedly, you can put the GRATTN call in a loop, in which case the program will wait for a light-pen hit on each pass through the loop.

CAUTION

Each time GRATTN is called, LPEN, PBH, PBS, or KBC must be called immediately afterward to clear the ATTENTION flag. Otherwise, repeated GRATTN calls (in a DO loop, for instance) will keep reporting the same attention that set the ATTENTION flag in the first place.

Because the device numbers can appear in any order, GRATTN can also check only the pushbutton box,

```
CALL GRATTN(1,IRETRN,2)
```

the pushbutton box and keyboard,

```
CALL GRATTN(1,IRETRN,2,3)
```

or any other combination of the three devices.

When you use GRATTN to poll several devices, IRETRN can be used in a "computed GO TO" statement to branch to the proper attention-processing routine. For example,

```
CALL GRATTN(1,IRETRN,3,2,1)
GO TO (100,200,300), IRETRN
```

In this example, IRETRN will equal 1 if Device 3 (the keyboard) created the graphic attention, 2 for Device 2 (the pushbutton box), and 3 for Device 1 (the light pen).

Because IRETRN is an index to the list of device numbers, rather than the device number itself, the list of device numbers can be expanded when new interactive devices are added in the future.

Routines called by GRATTN: TOSAT (h-s), FRSAT (h-s)

Routines that call GRATTN: None.

2.10 USING THE OPTIONAL PUSHBUTTON BOX

One of the optional features offered by the DECgraphic-11 FORTRAN Graphics Package is support for the LK-11 pushbutton box.

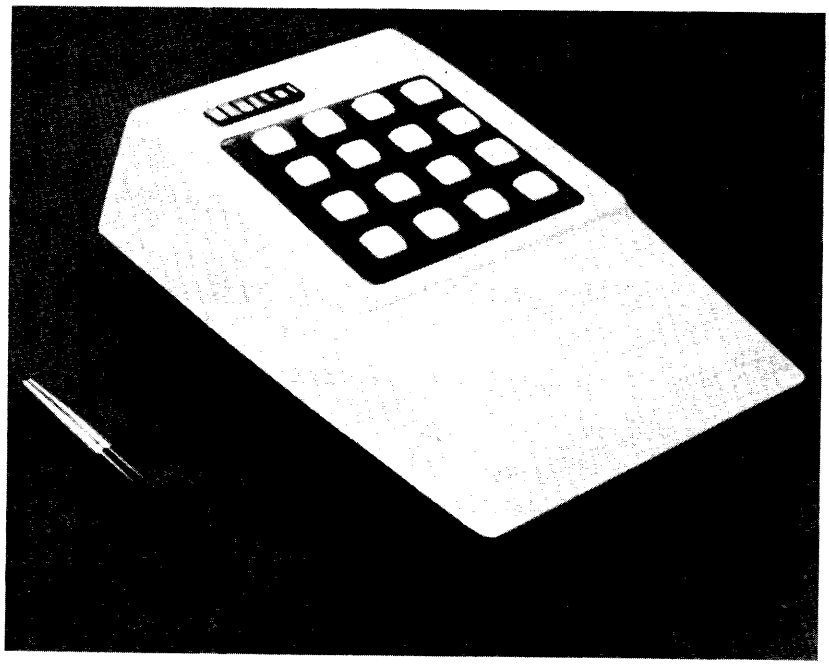

Figure 2-26 LK-11 Pushbutton Box

The LK-11 has 16 buttons, each with its own programmable light. Each button has two states, represented by the logical values .TRUE. and .FALSE. Normally, the buttons will light up when they are .TRUE., although you can also tell the program to light up buttons directly.

A button generates a graphic attention whenever it is pushed (that is, changed from .TRUE. to .FALSE., or from .FALSE. to .TRUE.).

The DECgraphic-11 FORTRAN Graphics Package controls the pushbuttons in three ways:

1. Records the current .TRUE./.FALSE. status of the buttons (PBS);

2. Looks for a pushbutton "hit" (PBH); and

3. Turns specific lights on or off (PBL).

As was explained in Section 2.9, the subroutine GRATTN can also look for pushbutton hits. Subroutine PBH is to the pushbuttons what LPEN is to the light pen; that is, PBH makes one check for a hit and returns. GRATTN can wait for a pushbutton hit without having to be placed in a loop, and can also poll other devices at the same time. Each time you use GRATTN to check for a hit, you must follow it with an attention-processing call -- PBH or PBS in the case of pushbuttons.

2.10.1 PBS: Checking the Status of the Pushbuttons

Form: CALL PBS(STATE)

Argument:

STATE is a LOGICAL*1 array of 16 elements, one for each button.

Instructions:

When you call PBS, the subroutine will return with the elements of STATE set to either .TRUE. or .FALSE.

PBS does not change the status of the buttons, but only records them in STATE.

In normal operations, a light on the LK-11 turns on when the state of the button is .TRUE. and off when the state is .FALSE. However, if you have used subroutine PBL (see Section 2.10.3) to turn certain lights on or off, the lights will no longer reflect the .TRUE./.FALSE. status of the buttons.

Routines called by PBS: TOSAT (h-s), FRSAT (h-s)

Routines that call PBS: None.

2.10.2 PBH: Checking for a Pushbutton Hit

Form: CALL PBH(IHIT,PUSHED)

Arguments:

IHIT will be 0 if no buttons have been pushed since the last call to PBH. IHIT will be a nonzero integer if any of the 16 buttons has been pushed since the last call. IHIT is therefore a "hit indicator" for the pushbutton box.

PUSHED is a LOGICAL*1 array of 16 elements. An element of PUSHED will be set to .TRUE. if the corresponding button was pushed.

Instructions:

The usual procedure for polling the pushbutton box is to put PBH in a programmed loop to wait for a hit and then to use elements of PUSHED in a logical IF statement. For example,

```
100   CALL PBH(IHIT,PUSHED)
      IF(IHIT.EQ.0) GO TO 100
      DO 110 I=1,16
      IF (PUSHED(I)) GO TO (120,130,140,150,160,170,
     X 180,190,200,210,220,230,240,250,260,270),I
110   CONTINUE
```

Notice that there is no connection between a .TRUE. value for an element of PUSHED and the same element of the PBS argument STATE. In other words, pushing a button may set STATE(1) to .FALSE. even though the same action will always set PUSHED(1) to .TRUE. PBH records whether a button has been pushed, without regard to its state.

Because there are two separate subroutines, PBS and PBH, that return logical information about the pushbuttons, you have two dimensions of interaction with the pushbutton box.

Routines called by PBH: TOSAT (h-s), FRSAT (h-s)

Routines that call PBH: None.

2.10.3 PBL: Generating the Pushbutton Lights

Form: CALL PBL[(DARK,LIT)]

Arguments:

DARK and LIT are both LOGICAL*1 arrays of 16 elements, one for each button. To turn off a button, set its element of DARK to .TRUE. before calling PBL. To light up a button, set its element of LIT to .TRUE. The DARK elements are processed first, so that setting the same element of both arrays to .TRUE. will light up the button.

No argument: Lights are returned to automatic control.

Instructions:

As noted in Section 2.10.1, buttons on the LK-11 are normally lit when their status (that is, their elements in STATE) is .TRUE. However, PBL disables this feature; PBS will still report the .TRUE./.FALSE. status of the buttons, but it will not be the same as the status of the lights.

To return the status of the lights to "automatic control," where a .TRUE. in the STATE (PBS) array represents a lit button, call PBL again with no arguments.

Routines called by PBL: TOSAT (h-s)

Routines that call PBL: None.

2.11 CONTROLLING THE KEYBOARD

The DECgraphic-11 FORTRAN Graphics Package includes three "keyboard control" subroutines, described in this section. The function of these subroutines is to read and write single characters or character strings from and to the keyboard you are using. The description of each routine shows that they offer advantages for graphic programming that do not exist with the usual FORTRAN READ and WRITE statements. In fact, READ and WRITE are among the statements that cannot be used at all in a satellite control program, and so these new subroutines are a necessity in that situation (see Section 5.3.4 if you are programming a host-satellite system).

One of these subroutines, KBC, allows you to use the keyboard as an interactive device. Used in this way, KBC is similar to LPEN (see Section 2.8); both subroutines are designed to return immediately when called, telling you whether a graphic attention came from the device since the last call. Therefore, KBC can be placed in a loop so that it will continually check for a "keyboard hit." As an alternative, you can use GRATTN (see Section 2.9.1) to make the program wait until a graphic attention occurs from the keyboard.

2.11.1 KBC: Reading a Character from the Keyboard

Form: CALL KBC(ICHAR)

Arguments:

ICHAR is an integer that returns the ASCII value of the character that was typed since the last call to KBC. If no character was typed, ICHAR is 0.

Instructions:

You can use KBC to return a character for text processing, or you can process the ASCII code in ICHAR and use the keyboard as an interactive device. For example, the letters A-P can be programmed to represent the same conditions as the 16 buttons on the LK-11 pushbutton box.

NOTE

In RT-11, KBC cannot read the character you type unless you precede it and follow it with a carriage return. In the other operating systems, the carriage returns are not required. If you want to activate MCR when KBC is in use in the stand-alone version of RSX-11M, you may have to type CTRL/C twice.

Routines called by KBC: TOSAT (h-s), FRSAT (h-s)

Routines that call KBC: None.

2.11.2 KBS: Reading a String from the Keyboard

Form: CALL KBS(length,STRING[,NUMBER])

Arguments:

The length argument has a maximum value of 72. STRING is a LOGICAL*1 array of maximum length equal to the length argument; it holds the ASCII values of characters read by KBS.

NUMBER is an optional argument that returns the actual number of characters read by KBS.

Instructions:

When you call KBS, the program will wait for a carriage return and then will read a string from the keyboard. (You can use the DELETE and CTRL/U keys to cancel characters and lines before typing the carriage return.) The maximum number of characters in the string (length) is 72; if you try to make KBS read an 80-character string, for example, it will ignore the last 8 characters. The maximum length does not include the NUL character (ASCII code 0) used for a terminator of FORTRAN strings.

Routines called by KBS: None.

Routines that call KBS: None.

2.11.3 TTW: Displaying Strings on the User's Terminal

Form: CALL TTW(number,ICHARS[,number,ICHARS,...])

Arguments:

The number argument tells TTW how many characters to display from the immediately following array, ICHARS.

ICHARS is a LOGICAL*1 array containing the characters to be displayed.

Instructions:

TTW sends character strings to the terminal, so that they are displayed on the screen (or on the hard copy, in the case of printer-type terminals).

If number is 0, you can use a FORTRAN literal (a character string enclosed in single quotation marks) for ICHARS. If number is -1, the characters in ICHAR are displayed as usual, but are not terminated with a carriage return; the cursor will remain at the end of the displayed text string instead of moving to the next line. This mode is useful for interactive displays in which the user must respond to a programmed question.

Notice that number and ICHARS can be repeated indefinitely, either to display several lines of the same string or to display different strings in the same call.

Routines called by TTW: None.

Routines that call TTW: None.

2.12 CONTROLLING THE OVERALL DISPLAY

Section 3.1.2 of this manual describes the use of the SUBP and OFF subroutines to create complete display files before any graphic information is actually displayed on the screen. Creating display files in this manner will speed up considerably the operation of a graphic program.

This section describes another way of creating display files quickly. If you are using the DECgraphic-11 FORTRAN Graphics Package in a stand-alone system (that is, with your display processor on the UNIBUS), there is only a small gain in speed with the technique discussed in this section versus the use of SUBP and OFF. The advantage for a stand-alone user is that the technique described in this section does not involve the creation of unnecessary subpictures, and that it uses a single, easily recognized subroutine.

If you are using the DECgraphic-11 FORTRAN Graphics Package in a host-satellite system, there will be a significant gain in speed if you apply the technique described in this section.

2.12.1 DISPLY: Rapid Creation of Display Files

Form: CALL DISPLY(n)

Argument:

The n argument is the mode switch for DISPLY. Legal values for n are -1, 0, and 1.

Instructions:

DISPLY is a control routine that you use to specify whether graphic instructions entered in the display file will be displayed immediately, or whether the instructions will be put in an "inactive display list" and then later displayed as a block.

When you call DISPLY with n=-1, no calls to the DECgraphic-11 subroutines that follow will display new primitives. Instead, the instructions will be put into the inactive display list. The exact location and size of the inactive display list depends on the type of system configuration you are using.

If you are using RT-11 or a stand-alone RSX-11M system, the inactive display list is the unused part of the display file. That is, the active display list consists of the primitives up to the DHALT, 0 instructions (see Appendix C) that mark the end of the current display. The remaining display-file space will be available for the inactive display list, and any subsequent graphic subroutine calls will insert primitives in this area. For example, the sequence

```
          .
          .
          .
CALL DISPLY(-1)
CALL APNT(200.,200.,,-4)
CALL VECT(100.,0.)
CALL VECT(0.,100.)
CALL VECT(-100.,0.)
CALL VECT(0.,-100.)
          .
          .
          .
```

enters instructions in the inactive display list for an absolute point and four vectors, which will form a square on the screen when they are displayed. Because of the CALL DISPLY(-1), however, these primitives will not yet be displayed on the screen.

When you have prepared the display file in this manner, you can display the image in two ways:

1. You can call DISPLY again with n=0 (or with no argument). This call will move the DHALT, 0 instructions to the first two vacant words in the display file. The primitives that were formerly in the inactive display list are now in the active display list and will immediately appear on the screen. However, this call does not change the display "mode"; that is, subsequent primitives are still inserted in the inactive display list. You can continue to write calls to the graphic subroutines, but no new images will automatically appear on the screen.

2. If you want to "activate" the inactive display list and also to return to the normal, automatic display mode, call DISPLY with n=1. The instructions in the inactive list will immediately be activated and the corresponding primitives will appear on the screen. Furthermore, subsequent primitives will now be inserted at the end of the active display list so that they appear on the screen immediately.

In a host-satellite system (RSX-11 and IAS only), the effect of the DISPLY arguments is essentially the same; however, the inactive display list in this case is actually a separate buffer in the host computer's memory, not a part of the display file (which is in the satellite's memory).

NOTE

Some subroutines (for example, SUBP and ESUB) must call DISPLY (0) internally. This occurrence can cause portions of the display to appear even though you have not directly called DISPLY.

If you are programming in a stand-alone system, you should choose carefully between this technique for rapid display files and the technique described in Section 3.1.2. DISPLY is a simpler solution to the problem and does not create an unnecessary subpicture. However, it is often convenient to use subpictures so that you will be able to refer back to a set of display instructions later in the program. Once CALL DISPLY(1) moves the instructions from the inactive to the active display list, there is no way to distinguish them from other display instructions. For this reason, you may often want to use subpictures when you also use DISPLY.

Although the same considerations apply in a host-satellite system, the gain in speed by using DISPLY makes it a much better method. As Section 5.3 explains, each DECgraphic-11 subroutine call that you make in satellite programs requires that the host send an "operation code coming" message to the satellite, then transmit the code itself, and finally receive an acknowledgment from the satellite. The warning message and acknowledgment make up a "communication overhead" for each subroutine call; when added up for several subroutine calls, the communication overhead will delay execution of the program a great deal.

The advantage of using DISPLY in a host-satellite system is that the operation codes for the subroutines are not executed immediately; they are actually placed in a buffer at the host end and then transmitted to the satellite as a block when you call DISPLY(1) or DISPLY(0). The communication overhead for this transmission will be the same as the overhead for a single subroutine call, but for a considerable saving in time.

Routines called by DISPLY: TOSAT (h-s)

Routines that call DISPLY: None.

2.13 COMPRESSING, SAVING, AND RESTORING THE DISPLAY FILE

DECgraphic-11 display files can be stored and retrieved on mass-storage devices. With the subroutines described in this section, you can write a program that draws pictures and saves them on a disk or magnetic tape. The display files stored in this way can also be retrieved by other programs, so that you need not repeat all the subroutine calls that created the display file in the first instance.

The first subroutine described here, CMPRS, also provides the valuable service of reclaiming space in the display file that is being taken up by erased primitives and subpictures. The SAVE subroutine (Section 2.13.2) also compresses the display file before writing it on your mass-storage device.

2.13.1 CMPRS: Compressing the Display File

Form: CALL CMPRS

Arguments: None.

Instructions:

CMPRS is used to reclaim space in the display file that is occupied by erased primitives and subpictures.

After part of a graphic program is executed, the display file may contain primitives and subpictures that are no longer needed and have been erased. The ERAS and ERASP subroutines do not actually delete elements of the display file; they simply tell the display processor to skip the erased elements so that they will not be displayed. Therefore, the space occupied by these erased display elements is not yet available for other purposes. Although the tags associated with erased primitives and subpictures are immediately available, the space in the display file is not. You must periodically call CMPRS to condense the display file and reclaim the space used by the erased display elements.

NOTE

A CMPRS call invalidates all current pointer assignments and detaches all primitives from the tracking object.

Example:

This example ilustrates a reasonable technique for monitoring the amount of space remaining in the display file; in this case, the amount is expressed as a percentage of the total initial space. It also calls CMPRS automatically when this total drops below a certain threshold -- 10%. The DPTR subroutine (see Section 2.14.1) is used to determine the amount of space used, and NMBR is used to display the percentage remaining in the display file. You could insert this fragment in a program and branch to it whenever the display file needs to be checked.

```
        .
        .
        .
     CALL INIT (2000)
        .
        .
        .
100  CALL DPTR (NEXT)
     PLEFT = (2001.-FLOAT(NEXT))/20.
     CALL APNT (0.,0.,,-4)
     CALL NMBR (2000,PLEFT,5,'(F5.1)')
     IF (PLEFT.GT.10.) GO TO 200
     CALL CMPRS
     CALL DPTR (I)
     IF (I.LT.NEXT) GO TO 200
     WRITE (5,110)
110  FORMAT (' WARNING: DISPLAY FILE NEARLY FULL')
200  CONTINUE
        .
        .
        .
```

Routines called by CMPRS: STOP, CONT, TOSAT (h-s)

Routines that call CMPRS: SAVE, RSTR

2.13.2 SAVE: Saving the Display File

Form: CALL SAVE ('[dev:]file descriptor')

Arguments:

The entire argument string must be enclosed in single quotation marks ('). The optional dev: is any legal device code used in your operating system (for example, DT0: for DECtape Drive 0). If you do not include dev:, the display file will be written on your system device. The file descriptor is the legal form of file names for your operating system, including the User Identification Code (UIC) for Files-11 systems. All the usual defaults apply; for example, if you do not include the UIC (RSX-11 and IAS only), the display file will be written with the current UIC of your terminal. If you do not specify an extension for the file name, the default is .DAT for RT-11 and no extension for IAS and RSX-11.

Instructions:

SAVE compresses the display file and saves it as a data file on a mass-storage device, such as a disk or floppy disk. You can restore the saved file as part of the current display file with the RSTR subroutine (see Section 2.13.3).

SAVE is particularly useful in creating display files that can be called in from secondary storage as needed by an application program. You can use one program to create displays that other programs can retrieve without incurring the overhead required to create new display files. For example, a display file "picture library," consisting of a large number of subpictures, can be created and saved on a disk. When the saved display file is restored, individual subpictures can be turned on and copied as desired with ON and COPY calls in the restoring program.

NOTE

A SAVE call invalidates all current pointer assignments and detaches all primitives from the tracking object.

An example using SAVE is included in the description of the RSTR subroutine (see Section 2.13.3).

Routines called by SAVE: CMPRS, ACCESS (h-s), READWR (h-s), FRSAT (h-s), TOSAT (h-s)

Routines that call SAVE: None.

2.13.3 RSTR: Restoring the Display File

Form: CALL RSTR ('[dev:]file descriptor')

Arguments:

The entire argument string must be enclosed in single quotation marks ('). The optional dev: is any legal device code used in your operating system (for example, DT0: for DECtape Drive 0). If you do not include dev:, the display file will be read from your system device. The file descriptor is the legal form of file names for your operating system, including the User Identification Code (UIC) for Files-11 systems. All the usual defaults apply; for example, if you do not include the UIC (RSX-11 and IAS only), the display file will be written with the current UIC of your terminal. If you do not specify an extension for the file name, the default is .DAT for RT-11 and no extension for IAS and RSX-11.

Instructions:

RSTR is used to restore a display file that has been saved on a mass-storage device by SAVE (see Section 2.13.2). RSTR copies the saved file into either an empty display file (i.e., one that has been initialized by INIT) or a display file that is not full. In the latter case, the restored file is appended to the end of the current display file.

When the restored file is appended to an existing display file, you should be aware that tags in the original file may duplicate those in the restored file area. In the DECgraphic-11 FORTRAN Graphics Package, any reference to a tag will always apply to the first occurrence of the tag in the file. To avoid such ambiguities, number your subpictures carefully.

If you find that two subpictures in the display file have the same tag, you can copy the original subpicture with a new tag, then erase the old subpicture. You will then be able to address the restored subpicture by referring to the new tag.

A restored display file brings with it all the properties it had when originally created, including values for the display parameters.

NOTE

A RSTR call invalidates all current pointer assignments and detaches all primitives from the tracking object.

Example:

As shown here, Program 1 can create a display and save the display file, and Program 2 can restore it.

```
                    .
                    .
                    .
[PROGRAM 1]     CALL VECT (X,Y)
                    .
                    .
                    .
                CALL SAVE ('PICTUR,DSP')


[PROGRAM 2]     COMMON/DFILE/IBUF(2000)
                    .
                    .
                    .
                CALL INIT (2000)
                CALL RSTR ('PICTUR.DSP')
                    .
                    .
                    .
```

Routines called by RSTR: CMPRS, ACCESS (h-s), READWR (h-s), TOSAT (h-s)

Routines that call RSTR: None.

2.14 INSERTING ADVANCED DISPLAY-FILE INSTRUCTIONS

This section describes three subroutines that can aid advanced users of the DECgraphic-11 FORTRAN Graphics Package. They allow you to insert new data and instructions in the display file without using any of the usual DECgraphic-11 FORTRAN subroutines.

These three special subroutines provide detailed control over the contents of the display file and should be considered advanced features. Do not use them unless you are confident that you understand the effects they will have on the rest of the display file.

All three of the subroutines in this section operate on the word currently occupied by the DHALT instruction. DHALT normally is the second-to-last word in the display file (the last word is a 0), and it thus marks the end of the active display list. However, if you have called the DISPLY subroutine (see Section 2.12.1) with a -1 argument, subsequent DECgraphic-11 subroutine calls will insert primitives after the DHALT, 0. After such a DISPLY call, the DPTR, DPYNOP, and DPYWD subroutines described in this section will address the first vacant word in the display file, rather than the word occupied by DHALT. This change in operation is required because DISPLY adds the new primitives to the active display list by relocating the DHALT, 0 instructions so that they once more occupy the last two nonvacant display-file words.

See Appendix C for a more detailed description of display-file structure.

2.14.1 DPTR: Returning the Next Available Display-File Position

Form: CALL DPTR (I)

Arguments:

If your display file is defined by the statement COMMON/DFILE/IDISP(1000), the dummy argument I returns a subscript of IDISP. IDISP(I) is the element that contains the DHALT instruction ending the display file, and is therefore the element that will contain the next instruction or data word entered in the display file.

Instructions:

One of the simpler uses of DPTR is to determine how much of the display file is currently in use. The description of CMPRS (see Section 2.13.1) shows DPTR being used for this purpose.

Routines called by DPTR: TOSAT (h-s), FRSAT (h-s)

Routines that call DPTR: None.

2.14.2 DPYNOP: Inserting No-Operation Instructions in the Display File

Form: CALL DPYNOP (n)

Arguments:

The n argument represents the number of display-file words, beginning at the end of the display file, that will be filled with "no-operation" instructions.

Instructions:

The DPYNOP subroutine allows you to insert any number of no-operation instructions at the end of the display file. These instructions make room for later insertion of data or instructions in the display file, with simple assignment statements such as

```
IDISP(200)="164000
```

where IDISP(200) is a display-file element that was filled with a no-operation instruction by DPYNOP.

As described in Section 2.14.1, you can determine the position of the DHALT instruction (that is, the location that will be addressed by a DPYNOP call) by calling DPTR.

Routines called by DPYNOP: None.

Routines that call DPYNOP: None.

2.14.3 DPYWD: Inserting a Data Word in the Display File

Form: CALL DPYWD (iword,idisp)

Arguments:

The iword argument is an integer that contains a 16-bit word to be entered at the position currently occupied by the DHALT instruction that ends the display file. The idisp argument indicates whether the inserted data will be displayed immediately.

Instructions:

DPYWD will insert the information in iword at the end of the current display file; iword will be inserted in place of the DHALT instruction, which will be moved forward one word.

When the idisp argument is 0, the result of the inserted iword will be displayed immediately on the screen. If idisp is a nonzero integer, the insertion is not immediately displayed, thus facilitating the successive insertion of several words of data.

Again, DPYWD should be used cautiously, and not at all unless you fully understand the structure of display files.

Routines called by DPYWD: None.

Routines that call DPYWD: None.

CHAPTER 3

PROGRAMMING TECHNIQUES

This chapter summarizes a variety of hints for making your graphic programs more efficient in memory use and execution time. It is divided into two major categories:

- subpicture techniques
- general graphic techniques

Some of the methods described in these categories are also included in Chapter 2 in the discussion of specific FORTRAN subroutines. They are repeated here for ease of reference. These techniques are illustrated in the FORTRAN programming examples in Appendix D.

3.1 SUBPICTURE TECHNIQUES

The programming techniques described in this section should help you in defining and using subpictures in DECgraphic-11 programming. See the discussions of subpictures in Sections 1.4.1 and 2.4.

3.1.1 Using Subpictures Like Subroutines

DECgraphic-11 programming is heavily oriented to the use of subpictures. Approximately half of the FORTRAN graphic subroutines described in Chapter 2 are intended for use in defining and manipulating subpictures or primitives within subpictures. For example, when a light-pen hit occurs, the position of the hit in a subpicture is returned. Pointers are positioned within subpicture definitions. On the VS60, characters and vectors can be enlarged or shrunk (with the CVSCAL subroutine) in particular subpictures. Subpictures can be turned on and off and can be erased.

In DECgraphic-11 programming, use subpictures for the same reasons that you use subroutines in ordinary FORTRAN. Group related graphic operations in subpictures for the sake of modularity and to help impose an orderly structure on the application program. "Tag" as a subpicture any item that you intend to refer to or modify. The use of subpictures is especially suited for -- but not limited to -- the expression of graphic entities that are repeated frequently in the display file.

Multiple displays of the same primitive group can be created in at least two ways:

1. You can call the subroutines that produce the primitives the required number of times; or

2. You can put the primitive-generating subroutines in a subpicture and then repeat the subpicture the required number of times.

Table 3-1 shows how to decide between these two techniques in order to use the available memory most efficiently. In Table 3-1, the letter S means "use repeated subpictures"; the letter P means "use repeated primitive subroutine calls."

Note that if the number of display-file words for one display of the primitives and the number of repetitions are sufficiently small, there is no advantage in terms of memory in using subpictures. However, subpictures always have the advantage of modularity and easy reference to specific picture components.

Table 3-1
When to Use Subpictures

Repetitions	Number of Words in Primitive Group											
	1	2	3	4	5	6	7	8	9	10	11	12
1	P	P	P	P	P	P	P	P	P	P	P	P
2	P	P	P	P	P	P	P	P	P	P	P	S
3	P	P	P	P	P	P	P	P	S	S	S	S
4	P	P	P	P	P	P	P	S	S	S	S	S
5	P	P	P	P	P	P	S	S	S	S	S	S
6	P	P	P	P	P	P	S	S	S	S	S	S
7	P	P	P	P	P	P	S	S	S	S	S	S
8	P	P	P	P	P	[=]	S	S	S	S	S	S
9	P	P	P	P	P	S	S	S	S	S	S	S
10	P	P	P	P	P	S	S	S	S	S	S	S
11	P	P	P	P	P	S	S	S	S	S	S	S

As an example of using Table 3-1, consider a group of three long vectors (created by the LVECT subroutine). Long vectors each require two words of memory, so the primitive group takes up six words of space. If you were to repeat this group in a display, you could use the space most efficiently by repeating the LVECT calls, up to seven times. At eight repetitions, using subpictures would require the same space, and for more repetitions, subpictures would require less space.

Table 3-1 is a tabulation of the following formula, which you can use to calculate the space-saving advantages for other cases:

$$5(\text{No. reps})+2+(\text{No. words}) < (\text{No. reps})(\text{No. words})$$

The left side of the inequality is the space required by repeated subpictures. Therefore, if the inequality condition is true, subpictures will take less space. You can see from the formula and from Table 3-1 that the advantage of subpictures is especially

dramatic when the number of words (i.e., the number of primitives) is very large. It is often desirable to use subpictures for reasons other than the obvious benefit of saving space in the display file. Some of the other reasons are described in the following sections.

3.1.2 Creating "All-at-Once" Displays

There are two approaches to creating graphic images. The more conventional approach is to display successive primitives on the screen as they are inserted in the display file. This approach has the effect of creating a "growing" picture on the display screen, one that changes dynamically as new primitives are entered.

As an alternative, you may want to display a graph or picture only after it has been completely defined. To implement this "all-at-once" technique, define the entire display as a subpicture (the SUBP subroutine) that is turned off (the OFF subroutine) as soon as it is begun. Then turn the subpicture on again (the ON subroutine) after the definition is complete. This approach is shown below.

```
CALL SUBP (100)
CALL OFF (100)
      .
      .
      .
definition
      .
      .
      .
CALL ESUB (100)
CALL ON (100)
```

Besides providing a good method of displaying a picture all at once, the use of this technique also speeds up the process of image creation.

NOTE

See the DISPLY subroutine, Section 2.12, for another method of creating quick, all-at-once displays.

3.1.3 Moving Subpictures on the Screen

You can move a subpicture around the display screen by attaching it to the tracking object (see the ATTACH subroutine, Section 2.8.4). When you are drawing a subpicture that is to be moved in this fashion, use an APNT call (see Section 2.3.1) as the first primitive in the subpicture, but make all the remaining primitives relative.

If several instances of the same subpicture are to be moved separately on the screen, then the APNT call should be moved out of the subpicture. Call APNT separately before each instance of the subpicture.

3.1.4 Creating Odometer Displays

You can use the NMBR subroutine to create odometer-type displays (see Section 2.4.7 for a description of NMBR). However, this application normally requires the use of run-time formatting code, which requires a great deal of memory. The use of memory will be much more efficient if you perform simple integer and floating-point output conversions in FORTRAN to accomplish the same task.

To accomplish the desired task, first convert the number to be displayed to an ASCII string and append a null byte to the string. Then, in order to display the number, call CHANGT (see Section 2.6.6) to change a character primitive that you have set up at the beginning of your program.

The subroutine ITOA is a special-purpose FORTRAN Graphics Package subroutine that converts integer variables to ASCII codes. ITOA has the form

```
CALL ITOA(integr,ibytes,LOGICL)
```

where integr is the name of the integer variable to converted, ibytes gives the number of bytes required to hold the ASCII values (one for each integral digit), and LOGICL is a LOGICAL*1 array that is at least large enough to contain all the ASCII codes. ITOA will insert the ASCII codes in the array LOGICL, with the high-order digit of integr in Byte 1, and so forth.

The example with the subroutine TRAKXY (see Section 2.8.3) shows TEXT, CHANGT, and ITOA being used for an odometer display.

The odometer can be intensified by means of the INTENS subroutine (see Section 2.7.2) or set to flash mode with the FLASH subroutine (see Section 2.7.3). If the numeric output is in its own subpicture and you are using a VS60, you can also scale the size of the characters by calling CVSCAL (see Section 2.4.8).

3.2 GENERAL GRAPHIC TECHNIQUES

The programming techniques described in this section describe a variety of approaches to speeding up program execution in the DECgraphic-11 FORTRAN Graphics Package.

3.2.1 Choosing the Appropriate Vector Format

Vectors can be drawn in any of three legal formats:

- short relative format
- long relative format
- long absolute format

Short vectors require only half the storage of long vectors, but have a limited range. A short vector is stored in one word and a long vector in two words.

The range of a short vector is 63 raster units along the x and y axes, or approximately one sixteenth of a full screen. The range of a long vector, on the other hand, is 1023 raster units.

In general, you should use the short-vector format when constructing static displays that consist of a large number of short lines (e.g., shading of small objects, very irregular outlines).

Short vectors can be altered by calls to the GET, CHANGA, and CHANGE subroutines (see Sections 2.6.3, 2.6.4, and 2.6.5), but not by attaching to the tracking object (see the ATTACH subroutine, Section 2.8.4). Long-vector format should be used when you are attaching a vector to the tracking object -- for example, for the "rubber-banding" of lines on the display screen.

Absolute vectors (VS60 only) should never be used in subpictures that are to be moved on the screen; you should always use relative vectors in such subpicture definitions. Absolute vectors do offer one significantly different feature: changing one absolute vector does not change the end-point location of the next vector.

When you are uncertain about the precise length of a vector in physical screen units, but are not concerned about subsequently changing the size of the vector, you should use the VECT subroutine (see Section 2.3.3). This subroutine automatically uses the short-vector format whenever possible. Unless short and long vectors alternate frequently, this technique optimizes the use of display-file space.

3.2.2 Ordering Picture Elements

A display often consists of several static elements, such as menus and background, as well as a dynamic picture that grows by repeatedly adding lines, points, and characters to the existing display. When constructing a display of this kind, create the static portions first. This technique is more efficient than using the INSRT subroutine (see Section 2.6.7) to add elements within the display file.

Static elements such as menus can be conveniently defined in a subpicture that is first turned off (see Section 3.1.2) and then turned on when needed.

3.2.3 Monitoring the Display File

The DPTR subroutine is helpful in determining how much space remains in the display file. You may want your program to take appropriate automatic actions when the file reaches a certain size, such as that provided by an automatic call to CMPRS (see Section 2.13.1).

3.2.4 Avoiding a Temporary Loss of a Display

The DECgraphic-11 FORTRAN Graphics Package provides four subroutines (SENSE, INTENS, FLASH, and LINTYP) that allow you to change the display parameters (l, i, f, t) defined earlier in primitives within a subpicture. However, the first time you use one of these subroutines within the subpicture, the display screen may become blank for a moment, and then reappear. A similar problem can occur when you use CVSCAL to scale the characters or vectors within a subpicture. This

temporary blanking of the screen can interfere with the effectiveness of some applications. The cause of this occurrence, and some steps you can take to avoid a temporary loss of the display, are described in the paragraphs that follow.

When you define a new subpicture, you typically set display parameters by including one or more of the optional arguments l, i, f, and t in the first primitive. Including some or all of these arguments creates a mode word in the display file. This mode word precedes the instructions that actually display your images. Figure 3-1 is a stylized presentation of such a file.

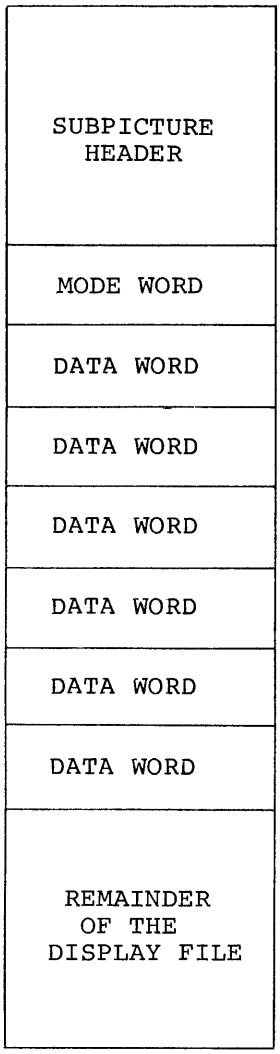

Figure 3-1 A Mode Word in a Display File

The subpicture contains instructions for three displayed objects (there are three sets of coordinates), but it contains only a single mode word. Therefore, this subpicture will display three absolute points, none of which is light-pen sensitive, and all of which are visible at Intensity Level 4.

Suppose that, at a subsequent point in the program, you use the INTENS subroutine to make the third point brighter than Points 1 and 2. The INTENS call results in a call to the INSRT subroutine, which inserts a new mode word before the x coordinate of Point 3. The new intensity level would then affect only the third point. It is this call to the INSRT subroutine that causes the temporary blanking of the screen.

You can avoid the temporary blanking by inserting a "dummy" display parameter (1, i, f, or t) in each primitive in the subpicture. Then each primitive has its own mode word, and the INSRT operation will not occur. For example:

```
CALL SUBP(100)
CALL RPNT(100.,100.,-1,-4,-1,1)
CALL VECT(0.,100.,0)
CALL VECT(100.,0.,-1)
CALL VECT(0.,-100.,0)
CALL VECT(-100.,0.,-1)
```

The call to RPNT sets the display parameters as follows: no light-pen sensitivity (l=-1), the current primitive is invisible and has an absolute intensity value of 4 (i=-4), the display will not flash (f=-1), and the vectors that follow will be drawn as solid lines (t=1). Each of the four following calls to VECT includes a single display parameter, which is sufficient to create a new mode word for each primitive. The alternating values 0 and -1 for the 1 display parameter leave light-pen sensitivity unchanged. Thus, any change in the value of a display parameter creates a new mode word, even if the two values have the same effect. Accordingly these four VECT calls display four solid-line segments at Intensity Level 4. None of the line segments flash and none are sensitive to the light pen.

Subsequent calls to SENSE, INTENS, FLASH, or LINTYP can change the parameters of any primitive (for example to make the first two vectors visible) without temporarily blanking the screen.

NOTE

This technique will require an additional word of memory for each primitive in the subpicture.

To avoid the temporary blanking problem with CVSCAL, insert a call to DPYWD with the arguments "154000, 0 immediately after the call to SUBP (which begins the subpicture definition), as follows:

```
CALL SUBP(120)
CALL DPYWD("154000, 0)
```

In effect the DPYWD subroutine inserts a "dummy" CVSCAL instruction in the program.

3.2.5 Using DPYWD and DISPLY to Speed Up Instruction Input

A useful technique for speeding up instruction input on a VS60 display subsystem is to combine features of DPYWD (see Section 2.14.3) and DISPLY (see Section 2.12.1) in the following manner:

```
CALL DPYWD("170010,0)
CALL DISPLY(-1)
```

The DPYWD call inserts the octal value 170010 at the end of the current display file to replace DHALT (which moves forward one word). The instruction 170010 changes the "refreshment" cycle of DECgraphic-11 display files from the usual free-running self-synchronized rate (see below) to a slower rate.

Ordinarily the refreshment of DECgraphic-11 files is self-synchronized. That is, the display processor goes through the display file from start to finish, taking as much time as it requires, and then immediately returns to the beginning of the display file and starts again. The instruction 170010 changes this self-synchronized process to one that occurs at a fixed rate of 40 frames per second. This rate generally is slower than the self-synchronized rate. Since a decrease in refreshment rate entails a decrease in the rate of direct memory accesses (DMAs) by the VS60 display processor, there is more UNIBUS time available to the CPU, and the program's speed is increased.

With this change in rate, it becomes important to minimize the number of additions to the active display list since each addition may require waiting one fortieth of a second instead of the considerably faster self-synchronized operation. The DISPLY(-1) call, which places the primitives entered immediately after the call in an inactive display list, is used for this purpose. Use of this call can also reduce transmission overhead in host-satellite systems.

When you want to activate the inactive display list and to return to automatic display mode, use the following call:

```
CALL DISPLY(1)
```

The instructions you have inserted in the inactive display list will be activated immediately and the corresponding primitives will appear on the screen. The DISPLY(1) call also returns the display subsystem to its ordinary mode, in which subsequent primitives are inserted at the end of the active display list and immediately appear on the screen.

To return the display processor to self-synchronized operation later in the program, use the DPTR subroutine immediately before DPYWD:

```
C     MARK POSITION
      CALL DPTR(I)
      CALL DPYWD("170010,0)
C     BEGIN FAST INSTRUCTION INPUT
      CALL DISPLY(-1)
           .
           .
           .
C     RESTORE TO ACTIVE DISPLAY LIST
      CALL DISPLY(1)
C     REPLACE 170010 WITH NO-OP
      IBUF(I)="164000
```

In this sequence, IBUF is assumed to be the name of your display file.

The use of DPTR in this sequence returns (in I) the number of the IBUF element that will receive the instruction created by the next call; in this case, that instruction is 170010. At a later point, IBUF(I)="164000 rewrites this display-file word with the "no-operation" instruction 164000. Now all primitives are displayed at the self-synchronized rate.

CHAPTER 4

INSTRUCTIONS FOR RT-11 USERS

This chapter gives full instructions for using the DECgraphic-11 FORTRAN Graphics Package with the RT-11 operating system, whether you are using a VT11 or VS60 graphic-display subsystem. The RT-11 instructions are arranged in the following way in this chapter:

Section 4.1 tells you what software is supplied in the DECgraphic-11 FORTRAN Graphics Package kit, and how to use it to build a library of object code for graphic programming.

Section 4.2 tells you how to link the library to each graphic program you write in FORTRAN.

4.1 BUILDING A FORTRAN GRAPHICS LIBRARY

Your DECgraphic-11 FORTRAN Graphics Package kit contains three source files:

COND.FOR, which generates source code that is tailored to the specific hardware you are using;

GRPACK.CND, a file of FORTRAN subroutine definitions;

GRSUBS.MAC, a file of MACRO-11 assembly-language code.

To copy the RK05 or RK06 software to your system device, put the distribution disk in Drive 1 and your RT-11 system disk in Drive 0. Then boot up your system and type the red parts of the following dialog:

```
.COPY dev1:*.* dev0: RET
.
```

[devn: is DK0: for RK05, and DM0: for RK06.]

The first step in creating a graphic library is to convert COND from a FORTRAN source file to a runnable RT-11 load module, or .SAV file. Boot up your RT-11 system and type the red parts of the following dialog:

```
.FOR COND RET
.LINK COND RET
.
```

The FORTRAN compiler creates the object file COND.OBJ, and the linker creates the load module COND.SAV. Now run the COND program by typing the red responses:

```
.R COND (RET)
COND V2.1
FILENAME?  GRPACK (RET)
```

COND will start running and will begin printing a series of messages. It will first identify itself: DECGRAPHIC-11 FORTRAN GRAPHICS PACKAGE / V 1.1. It then will ask you a series of questions. The first question is

```
WOULD YOU LIKE THE LONG FORM OF THE QUESTIONS (Y OR N)?
```

If this is the first time you have run COND, type Y followed by a carriage return. This response makes COND explain each question in detail and makes the library building procedure more understandable. When you run COND in the future, you can do without the detailed explanations.

The full COND dialog is shown here:

```
.R COND (RET)
COND V2.1
FILE NAME ? GRPACK(RET)
DECGRAPHIC-11 FORTRAN GRAPHICS PACKAGE / V1.1

WOULD YOU LIKE THE LONG FORM OF THE QUESTIONS (Y OR N) ? Y(RET)

THIS PROGRAM PRODUCES SEVERAL FILES NEEDED FOR THE GENERATION
OF THE DECGRAPHIC-11 LIBRARY. THESE FILES, TOGETHER WITH THE
LIBRARY ITSELF CAN BE DIRECTED TO ANY FILE STRUCTURED DEVICE.
OUTPUT DEVICE (DDU:) ? DK2:(RET)

THE GENERATION PROCESS PRODUCES (OPTIONALLY) SEVERAL LISTING
FILES WHICH CAN BE DIRECTED TO ANY PRINTER-LIKE DEVICE FOR
IMMEDIATE OUTPUT, OR TO SOME OTHER DEVICE FOR LATER LISTING.
IF A FILE STRUCTURED DEVICE IS SPECIFIED AND THE SYSTEM CONTAINS
A PRINT SPOOLER, THE LISTINGS WILL BE AUTOMATICALLY SPOOLED.
LISTING DEVICE (DDU:) ? LP0:(RET)

THE DECGRAPHIC-11 SOFTWARE SUPPORTS BOTH THE VT11 AND VS60
DISPLAY PROCESSORS CONNECTED DIRECTLY TO THE UNIBUS, OR THE
GT43 AND GT62 DISPLAY TERMINALS CONNECTED VIA A COMM-
UNICATION INTERFACE. IF YOU HAVE EITHER A VS60 OR GT62 ANSWER
YES TO THIS QUESTION.
VS60 (Y OR N) ? Y(RET)

THE VS60 CAN OPTIONALLY SUPPORT TWO SCOPES (OR DISPLAY SCREENS)
ON THE SAME CONTROLLER.
TWO SCOPES (Y OR N) ? Y(RET)

THE LK-11 PUSHBUTTON BOX CAN ALSO BE SUPPORTED AS A
PART OF THE PACKAGE.  IT IS TYPICALLY USED BY THE APPLICATION
AS AN ALTERNATIVE OR SUPPLEMENT TO LIGHT PEN MENU SELECTION
AS A PROGRAM CONTROL TECHNIQUE.
LK-11 ? Y(RET)

THE DECGRAPHIC-11 SOFTWARE WILL RUN UNDER SEVERAL OPERATING
SYSTEMS. IN THE FOLLOWING QUESTION(S) ANSWER YES FOR THE SYSTEM
YOU ARE USING.
RT-11 (Y OR N) ? Y(RET)
```

Note: Short form of questions is underlined.

```
THE DECGRAPHIC-11 SOFTWARE WILL PRODUCE FAIRLY DESCRIPTIVE
ERROR MESSAGES UPON DETECTION OF ANY ERRORS IN THE USE OF
THE GRAPHIC SUBROUTINES. UNFORTUNATELY THE TEXT OF THESE
MESSAGES TAKES UP A FAIRLY LARGE AMOUNT OF MEMORY SPACE.IF
YOU WOULD LIKE TO ELIMINATE THIS TEXT, ANSWER NO TO THE NEXT
QUESTION. IN THIS EVENT, ANY ERROR WILL PRODUCE A CODE WHICH
IS DESCRIBED IN APPENDIX B OF THE DECGRAPHIC-11
FORTRAN PROGRAMMING MANUAL.
ERROR MESSAGE TEXT (Y OR N) ? N(RET)

THE SOFTWARE MAY BE GENERATED TO ACCEPT UNSCALED INTEGER DATA
INSTEAD OF REAL DATA. IN THE INTEGER CASE, ALL NUMBERS REPRE-
SENTING POINTS AND VECTORS MUST BE INTEGERS IN THE APPROPRIATE
RANGE (SHORT VECTOR: -63<>63, LONG VECTOR: -1023<>1023, ABSOLUTE
POINT FOR VT11: 0<>1023, ABSOLUTE POINT, ABSOLUTE VECTOR, OR
VIEWPORT FOR VS60: -4095<>4095). THE ADVANTAGES OF USING THE
INTEGER FORMAT ARE SMALLER PROGRAMS (DATA ONLY TAKE ONE WORD,
NOT TWO) AND FASTER EXECUTION TIMES (INTEGER ARITHMETIC IS
FASTER THAN REAL ARITHMETIC). THE DISADVANTAGE IS LESS FLEXI-
BILITY IN CREATING DISPLAYS NOT REPRESENTED IN CONVENIENT UNITS.
INTEGER ARGUMENTS ? N(RET)

THE GENERATION PROCEDURE WILL PRODUCE LISTINGS OF BOTH THE
MACRO-11 AND FORTRAN COMPONENTS OF THE DECGRAPHIC-11 SOFTWARE
IF DESIRED. THESE LISTINGS WILL BE SENT TO THE LISTING DEVICE
NAMED PREVIOUSLY.
MACRO LISTINGS (Y OR N) ? N(RET)

FORTRAN LISTINGS (Y OR N) ? Y(RET)

THE COMMAND FILE NORMALLY PRODUCED WILL DELETE ALL TEMPORARY
FILES CREATED DURING THE GENERATION PROCESS. THESE FILES MAY
BE PRESERVED IF DESIRED.
DELETE FILES (Y OR N) ? Y(RET)

NOTE:  BEFORE CONTINUING WITH THE BUILD PROCESS ...
MAKE SURE THAT THE FILE GRSUBS.MAC IS ON DK2:
STOP --

.
```

COND is specific about which type of display subsystem you are using, whether or not you have an LK-11 pushbutton box, whether you will use both real and integer arguments or integers only, and so forth. COND is using these questions to set up a library that is specially tailored to your hardware and your programming needs. If your needs change, or if you add or subtract hardware from your graphic-display subsystem, you will have to run COND again to create a new library.

After you answer the last question (DELETE FILES?), COND begins its conditionalizing process. At this point, COND is actually choosing from the FORTRAN source code in GRPACK.CND to create a new library source that will include only features you have selected by your answers to COND. This process requires a minute or more to complete. When the message STOP -- appears on the terminal, the conditionalizing process is finished, and COND returns control to the RT-11 monitor.

COND has created the following new files on your output device:

GRPAK1.FO, a conditionalized form of the FORTRAN source code in GRPACK.CND.

GRSUBS.MA, a conditionalized MACRO-11 file that has recorded your answers to the COND questions;

GRCOM.CND, a FORTRAN source file that defines the DECgraphic-11 COMMON areas DFILE and GRDAT.

GRGEN.BAT, an RT-11 BATCH control file that compiles and assembles the source code into a library;

GRLINK.BAT, which will link your programs to the library.

You now use the file GRGEN.BAT to create the actual DECgraphic-11 FORTRAN Graphics Package library. Before running GRGEN.BAT, be sure you have 600 contiguous blocks of space available in memory. Use the following sequence:

```
.ASSIGN TT: LOG (RET)
.ASSIGN TT: LST (RET)
.LOAD BA (RET)
.R BATCH (RET)
*GRGEN (RET)
```

The first action of GRGEN is to delete old versions of the files used to create previous DECgraphic-11 libraries. GRGEN also deletes the file GLIB.OBJ from your output device (that is, GRGEN erases any old version of a DECgraphic-11 library). This deletion is done to conserve space on your storage device, but if you want to save several different libraries on the same device, use the RENAME command to give each library a unique name. Typical names are:

GLIBVT.OBJ (for VT11 subsystems)

GLIBVS.OBJ (for VS60 subsystems)

GLIBIN.OBJ (for an "integer-only" library)

If you use such names for your old libraries (or any name other than GLIB.OBJ), they will be preserved.

The results of GRGEN are summarized below:

1. All files on your output device with the extensions .OB or .LI are deleted (these are old object and listing files created by previous runs of GRGEN).

2. The file GLIB.OBJ is deleted from your output device.

3. The file GRSUBS.MAC is conditionally assembled to create the object file GRSUBS.OB: if you told COND you wanted MACRO listings, they will be printed on your listing device.

4. The entire file GRPAK1.FO will be compiled and (optionally) listed. The object file will be named GRPAK1.OB.

5. If you told COND to DELETE FILES, all files with the extension .FO are now deleted.

6. GRGEN now uses the RT-11 Librarian (LIBR) to create GLIB.OBJ, which is your new DECgraphic-11 subroutine library.

7. If you told COND to DELETE FILES, GRGEN now deletes GRGEN.BAT, GRCOM.CND, and all files with the extensions .MA and .OB.

8. The message $EOJ appears when GRGEN is finished.

When GRGEN is finished, you are ready to link the DECgraphic-11 FORTRAN Graphics Package to graphic programs that you have written. The next section describes this process.

4.2 LINKING PROGRAMS TO THE DECGRAPHIC-11 FORTRAN GRAPHICS PACKAGE

As mentioned in the last section, it is often desirable to give your DECgraphic-11 library a unique name so that it will not be accidentally deleted by later runs of GRGEN.BAT. In the following examples, a new library for the VT11 subsystem is renamed:

```
·RENAME GLIB.OBJ GLIBVT.OBJ (RET)
·
```

The name GLIBVT is now assigned to the VT11 library. To link this library to a graphic program called GRAPH.FOR, proceed as follows:

```
·FOR GRAPH (RET)
·LINK GRAPH, GLIBVT (RET)
·
```

You can then run the resulting load module, GRAPH.SAV, by typing

```
·R GRAPH (RET)
```

USR OPERATIONS

Some FORTRAN statements and functions, such as CALL ASSIGN and CALL CLOSE, require the RT-11 USR (User Service Routine) to be swapped into memory.

When you use one of the these statements, you must call the STOP subroutine (Section 2.1.2) before the USR operation takes place. Otherwise, the display will disappear from the screen and the display processor will hang. When the USR operation is complete, you can restart the display with a call to CONT (see Section 2.1.3). You can also protect the display with the RT-11 SET USR NOSWAP command, although that command will require much more memory. Section 2.13.3 has an example of the STOP and CONT subroutines being used to protect the display. For more information on the USR, refer to the RT-11 System Reference Manual.

CHAPTER 5

INSTRUCTIONS FOR RSX-11 AND IAS USERS

If you are using the RSX-11M operating system, the DECgraphic-11 FORTRAN Graphics Package can be used in two ways: to support a VT11 or VS60 display subsystem operating as a UNIBUS peripheral device (called a "stand-alone" system) or to support a "satellite" graphic-display subsystem operating as an intelligent terminal to an RSX-11M host system. The DECgraphic-11 FORTRAN Graphics Package also runs in the RSX-11D and IAS operating systems; in either of these two cases it must use the host-satellite mode.

This chapter is organized as follows:

Sections 5.1 and 5.2 describe procedures for building DECgraphic-11 libraries and for using the libraries to create graphic tasks, respectively. Both sections contain separate instructions for stand-alone systems and host-satellite systems.

Section 5.3 gives new users a brief overview of the principles of host-satellite programming with the DECgraphic-11 FORTRAN Graphics Package. Section 5.3.1 describes a typical host-satellite configuration and discusses communication between the host computer and the graphic satellite. Section 5.3.2 describes the DECgraphic-11 subroutines that make the host-satellite communication possible; for more advanced applications, you can modify the routine USRSAT, which handles host-satellite messages. Section 5.3.3 gives instructions for running graphic tasks (programs) at the satellite. For many applications, perhaps the majority, you can go straight to Section 5.3.3 for operating instructions after building your libraries. The programmer should understand the principles by which DECgraphic-11 manages host-satellite systems; however, the routines described in Section 5.3.2 are invisible to the user unless USRSAT has been modified.

5.1 BUILDING DECGRAPHIC-11 LIBRARIES

This section gives you full instructions for preparing the DECgraphic-11 library for your operating system. Sections 5.1.1 through 5.1.4 contain important information for all users of the RSX-11 and IAS systems. After you have read these sections and completed the tasks they describe, go on to either Section 5.1.5 (for a stand-alone system) or 5.1.6 (for a host-satellite system).

5.1.1 Contents of the Software Kit

Your DECgraphic-11 FORTRAN Graphics Package kit contains the following software, regardless of the operating system or operating mode you will use:

COND.FOR, a FORTRAN program in source form that will generate graphic libraries;

GRPACK.CND, the file of FORTRAN subroutine sources on which COND will operate;

GRSUBS.MAC, a second input file for COND that contains MACRO-11 assembly-language code.

PBDRV.MAC and PBTAB.MAC, which are source files containing code for the driver and the driver table for the LK-11 pushbutton box. If you intend to use an LK-11 pushbutton box with a stand-alone RSX-11M system, do a new system generation that includes these two files in the system.

NOTE

The host-satellite software assumes that your satellite communications interface (DL11) has a CSR=175610 and vector address=300. If your interface is not at this standard address, edit the file GRSUBS.MAC and change the variables DLICSR and DLVEC to the correct CSR and vector address, respectively.

5.1.2 Summary of Hardware/Software Configurations

In summary, the following software from the kit is required for each operating system and configuration:

Stand-alone systems with RSX-11M and one or two graphic scopes:

COND.FOR, GRPACK.CND, and GRSUBS.MAC

Stand-alone systems with RSX-11M, one or two scopes, and an LK-11:

COND.FOR, GRPACK.CND, GRSUBS.MAC, PBDRV.MAC (SYSGEN), and PBTAB.MAC (SYSGEN)

All host-satellite systems:

COND.FOR, GRPACK.CND, and GRSUBS.MAC

The system-generation dialog for RSX-11M gives full instructions for assembling and incorporating "user-written" device drivers, such as PBDRV. See also the RSX-11M System Generation Manual for your version of the operating system.

System-Generation Options

If you are generating a new operating system, be sure to comply with the requirements described in Section 1.2.

5.1.3 Copying the Software Kit

The distribution kit for RK05 and RK06 disks is in RT-11 file format. The magnetic-tape kit is in DOS-11 format. To copy the software onto your Files-11 system device, follow the procedure given for your operating system.

To copy the RK05 or RK06 software to an RSX-11M system, put the distribution disk in Drive 0 and type the following commands in response to the operating system's prompt symbol (> or MCR>):

```
FLX SY:/RS=dev:*.*/RT RET
```

[dev: is DK0: for RK05, and DM0: for RK06.]

To copy the magnetic-tape software, thread the distribution tape on Drive 0 and type the following commands. (If your system has loadable device drivers, you may have to type LOA MM: or LOA MT: first.)

```
FLX SY:/RS=dev:[*,*]*.*/DO RET
```

[dev: is MM0: for TJU16 drives, and MT0: for TU10 drives.]

NOTE

Use the same command for copying to an RSX-11D system, but precede the FLX command with the command

```
MOU dev:/CHA=[FOR] RET
```

where dev: is the device containing the software kit (i.e., DK0:, DM0:, MM0:, or MT0:).

To copy the RK05 or RK06 software to an IAS system device, put the distribution disk in Drive 0 and type the red parts of this dialog:

```
PDS>MOUNT/FOREIGN/NOOPERATOR RET
DEVICE?dev: RET
VOLUME-ID?XXX RET
PDS>COPY RET
FROM?dev:*.*/RT11 RET
TO?*.* RET
```

[dev: is DK0: for RK05, and DM0: for RK06.]

To copy the magnetic-tape software to an IAS system device, thread the distribution tape on Drive 0 and type the red parts of this dialog:

```
PDS>MOUNT/FOREIGN/NOOPERATOR (RET)
DEVICE?MM0: (RET)
VOLUME-ID?XXX (RET)
PDS>COPY (RET)
FROM?dev:[*,*]*.*/DOS (RET)
TO?*.* (RET)
```

[dev: is MT0: instead of MM0: if you have a TU10 tape drive.]

5.1.4 Compiling and Linking COND

Your first step in creating a DECgraphic-11 library is to compile and build a task from the file COND.FOR. After you have your operating system running and have copied the DECgraphic-11 kit onto the system device, prepare COND by typing the red parts of the following dialog:

RSX-11M:

```
>FOR COND=COND.FOR (RET)
>TKB COND=COND (RET)
```

RSX-11D:

```
MCR>FOR COND=COND.FOR (RET)
MCR>TKB COND=COND (RET)
```

IAS:

```
PDS>FORTRAN COND.FOR (RET)
PDS>LINK COND (RET)
```

Now begin the library generation process by typing

```
RUN COND (RET)
```

In response to the prompt symbol for your operating system, COND will print its own version number, ask you for a file name (the answer is GRPACK), and begin asking you a series of questions. The first question is:

```
WOULD YOU LIKE THE LONG FORM OF THE QUESTIONS (Y OR N)?
```

If you have not run COND before, type Y followed by a carriage return. The long form is self-explanatory and will help you make correct decisions. When you run COND again in the future, you can do without the long form.

If you are going to create a library for use on a stand-alone RSX-11M system, proceed to Section 5.1.5. If you are an RSX-11D or IAS user, or if you are creating a host-satellite library for your RSX-11M system, see Section 5.1.6.

5.1.5 RSX-11M Stand-Alone Systems

Here is a sample listing of a COND dialog creating a stand-alone system for RSX-11M:

```
.R COND (RET)
COND V2.1
FILE NAME ? GRPACK (RET)
DECGRAPHIC-11 FORTRAN GRAPHICS PACKAGE / V1.1


WOULD YOU LIKE THE LONG FORM OF THE QUESTIONS (Y OR N) ? Y(RET)


THIS PROGRAM PRODUCES SEVERAL FILES NEEDED FOR THE GENERATION
OF THE DECGRAPHIC-11 LIBRARY. THESE FILES, TOGETHER WITH THE
LIBRARY ITSELF CAN BE DIRECTED TO ANY FILE STRUCTURED DEVICE.
OUTPUT DEVICE (DDU:) ? DK2:(RET)


THE GENERATION PROCESS PRODUCES (OPTIONALLY) SEVERAL LISTING
FILES WHICH CAN BE DIRECTED TO ANY PRINTER-LIKE DEVICE FOR
IMMEDIATE OUTPUT, OR TO SOME OTHER DEVICE FOR LATER LISTING.
IF A FILE STRUCTURED DEVICE IS SPECIFIED AND THE SYSTEM CONTAINS
A PRINT SPOOLER, THE LISTINGS WILL BE AUTOMATICALLY SPOOLED.
LISTING DEVICE (DDU:) ? LP0:(RET)


THE DECGRAPHIC-11 SOFTWARE SUPPORTS BOTH THE VT11 AND VS60
DISPLAY PROCESSORS CONNECTED DIRECTLY TO THE UNIBUS, OR THE
GT43 AND GT62 DISPLAY TERMINALS CONNECTED VIA A COMM-
UNICATION INTERFACE. IF YOU HAVE EITHER A VS60 OR GT62 ANSWER
YES TO THIS QUESTION.
VS60 (Y OR N) ? Y (RET)


THE VS60 CAN OPTIONALLY SUPPORT TWO SCOPES (OR DISPLAY SCREENS)
ON THE SAME CONTROLLER.
TWO SCOPES (Y OR N) ? Y (RET)


THE LK-11 PUSHBUTTON BOX CAN ALSO BE SUPPORTED AS A
PART OF THE PACKAGE.  IT IS TYPICALLY USED BY THE APPLICATION
AS AN ALTERNATIVE OR SUPPLEMENT TO LIGHT PEN MENU SELECTION
AS A PROGRAM CONTROL TECHNIQUE.
LK-11 ? Y (RET)


THE DECGRAPHIC-11 SOFTWARE WILL RUN UNDER SEVERAL OPERATING
SYSTEMS. IN THE FOLLOWING QUESTION(S) ANSWER YES FOR THE SYSTEM
YOU ARE USING.
RT-11 (Y OR N) ? N (RET)


RSX-11M (Y OR N) ? Y(RET)
```

THE DECGRAPHIC-11 SOFTWARE WILL SUPPORT SEVERAL POSSIBLE HOST-SATELLITE CONFIGURATIONS. THESE CONFIGURATIONS ALL INVOLVE A MAIN SYSTEM RUNNING EITHER RSX-11M, RSX-11D, OR IAS, AND ONE OR MORE GRAPHIC TERMINALS (GT43 OR GT62) CONNECTED TO THE MAIN SYSTEM VIA A COMMUNICATION LINE. IF YOU ANSWER NO TO THE FOLLOWING QUESTION, IT IMPLIES THAT YOU ARE RUNNING WITH THE DISPLAY PROCESSOR (VT11 OR VS60) CONNECTED DIRECTLY TO THE UNIBUS.
HOST-SATELLITE (Y OR N) ? N RET

THE DECGRAPHIC-11 SOFTWARE WILL PRODUCE FAIRLY DESCRIPTIVE ERROR MESSAGES UPON DETECTION OF ANY ERRORS IN THE USE OF THE GRAPHIC SUBROUTINES. UNFORTUNATELY THE TEXT OF THESE MESSAGES TAKES UP A FAIRLY LARGE AMOUNT OF MEMORY SPACE.IF YOU WOULD LIKE TO ELIMINATE THIS TEXT, ANSWER NO TO THE NEXT QUESTION. IN THIS EVENT, ANY ERROR WILL PRODUCE A CODE WHICH IS DESCRIBED IN APPENDIX B OF THE DECGRAPHIC-11 FORTRAN PROGRAMMING MANUAL.
ERROR MESSAGE TEXT (Y OR N) ? N RET

THE DECGRAPHIC-11 SOFTWARE ALSO SUPPORTS THE FORTRAN IV-PLUS COMPILER, WHICH GIVES MUCH FASTER PROGRAM EXECUTION AT THE COST OF MEMORY SPACE. THE FORTRAN IV-PLUS SYSTEM REQUIRES THAT THE FP11 HARDWARE BE PRESENT WHEN THE PROGRAM IS EXECUTED. NOTE: FORTRAN IV-PLUS CANNOT BE USED IN THE SATELLITE OF HOST-SATELLITE SYSTEM.
FORTRAN IV-PLUS (Y OR N) ? Y RET

THE SOFTWARE MAY BE GENERATED TO ACCEPT UNSCALED INTEGER DATA INSTEAD OF REAL DATA. IN THE INTEGER CASE, ALL NUMBERS REPRESENTING POINTS AND VECTORS MUST BE INTEGERS IN THE APPROPRIATE RANGE (SHORT VECTOR: -63<>63, LONG VECTOR: -1023<>1023, ABSOLUTE POINT FOR VT11: 0<>1023, ABSOLUTE POINT, ABSOLUTE VECTOR, OR VIEWPORT FOR VS60: -4095<>4095). THE ADVANTAGES OF USING THE INTEGER FORMAT ARE SMALLER PROGRAMS (DATA ONLY TAKE ONE WORD, NOT TWO) AND FASTER EXECUTION TIMES (INTEGER ARITHMETIC IS FASTER THAN REAL ARITHMETIC). THE DISADVANTAGE IS LESS FLEXIBILITY IN CREATING DISPLAYS NOT REPRESENTED IN CONVENIENT UNITS.
INTEGER ARGUMENTS ? N RET

THE GENERATION PROCEDURE WILL PRODUCE LISTINGS OF BOTH THE MACRO-11 AND FORTRAN COMPONENTS OF THE DECGRAPHIC-11 SOFTWARE IF DESIRED. THESE LISTINGS WILL BE SENT TO THE LISTING DEVICE NAMED PREVIOUSLY.
MACRO LISTINGS (Y OR N) ? N RET

FORTRAN LISTINGS (Y OR N) ? Y RET

```
THE COMMAND FILE NORMALLY PRODUCED WILL DELETE ALL TEMPORARY
FILES CREATED DURING THE GENERATION PROCESS. THESE FILES MAY
BE PRESERVED IF DESIRED.
DELETE FILES (Y OR N) ? Y(RET)


NOTE:  BEFORE CONTINUING WITH THE BUILD PROCESS ...
MAKE SURE THAT YOU HAVE INSTALLED LBR
MAKE SURE THAT THE FILE GRSUBS.MAC IS ON DK2:
STOP --

.
```

Notice that the answer to the question RSX-11M (Y OR N) ? is Y, and the answer to HOST-SATELLITE (Y OR N) ? is N.

Note also that the COND program includes specific questions about the display subsystem in use (and number of scopes), the presence or absence of an LK-11 pushbutton box, and the use of integer-only subroutine arguments. Your answers to these questions tell COND to change the FORTRAN code in some subroutines and to delete sources for unneeded subroutines from the library. Therefore, if your hardware configuration or programming needs change, you will have to run COND again and make a new DECgraphic-11 library.

After you answer the last COND question (DELETE FILES (Y OR N) ?), COND begins the actual "conditionalizing" process: It records your answers to its questions, removes unnnecessary code from the FORTRAN sources, and prepares for the final creating of a library of object code. This process requires a minute or more to complete. When the message STOP -- appears, the conditionalizing process is finished. COND has created the following new files on your output device:

GRSUBS.MA, which records your answers to the COND questions, ready to be passed to the MACRO-11 Assembler;

GRGEN.CMD, an "indirect" command file that will be used to create the final library;

GRPAK1.FO, a subset of the FORTRAN sources in GRPACK.CND, created according to your answers to the COND questions;

GRCOM.CMD, which defines the DECgraphic-11 COMMON areas DFILE and GRDAT;

GRBLD.CMD, another indirect command file that builds graphic tasks by combining your compiled FORTRAN programs with the library created by GRGEN.CMD.

When these files have been created, you are ready to finish building the library. Type the red command:

>@GRGEN (RET)

The RSX-11M MCR commands in GRGEN.CMD will be executed, with the following results:

1. All old files with the extensions .OB and .LI will be deleted from your output device. This clean-up operation removes old versions of the DECgraphic-11 object modules and FORTRAN listings from your storage device, to conserve storage space. Any old versions of the DECgraphic-11 library (GLIB.OLB) will also be deleted. If your storage device contains old libraries that you want to save, use the Peripheral Interchange Program (PIP) to rename them; for example, to rename your old VT11 library:

   >PIP GLIBVT.OLB=GLIB.OLB/RE (RET)

2. GRGEN assembles the MACRO-11 routines in GRSUBS.MAC, using the information in GRSUBS.MA to remove unneeded code from the object file. GRGEN also compiles the FORTRAN sources in GRPAK1.FO, using either the standard FORTRAN IV compiler (FOR) or the FORTRAN IV-PLUS compiler (F4P), depending on your instructions to COND. If you told COND that you wanted them, MACRO-11 and FORTRAN listings will be printed on your listing device.

3. GRGEN now combines the object modules GRPAK1.OB and GRSUBS.OB into a new DECgraphic-11 library, GLIB.OLB.

4. Finally, GRGEN deletes all but the most recent version of GRBLD.CMD and, if you told COND to DELETE FILES, also deletes all versions of all files that have the extensions .FO, .OB, and .MA.

The process of creating a library does not change the contents of GRPACK.CND or GRSUBS.MAC; the DECgraphic-11 software in your kit remains in its original form. Now that you have created a new library, it is a good practice to use PIP immediately to give the library a new name. Typical names for libraries are:

GLIBVT.OLB (for libraries built for the VT11 subsystem);

GLIBVS.OLB (for VS60 libraries);

GLIBIN.OLB (for "integer-only" libraries).

Giving unique names to your finished libraries allows you to save them all on the same storage device, with no danger of accidental deletion by GRGEN.CMD.

Now see Section 5.2.1 for instructions on building graphic tasks.

5.1.6 IAS, RSX-11M, and RSX-11D Host-Satellite Systems

Here is a sample of a COND listing for creating an IAS host-satellite system:

```
.R COND
COND V2.1
FILE NAME ? GRPACK (RET)
DECGRAPHIC-11 FORTRAN GRAPHICS PACKAGE / V1.1


WOULD YOU LIKE THE LONG FORM OF THE QUESTIONS (Y OR N) ? Y(RET)


THIS PROGRAM PRODUCES SEVERAL FILES NEEDED FOR THE GENERATION
OF THE DECGRAPHIC-11 LIBRARY. THESE FILES, TOGETHER WITH THE
LIBRARY ITSELF CAN BE DIRECTED TO ANY FILE STRUCTURED DEVICE.
OUTPUT DEVICE (DDU:) ? DK2:(RET)


THE GENERATION PROCESS PRODUCES (OPTIONALLY) SEVERAL LISTING
FILES WHICH CAN BE DIRECTED TO ANY PRINTER-LIKE DEVICE FOR
IMMEDIATE OUTPUT, OR TO SOME OTHER DEVICE FOR LATER LISTING.
IF A FILE STRUCTURED DEVICE IS SPECIFIED AND THE SYSTEM CONTAINS
A PRINT SPOOLER, THE LISTINGS WILL BE AUTOMATICALLY SPOOLED.
LISTING DEVICE (DDU:) ? LP0:(RET)


THE DECGRAPHIC-11 SOFTWARE SUPPORTS BOTH THE VT11 AND VS60
DISPLAY PROCESSORS CONNECTED DIRECTLY TO THE UNIBUS, OR THE
GT43 AND GT62 DISPLAY TERMINALS CONNECTED VIA A COMM-
UNICATION INTERFACE. IF YOU HAVE EITHER A VS60 OR GT62 ANSWER
YES TO THIS QUESTION.
VS60 (Y OR N) ? N(RET)


THE LK-11 PUSHBUTTON BOX CAN ALSO BE SUPPORTED AS A
PART OF THE PACKAGE.  IT IS TYPICALLY USED BY THE APPLICATION
AS AN ALTERNATIVE OR SUPPLEMENT TO LIGHT PEN MENU SELECTION
AS A PROGRAM CONTROL TECHNIQUE.
LK-11 ? Y(RET)


THE DECGRAPHIC-11 SOFTWARE WILL RUN UNDER SEVERAL OPERATING
SYSTEMS. IN THE FOLLOWING QUESTION(S) ANSWER YES FOR THE SYSTEM
YOU ARE USING.
RT-11 (Y OR N) ? N(RET)


RSX-11M (Y OR N) ? N(RET)


RSX-11D (Y OR N) ? N(RET)


IAS (Y OR N) ? Y(RET)
```

```
WHEN GENERATING THE DECGRAPHIC-11 SOFTWARE FOR A HOST-SATELLITE
CONFIGURATION IT IS NECESSARY TO MAKE TWO PASSES THROUGH
THIS PROGRAM, ONE FOR EACH END OF THE SYSTEM.
SATELLITE END (Y OR N) ? N(RET)


THE DECGRAPHIC-11 SOFTWARE ALSO SUPPORTS THE FORTRAN IV-
PLUS COMPILER, WHICH GIVES MUCH FASTER PROGRAM EXECUTION AT
THE COST OF MEMORY SPACE. THE FORTRAN IV-PLUS SYSTEM REQUIRES
THAT THE FP11 HARDWARE BE PRESENT WHEN THE PROGRAM IS EXECUTED.
NOTE: FORTRAN IV-PLUS CANNOT BE USED IN THE SATELLITE OF
HOST-SATELLITE SYSTEM.
FORTRAN IV-PLUS (Y OR N) ? N(RET)

THE SOFTWARE MAY BE GENERATED TO ACCEPT UNSCALED INTEGER DATA
INSTEAD OF REAL DATA. IN THE INTEGER CASE, ALL NUMBERS REPRE-
SENTING POINTS AND VECTORS MUST BE INTEGERS IN THE APPROPRIATE
RANGE (SHORT VECTOR: -63<>63, LONG VECTOR: -1023<>1023, ABSOLUTE
POINT FOR VT11: 0<>1023, ABSOLUTE POINT, ABSOLUTE VECTOR, OR
VIEWPORT FOR VS60: -4095<>4095). THE ADVANTAGES OF USING THE
INTEGER FORMAT ARE SMALLER PROGRAMS (DATA ONLY TAKE ONE WORD,
NOT TWO) AND FASTER EXECUTION TIMES (INTEGER ARITHMETIC IS
FASTER THAN REAL ARITHMETIC). THE DISADVANTAGE IS LESS FLEXI-
BILITY IN CREATING DISPLAYS NOT REPRESENTED IN CONVENIENT UNITS.
INTEGER ARGUMENTS ? Y(RET)


THE GENERATION PROCEDURE WILL PRODUCE LISTINGS OF BOTH THE
MACRO-11 AND FORTRAN COMPONENTS OF THE DECGRAPHIC-11 SOFTWARE
IF DESIRED. THESE LISTINGS WILL BE SENT TO THE LISTING DEVICE
NAMED PREVIOUSLY.
MACRO LISTINGS (Y OR N) ? N(RET)


FORTRAN LISTINGS (Y OR N) ? Y(RET)


THE COMMAND FILE NORMALLY PRODUCED WILL DELETE ALL TEMPORARY
FILES CREATED DURING THE GENERATION PROCESS. THESE FILES MAY
BE PRESERVED IF DESIRED.
DELETE FILES (Y OR N) ? Y(RET)


NOTE:  BEFORE CONTINUING WITH THE BUILD PROCESS ...
MAKE SURE THAT YOU HAVE INSTALLED LBR
MAKE SURE THAT THE FILE GRSUBS.MAC IS ON DK2:
STOP --

.
```

When you say Y to the question IAS?, COND assumes that you are building a host-satellite library. The same assumption is made for RSX-11D. These systems are always used in the host-satellite mode. With RSX-11M, which can be used in either mode, COND asks the question HOST-SATELLITE?. Answering N implies a stand-alone system.

The three host-satellite systems have one requirement in common; you must make two passes through COND. In other words, you build two libraries for host-satellite systems, one for the "satellite end" and one for the "host end." Answering N to the question SATELLITE END? tells COND that you are building the host-end library. After you have answered N, proceed to the end of the COND dialog and let COND finish its conditionalizing process. Then use the GRGEN command to create the library for the host end. When GRGEN is finished, run COND a

second time, but this time answer Y to the question SATELLITE END?. Now the questions from COND will be slightly different because you are making the satellite library:

```
.R COND
COND V2.1
FILE NAME ? GRPACK(RET)
DECGRAPHIC-11 FORTRAN GRAPHICS PACKAGE / V1.1


WOULD YOU LIKE THE LONG FORM OF THE QUESTIONS (Y OR N) ? Y(RET)


THIS PROGRAM PRODUCES SEVERAL FILES NEEDED FOR THE GENERATION
OF THE DECGRAPHIC-11 LIBRARY. THESE FILES, TOGETHER WITH THE
LIBRARY ITSELF CAN BE DIRECTED TO ANY FILE STRUCTURED DEVICE.
OUTPUT DEVICE (DDU:) ? DK2:(RET)


THE GENERATION PROCESS PRODUCES (OPTIONALLY) SEVERAL LISTING
FILES WHICH CAN BE DIRECTED TO ANY PRINTER-LIKE DEVICE FOR
IMMEDIATE OUTPUT, OR TO SOME OTHER DEVICE FOR LATER LISTING.
IF A FILE STRUCTURED DEVICE IS SPECIFIED AND THE SYSTEM CONTAINS
A PRINT SPOOLER, THE LISTINGS WILL BE AUTOMATICALLY SPOOLED.
LISTING DEVICE (DDU:) ? LP0:(RET)


THE DECGRAPHIC-11 SOFTWARE SUPPORTS BOTH THE VT11 AND VS60
DISPLAY PROCESSORS CONNECTED DIRECTLY TO THE UNIBUS, OR THE
GT43 AND GT62 DISPLAY TERMINALS CONNECTED VIA A COMM-
UNICATION INTERFACE. IF YOU HAVE EITHER A VS60 OR GT62 ANSWER
YES TO THIS QUESTION.
VS60 (Y OR N) ? N(RET)


THE LK-11 PUSHBUTTON BOX CAN ALSO BE SUPPORTED AS A
PART OF THE PACKAGE.  IT IS TYPICALLY USED BY THE APPLICATION
AS AN ALTERNATIVE OR SUPPLEMENT TO LIGHT PEN MENU SELECTION
AS A PROGRAM CONTROL TECHNIQUE.
LK-11 ? Y(RET)


THE DECGRAPHIC-11 SOFTWARE WILL RUN UNDER SEVERAL OPERATING
SYSTEMS. IN THE FOLLOWING QUESTION(S) ANSWER YES FOR THE SYSTEM
YOU ARE USING.
RT-11 (Y OR N) ? N(RET)


RSX-11M (Y OR N) ? N(RET)


RSX-11D (Y OR N) ? N(RET)


IAS (Y OR N) ? Y(RET)
```

```
WHEN GENERATING THE DECGRAPHIC-11 SOFTWARE FOR A HOST-SATELLITE
CONFIGURATION IT IS NECESSARY TO MAKE TWO PASSES THROUGH
THIS PROGRAM, ONE FOR EACH END OF THE SYSTEM.
SATELLITE END (Y OR N) ? Y(RET)


THE SATELLITE WILL CONTAIN THREE MAIN COMPONENTS: THE SATELLITE
CONTROL PROGRAM, THE DISPLAY FILE, AND THE (OPTIONAL) EXTENSIONS
TO THE CONTROL PROGRAM. THE CONTROL PROGRAM OCCUPIES ABOUT
8K WORDS OF MEMORY, WITH THE REMAINDER BEING AVAILABLE FOR
DISPLAY FILE PLUS USER EXTENSIONS TO THE CONTROL PROGRAM.
THIS LEAVES 8K WORDS IN A 16K SATELLITE. IF YOU PLAN TO EXTEND
THE CONTROL PROGRAM, SUBTRACT THE AMOUNT OF SPACE YOU WILL
NEED FROM 8K AND EXPRESS THAT AS OCTAL BYTES (HINT: 2K WORDS
IS 10000 OCTAL BYTES). YOUR ANSWER WILL BE THE SIZE OF
THE SATELLITE'S DISPLAY FILE UNTIL YOU RUN THIS PROGRAM
AGAIN TO CHANGE THE SIZE.
DISPLAY FILE SIZE (OCTAL BYTES) ? 10000(RET)


IF YOU HAVE A GT62 TERMINAL WITH RX11 (FLOPPY) DISKS, YOU
MAY WANT TO HAVE A LOCAL SAVE/RESTORE CAPABILITY IN ADDITION
TO THE SAVE/RESTORE AVAILABLE VIA THE HOST SYSTEM.
SELECTION OF THIS OPTION WILL ALSO PROVIDE SOME GENERAL
FILE ACCESS SUBROUTINES FOR USE IN THE SATELLITE.
IF YOU SAY 'YES' TO THE FOLLOWING QUESTION, YOU MUST
MODIFY THE USER'S SATELLITE ROUTINE (USRSAT) SO THAT
THE SAVE AND RSTR SUBROUTINES WILL BE EXECUTED BY THE
SATELLITE (SEE APPENDIX D OF THE DECGRAPHIC-11 FORTRAN
PROGRAMMING MANUAL).
LOCAL SAVE/RSTR ? Y(RET)


THE SOFTWARE MAY BE GENERATED TO ACCEPT UNSCALED INTEGER DATA
INSTEAD OF REAL DATA. IN THE INTEGER CASE, ALL NUMBERS REPRE-
SENTING POINTS AND VECTORS MUST BE INTEGERS IN THE APPROPRIATE
RANGE (SHORT VECTOR: -63<>63, LONG VECTOR: -1023<>1023, ABSOLUTE
POINT FOR VT11: 0<>1023, ABSOLUTE POINT, ABSOLUTE VECTOR, OR
VIEWPORT FOR VS60: -4095<>4095). THE ADVANTAGES OF USING THE
INTEGER FORMAT ARE SMALLER PROGRAMS (DATA ONLY TAKE ONE WORD,
NOT TWO) AND FASTER EXECUTION TIMES (INTEGER ARITHMETIC IS
FASTER THAN REAL ARITHMETIC). THE DISADVANTAGE IS LESS FLEXI-
BILITY IN CREATING DISPLAYS NOT REPRESENTED IN CONVENIENT UNITS.
INTEGER ARGUMENTS ? Y(RET)


THE GENERATION PROCEDURE WILL PRODUCE LISTINGS OF BOTH THE
MACRO-11 AND FORTRAN COMPONENTS OF THE DECGRAPHIC-11 SOFTWARE
IF DESIRED. THESE LISTINGS WILL BE SENT TO THE LISTING DEVICE
NAMED PREVIOUSLY.
MACRO LISTINGS (Y OR N) ? N(RET)


FORTRAN LISTINGS (Y OR N) ? Y(RET)
```

```
THE COMMAND FILE NORMALLY PRODUCED WILL DELETE ALL TEMPORARY
FILES CREATED DURING THE GENERATION PROCESS. THESE FILES MAY
BE PRESERVED IF DESIRED.
DELETE FILES (Y OR N) ? Y(RET)


NOTE:  BEFORE CONTINUING WITH THE BUILD PROCESS ...
MAKE SURE THAT YOU HAVE INSTALLED LBR
MAKE SURE THAT THE FILE GRSUBS.MAC IS ON DK2:
STOP --
```

When the STOP -- message appears from COND (which occurs after a delay of a minute or more), use the GRGEN command; this time it creates the final satellite-end library.

NOTES ON GRGEN PROCEDURE

1. It is vital that you follow the exact procedure described in the previous paragraph. Do not run COND twice and then command GRGEN twice. If you do, only one library will be created and the host-satellite system will not work.

2. The GRGEN command file assumes that the standard FORLIB is to be used in compiling your satellite library. If for any reason different FORTRAN libraries are used to build the host- and satellite-end libraries, re-edit the file GRGEN.CMD so that the TKB command will retrieve the proper FORTRAN library.

When you run COND for a host-satellite system, eight new files will be created on your output device. The first four are identical in function to the files created in a stand-alone system:

GRSUBS.MA, which records your answers to the COND questions, ready to be passed to the MACRO-11 Assembler;

GRGEN.CMD (called GRGEN.BIS in RSX-11D), an "indirect" command file that will be used to create the final library;

GRPAK1.FO, a subset of the FORTRAN sources in GRPACK.CND, created according to your answers to the COND questions;

GRCOM.CND, which defines the DECgraphic-11 COMMON areas DFILE and GRDAT.

The name of the fifth file depends on whether you are building a library for the host end or the satellite end:

GRSBLD.CMD, created when you are building for the satellite end; this file will create a new satellite control program (SATCTL.TSK); the section on building host-satellite graphic tasks (Section 5.2.2) explains this command in more detail;

GRHBLD.CMD, created when you are building for the host end; this file will link your compiled graphic program to the host-end library, creating a graphic task that can be executed by commands from the satellite. Again, see Section 5.2.2 for more information.

The sixth file is created only when you are building the host library:

LGR.FO, a FORTRAN (or FORTRAN IV-PLUS) source file containing the Graphic Loader; its use will be explained in Section 5.2.2.

Finally, the seventh and eighth files will be created only when you are building the satellite library:

SATDSP.FO, the source file of the Satellite Dispatcher;

USRSAT.FO, the source file of the User's Satellite Routine.

These last two files are explained in Section 5.2.2.

For the creation of the libraries, follow the same procedure, whether you are creating the host-end or satellite-end library:

RSX-11M:

```
>@GRGEN (RET)
```

RSX-11D:

```
MCR>BAT GRGEN (RET)
```

IAS:

```
PDS>@GRGEN (RET)
```

The GRGEN command performs the same functions in all three operating systems. When you are building the host-end library, GRGEN performs the following functions:

1. Deletes all files with the extensions .OB and .LI (that is, the object and listing files left over from previous runs of GRGEN).

2. Deletes all versions of the file GLIBH.OLB (old host-end libraries).

3. Using the MACRO-11 Assembler, conditionally assembles the code in GRSUBS.MAC, with the COND answers in GRSUBS.MA serving as conditional assembly parameters. The output file is GRSUBS.OB.

4. Depending on your instructions to COND, uses either the FORTRAN IV or FORTRAN IV-PLUS compiler to compile GRPAK1.FO, and prints listings on your listing device if you asked for them. The output file is named GRPAK1.OB.

5. Uses the FORTRAN IV or FORTRAN IV-PLUS compiler to compile LGR.FO, and optionally prints listings. The output file is named LGR.OBJ. Builds the task image LGR.TSK from LGR.OBJ and prints the task-build map.

6. Uses the system's Librarian to create GLIBH.OLB, the new host-end graphic library.

7. If you told COND to DELETE FILES, deletes all versions of all files with the extensions .FO, .OB, and .MA, as well as all versions of GRCOM.CND.

8. Deletes all but the most recent versions of LGR.OBJ and GRHBLD.CMD (that is, all versions except the ones you have just created).

If you are building the satellite-end library, GRGEN performs different operations:

1. Deletes all files with the extensions .OB and .LI and all old versions of GLIBS.OLB (old satellite-end libraries).

2. Conditionally assembles GRSUBS.MAC. Listings are printed if you request them. Note that this is a different assembly than in the host-end case, because your COND answers in GRSUBS.MA are for the satellite end this time.

3. Compiles (in FORTRAN IV only) the file GRPAK1.FO, which is also specific to the satellite end this time. Listings are printed if you request them.

4. Compiles the Satellite Dispatcher (SATDSP.FO) and the User's Satellite Routine (USRSAT.FO). Only the FORTRAN IV compiler is used; FORTRAN IV-PLUS cannot be used in satellite control programs. Listings are printed if you request them.

5. Uses the librarian to create GLIBS.OLB, the new satellite-end library.

6. If you told COND to DELETE FILES, deletes all files with the extensions .FO, .OB, and .MA, and deletes all but the most recent versions of SATDSP.OBJ, USRSAT.OBJ, and GRSBLD.CMD.

When you have finished running GRGEN this second time, both the satellite- and host-end libraries are complete. See Section 5.2.2 for instructions on building host-satellite tasks.

5.2 CREATING GRAPHIC TASKS

Graphic tasks are the final result of your graphic programming work in RSX-11 and IAS. They combine a FORTRAN or FORTRAN IV-PLUS program that you write with the libraries created from the DECgraphic-11 FORTRAN Graphics Package; they will run and display graphics on your display screen in response to simple commands. Section 5.3 describes the principles of host-satellite programming in detail. However, whether you are using the DECgraphic-11 FORTRAN Graphics Package in stand-alone mode or host-satellite mode, you may want to read this section first and try a few programs before you get into the theory of operation.

5.2.1 RSX-11M Stand-Alone Systems

In RSX-11M, tasks are created by the Task Builder (TKB). This system utility links together a number of object files to build an executable task image. One of the object files, in the case of graphic programming, will be a FORTRAN program you have written that includes calls to the subroutines described in this manual. The other file will be GLIB.OLB, the library of DECgraphic-11 subroutines.

One way to use the Task Builder is as follows (the red parts of the dialog are the ones you type):

```
>TKB (RET)
TKB>task=object.GLIB/LB (RET)
TKB>/ (RET)

TKB>ASG=GR0:1 (RET)
TKB>MAXBUF=512 (RET)
TKB>// (RET)
```

where:

task	= Your choice for the name of the final task image;
object	= Your compiled FORTRAN graphic program;
GLIB	= The DECgraphic-11 library;
GR0:1	= The assignment of Logical Unit 1 to the device GR0: (your VT11 or VS60);
MAXBUF=512	= The definition of the maximum buffer size when using SAVE and RSTR.

NOTE

As discussed in Section 1.2, the display-file common area (DFILE) must reside in the physical lower 28K of memory if you are using a VT11 with a mapped system. If, as was recommended, you created a global common called DFILE at the time you generated your RSX-11M system, you must include the second option COMMON=DFILE in the TKB commands.

The object file for your program can be created by the FORTRAN compiler:

```
>FOR object=source (RET)
```

or by the FORTRAN IV-PLUS compiler:

```
>F4P object=source (RET)
```

where:

source = The file of FORTRAN code you have created by any of several means, most commonly by use of the RSX-11M Text Editor (EDI).

The procedure just described creates a task image of your program, which can be executed by typing:

```
>RUN task (RET)
```

As an alternative to this procedure, you can edit the file GRBLD.CMD, which COND produces. In editing GRBLD.MAC, substitute the name of your program for the file name USER.

NOTE

GRBLD.CMD does not contain the option statement COMMON=DFILE. If you are using a mapped RSX-11M system with a VT11 and have created a global common called DFILE, you should edit GRBLD.CMD and add the COMMON=DFILE statement. Otherwise, you should not use GRBLD.

5.2.2 RSX-11M, RSX-11D, and IAS Host-Satellite Systems

Host-satellite tasks are created by the Task Builder utility (TKB in RSX-11 systems and LINK in IAS). To run a host-satellite graphic task, you need to create two related tasks:

1. The task created from your FORTRAN program. To create this task, the program (GRAPH.FTN is used as an example in this section) is compiled and then linked by TKB or LINK to the host library, GLIBH.OLB.

2. The satellite control task, SATCTL.TSK. SATCTL.TSK is built by linking SATCTL.OBJ to the satellite library (GLIBS.OLB), to the Satellite Dispatcher, and to the User's Satellite Routine (USRSAT.OBJ). USRSAT, in its simplest form, is provided automatically when you build the satellite library. You can write modifications to USRSAT to extend the capability of the host-satellite software (see Section 5.3.2). Section 5.3.2 also explains the Satellite Dispatcher and Satellite Control Task in more detail.

INSTRUCTIONS FOR RSX-11 AND IAS USERS

To prepare for running a host-satellite system, bootstrap the computer in your satellite display subsystem by the following procedure:

1. Load address 776000 and press the START button.

2. A blinking cursor will appear on the screen, meaning that your display subsystem is now a terminal to the host computer. Type either CTRL/C or CTRL/Z to make the prompt symbol for your system appear. Log on to your account on the host system (if necessary).

Your satellite subsystem is now functioning as a terminal to the host operating system. Build the task from your program (GRAPH.FTN) by typing the red commands shown below in response to your operating system's prompt:

RSX-11:

```
FOR USER=GRAPH (RET)
TKB @GRHBLD (RET)
PIP GRAPH.TSK=USER.TSK/RE (RET)
```

IAS:

```
FORTRAN/OBJECT:USER GRAPH (RET)
@GRHBLD (RET)
RENAME USER.TSK GRAPH.TSK (RET)
```

The program GRAPH.FTN is now the graphic task GRAPH.TSK, linked to the host library. The name USER.OBJ was used temporarily because that is the name used by the command file GRHBLD.CMD for input and output.

Now build the Satellite Control Task:

RSX-11:

```
TKB @GRSBLD (RET)
```

IAS:

```
@GRSBLD (RET)
```

GRSBLD.CMD uses whatever form of the User's Satellite Routine (USRSAT.OBJ) you have on your storage device. If you do not need to modify USRSAT (see Section 5.3.2), let GRSBLD use the version that was created when you built the satellite library. When GRSBLD is finished, the Satellite Control Task (SATCTL.TSK) will have been created on your storage device. Unless you modify USRSAT, this is the only time you have to use GRSBLD. The same SATCTL can run with all your programs.

As explained in Section 1.2, you must be sure that the FORTRAN library used to build SATCTL matches the hardware of the satellite terminal. GRSBLD uses the default FORTRAN library that you have incorporated in SYSLIB. If the default FORTRAN library does not match the satellite hardware, you should edit GRSBLD.CMD and add a reference in the TKB command to the proper FORTRAN library for the satellite.

Now you can run the Graphic Loader and load SATCTL.TSK down-line from the host to the satellite:

RSX-11M:

```
RUN LGR (RET)
LGR>SATCTL (RET)
```

RSX-11D/IAS:

```
RUN LGR (RET)
LGR>SATCTL (RET)
STARTING ADDRESS? 1000 (RET)
```

NOTE

If your system is RSX-11D or IAS, always check the task-build map of SATCTL (SATCTL.MAP) for the correct starting address. The starting address is not required for RSX-11M.

When SATCTL is running, the special satellite cursor appears on the screen. This cursor is a small square with a flashing center. (The process of down-line loading may require a minute or more before the cursor appears.) You are ready to run your graphic task:

```
RUN GRAPH (RET)
```

The GRAPH program will be executed by the host, creating a display file and graphic display image at the satellite.

For more details on host-satellite principles, see Section 5.3, especially the part (Section 5.3.4) dealing with special programming precautions and special keyboard characters used in host-satellite programming.

5.3 HOST-SATELLITE SYSTEMS

This section describes the concept and special software components of a DECgraphic-11 host-satellite system. As was mentioned previously, you may want to run some sample programs (such as the DRAWH/DRAWS program in Appendix D) before you concern yourself with this theoretical detail. Section 5.3.3, however, is useful for beginners.

Section 5.3.1 contains general information on the configuration of host-satellite systems.

Section 5.3.2 describes in detail the special software that is called internally by the DECgraphic-11 FORTRAN Graphics Package subroutines when they are used in a host-satellite system. This special software is in the form of an extra set of subroutines. In some situations, you can call them directly to perform special operations.

Section 5.3.3 gives a detailed discussion of running host-satellite graphic tasks, and of the function of the Graphic Loader (LGR).

Section 5.3.4 lists the precautions you must observe when writing FORTRAN programs for a host-satellite system.

Finally, Section 5.3.5 lists the special control characters that you can use on the keyboard of your satellite terminal.

5.3.1 The Host-Satellite Concept

You can best understand the concept of host-satellite graphic programming by studying Figure 5-1.

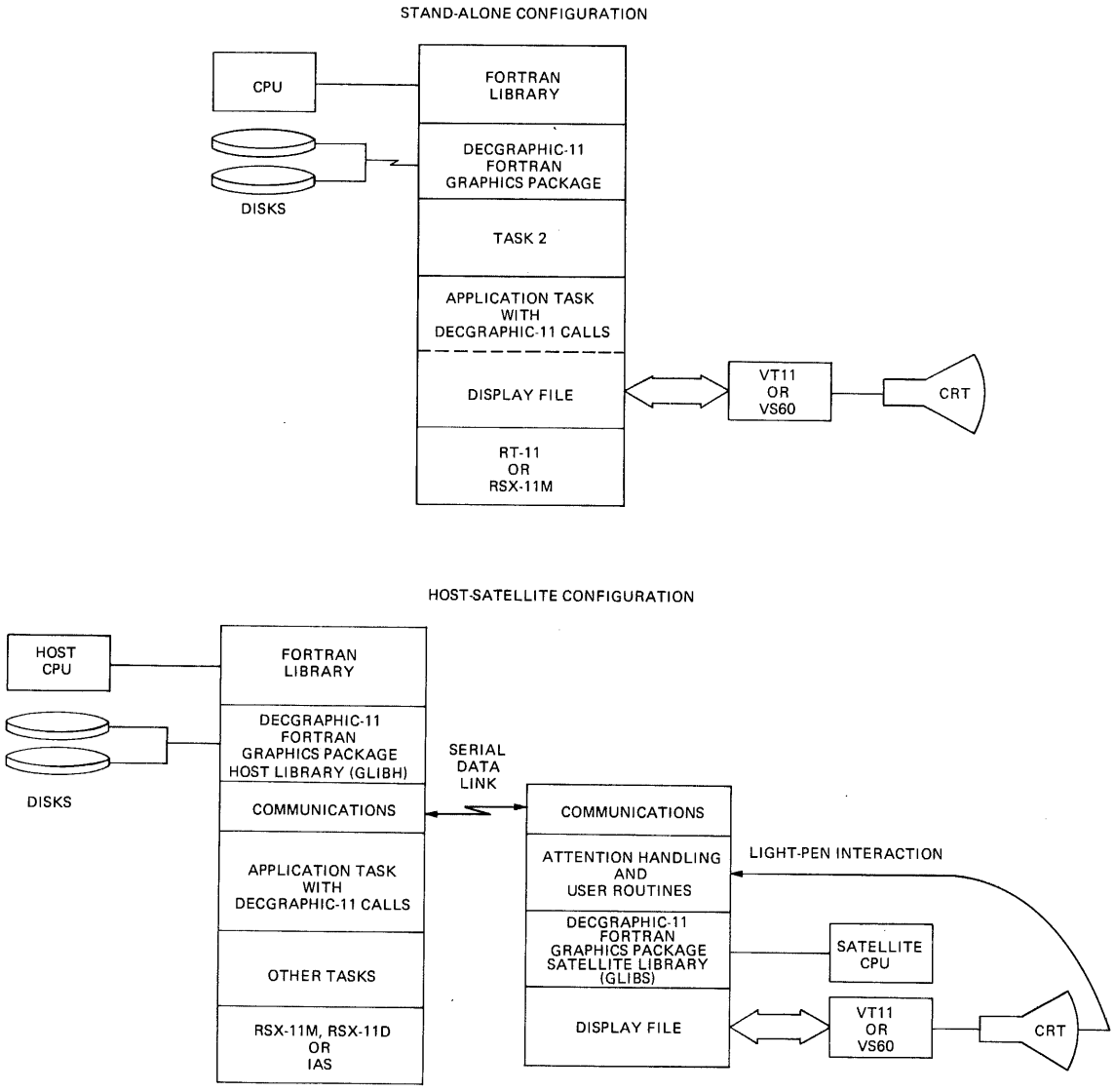

Figure 5-1 Comparison of Stand-Alone and Host-Satellite Configurations

In a stand-alone configuration, you write an application task with DECgraphic-11 subroutine calls like the ones described in this manual. These calls each retrieve object code from the disk-resident library (DECgraphic-11 FORTRAN Graphics Package), and the code inserts display instructions in the area of memory called the display file. The VS60 display processor (or the VT11, as the case may be) can then execute the instructions by using the display file as its own portion of memory, without using the CPU.

You can see the basic differences in the software configuration of host-satellite systems:

1. The graphic library is separated into two parts, the DECgraphic-11 FORTRAN Graphics Package on the host end and DECgraphic-11 image generation on the satellite end.

2. Each end of the system has an area labeled "Communications."

3. Instead of a single application task on the host end, there is also an area on the satellite end labeled "Attention Handling and User Routines."

4. There is no display file on the host end; the display file and the display hardware are on the satellite end of the system.

The two libraries are simply GLIBH.OLB and GLIBS.OLB, which were created by the COND/GRGEN procedure described in Section 5.1.6. Because the actual image generation takes place on the satellite end, these two libraries are related: a call to one of the subroutines in GLIBH will be translated into a call to a subroutine in GLIBS, so that the appropriate display instruction is placed in the display file for the satellite.

The "communication" areas of each end are in charge of translating and sending messages from host to satellite and back. The routines in these categories convert a regular subroutine call to a format that is easily transmitted across the serial data link. Communication software is described in more detail in Section 5.3.2. For many applications, you can write a host-satellite program without having to make any reference to the communication routines in your program.

The drawing shows the application task divided into two parts, one on each end of the serial data link. In the simpler cases, the task on the satellite end is simply the Satellite Control Task (SATCTL.TSK) as built by the GRSBLD command, without modification by the programmer. The initial form of SATCTL does all the basic work of communicating with the host, calling the appropriate subroutines from GLIBS, and building the display file. The application task on the host end is, in these same cases, identical to a task that would run on a stand-alone system, except for the way in which the display file is handled.

When you built the satellite-end library with COND, COND asked you how big the satellite display file should be. Your answer was recorded, and the host-satellite software defines and initializes a display file this size each time you run a host-satellite graphic task. Therefore, you do not need to put the COMMON/DFILE or CALL INIT statements at the beginning of the application task; the definition and initialization are done automatically.

Host-satellite programming becomes more complicated than programming for stand-alone systems when you write interactive graphic programs, since such programs require a lot of graphic-attention handling for the light pen or other interactive devices. In such programs, it is most efficient to modify the User's Satellite Routine (USRSAT) so that it provides local attention handling for interactive devices that are part of the satellite hardware. This technique eliminates the need to communicate with the host each time a graphic attention occurs, and the interactive program runs much faster. Appendix D has an example of an interactive program (DRAWH/DRAWS) that employs a modified USRSAT to provide attention handling at the satellite for the light pen and the pushbutton box.

5.3.2 The Host-Satellite Software

As was mentioned in describing the process of building host and satellite graphic libraries, host-satellite systems contain software not found in DECgraphic-11 stand-alone systems. This special software falls into three categories:

1. An executive task, SATCTL.TSK, that interprets messages from the host computer and the satellite terminal and executes code in the satellite to create images on the satellite screen.

2. Routines that control communication between the host and satellite computers.

3. Routines that do local jobs at the satellite, such as storing display files on the satellite's mass-storage devices.

Satellite Control and Dispatching

The Satellite Control Task (SATCTL.TSK) is produced during the execution of the GRSBLD command. SATCTL is built from the Satellite Dispatcher (SATDSP), the User's Satellite Routine (USRSAT), and the satellite library (GLIBS.OLB).

SATDSP and USRSAT are both created by COND when you build the satellite-end library. The contents of USRSAT as created by COND are short enough to state here:

```
SUBROUTINE USRSAT(B,I)
LOGICAL*1 B(60)
RETURN
END
```

In its initial form, USRSAT simply returns when called.

In host-satellite graphic systems, the User's Satellite Routine (USRSAT) can be modified to provide such features as local attention handling at the satellite. Appendix D has a pair of programs, DRAWH and DRAWS, that demonstrate one method of distributed processing in an interactive graphic program, wherein USRSAT is modified to process all graphic attentions at the satellite level rather than relying on the host computer.

The Satellite Dispatcher, SATDSP, interprets messages coming from the host to determine whether they are subroutine calls. If they are, SATDSP transfers control to the appropriate DECgraphic-11 subroutine in GLIBS.

Communication between Host and Satellite

The DECgraphic-11 FORTRAN Graphics Package includes several subroutines that allow the host and satellite computers to cooperate in producing a graphic display on the satellite display screen. The communication involves, for the most part, messages from the host that tell the satellite to execute a particular subroutine from the satellite library. In this manner, a DECgraphic-11 program can run in the host, but its subroutine calls are executed at the satellite display. In general terms, the following dialog takes place for a subroutine call (VECT is used as an example):

1. The host program uses the subroutine TOSAT to send the satellite a message requesting the VECT subroutine. TOSAT has the following form:

```
      CALL TOSAT(vcode,2,4,x,4,y,NOPTS(2),
     x 2,1,0,2,i,0,2,f,0,2,t,0)
```

   which corresponds to the statement CALL VECT(x,y,1,i,f,t). The vcode is a specific code that tells the satellite to call the VECT subroutine from GLIBS. The rest of the TOSAT arguments pass arguments to the VECT call that will be put together in the satellite. The argument after vcode tells TOSAT that two argument pairs follow. These two arguments for the example (VECT) are the mandatory x and y arguments. The two pairs give (a) the argument length in bytes, i.e., 4; and (b) the argument names (x and y). NOPTS(2) is a reference to the DECgraphic-11 statement function NOPTS, which returns the number of optional arguments supplied in the VECT call when two mandatory arguments are supplied. NOPTS will be 4 in this case, since all four status parameters (1, i, f, and t) are included in the VECT call. This value of NOPTS will tell the satellite to expect four data triplets, which are the last 12 arguments of the TOSAT call. The first element in a triplet is the length of the argument in bytes (2 for these integer arguments). The second part of the triplet is the actual value (one of the legal status parameters in this case). The last element of the triplet is the default value for each argument (0 for the status parameters).

2. The TOSAT call has a corresponding FRHOST call in the satellite. FRHOST has the form

```
      CALL FRHOST(B,I)
```

   where B is the 60-element message buffer defined by SATDSP, and I is an integer flag for messages (0=No message, 1=Message). The FRHOST call is located in the Satellite Dispatcher (SATDSP), which is built into the Satellite Control Task. SATDSP interprets the message by reading vcode, calling VECT from the satellite library (GLIBS.OLB), and giving VECT the same arguments that were supplied in the TOSAT call.

3. If the subroutine being executed is supposed to return values to the calling program, a similar pair of subroutines, FRSAT and TOHOST, is executed.

   FRSAT has the form

```
      CALL FRSAT(numsnd,ibytes,
     x destination argument,...)
```

FRSAT requests information from the satellite, where numsnd is the number of values to supply, and ibytes and destination argument (which are repeated as required in the FRSAT call) give the number of bytes to supply for each argument and the name of each argument for which values are being requested from the satellite. The scaling factors in a WINDW call are examples of data that would be transmitted from satellite to host in this manner. If the host does not receive an immediate response to an FRSAT call, it tries 10 times to read a message ibytes long from the satellite, with a delay of 10 seconds between tries.

4. For each FRSAT call in the host program, there is a TOHOST call in the satellite. TOHOST has the form

```
CALL TOHOST(ibytes, source argument,...)
```

TOHOST only occurs in response to an FRSAT call.

5. If an FRSAT call occurs in the host before the satellite is ready to transmit the requested data, the Satellite Control Task substitutes a DHOST call (for "Delay Host") for the TOHOST. DHOST has the form

```
CALL DHOST(ibytes)
```

When the host receives a DHOST message, it waits indefinitely until it receives a TOHOST with the proper message length.

NOTE

If a failure occurs in the data link or in the satellite computer while the host is delayed by a DHOST, the system will hang because the host will continue to wait for the message signaled by DHOST.

Local Storage at the Satellite

The DECgraphic-11 FORTRAN Graphics Package allows you to store and retrieve display files on your satellite's storage devices (if any). To do so, two supplementary subroutines are called by the SAVE and RSTR subroutines described in Chapter 2. These two extra subroutines are ACCESS and READWR.

ACCESS and READWR are designed to manipulate files on RT-11 floppy disks, such as are found with some GT62 satellite terminals. ACCESS and READWR are used together to read or write files on the floppy disks. To delete or rename files, ACCESS can be used by itself. The SAVE and RSTR subroutines call ACCESS and READWR automatically when they are called from your program. SAVE and RSTR operate only on display files; for more general file transfers, you can call ACCESS and READWR directly.

ACCESS has the form

```
CALL ACCESS(imode,filnam,ichan,IORET,isize)
```

where:

imode is the access mode (1 to 4);

filnam is the name of the file to look up, in a legal FORTRAN alphanumeric format (usually the name is enclosed in single quotation marks);

ichan is a channel number from 1 to 8 (not one of the reserved logical unit numbers listed in Section 5.3.4);

IORET returns a condition code that tells whether the ACCESS call was successful (IORET > 0) and gives the cause of failure if the call failed (IORET < 0);

isize is the number of 256-word blocks to allocate on the floppy disk when a new file is being created.

ACCESS can either look up (imode=1), create (mode=2), delete (imode=3), or rename (imode=4) a file on the floppy disk. The isize argument is required only when you are creating a new file name.

When you are renaming a file with ACCESS, put the old file name in place of the ichan argument.

The create and rename operations will both fail if the file name you specify is in use. The failure will be noted by IORET. The specific values for IORET, and their meanings, are as follows:

Value	I/O Status Return (IORET)
+1	Operation successfully completed
-1	I/O error on file
-2	Bad file-name string (e.g., file name too long)
-3	Error reading directory (e.g., disk not in RT-11 format)
-4	Error writing directory
-5	File name in use
-6	File not found
-7	Device full
-8	Directory full (one directory segment for each volume, not expandable)
-9	Bad directory
-10	End of file encountered

READWR has the form

```
CALL READWR(ioper,ichan,iblock,nwords,ibuf,IORET)
```

where:

ioper indicates that the call will read (ioper=0) or write (ioper=1) a file;

ichan (1 to 8) is the same channel number used in the corresponding ACCESS call;

iblock is the number of 256-word blocks in the file;

nwords is the number of words in the file;

ibuf is the name of the buffer for the file;

IORET is the status return, with the values having the same meaning as for the ACCESS subroutine.

The DRAWS program (a modified USRSAT) in Appendix D shows ACCESS and READWR being used for a variety of file transfers.

CAUTION

If you answer Y to the question LOCAL SAVE/RSTR asked by COND and you do not modify USRSAT to contain the SAVE and RSTR calls (in other words, if they are processed on the host end), the two calls will perform the file storage or retrieval on the host's storage device.

5.3.3 Running Host-Satellite Graphic Tasks

In the process of creating your host-end graphic library, GRGEN also creates a task called LGR (the Graphic Loader). LGR, when run from your satellite terminal, searches the mass-storage device of the host computer for a task image that you request. The task image is then "loaded" down the serial data link from the host computer to the satellite computer, where it is executed.

Because of its general purpose, you can use LGR to "down-line load" nearly any executable task, including graphic programs written with the DECgraphic-11 FORTRAN Graphics Package.

NOTE

Some FORTRAN statements are illegal in satellite programs; see Section 5.3.4.

As long as the satellite CPU has enough memory to hold the task image and the display file it creates, any functioning DECgraphic-11 program can be run at the satellite by typing the following command from the satellite terminal in response to the operating system's prompt symbol:

LGR task (RET)

where task is the file name of the task image you want to run.

However, LGR is not intended to be used in such an elementary way because the satellite processor often lacks enough free memory to run your task. In general, LGR is used to run the task SATCTL (the Satellite Control Task) in the satellite CPU. SATCTL is created by the GRSBLD command described earlier in this chapter. When you type the command

LGR SATCTL (RET)

SATCTL is down-line loaded and becomes a kind of "mini-executive" task for the entire satellite system. As noted in Section 5.2.2, the appearance of the screen cursor will change to show that SATCTL is running the satellite system.

When you type

RUN task (RET)

while SATCTL is running the satellite system, the "task" (graphic program) is executed not in the satellite, but in the host. However, when graphic subroutines in the host program create display instructions for the VS60 or VT11, the display instructions themselves are sent down the data link and intercepted by SATCTL. SATCTL then refers to its own subroutine library (GLIBS.OLB) and creates a display file in the satellite CPU.

The standard form of SATCTL, the one created during the library building process, is already capable of serving as the executive for the satellite, in the mode described in the previous paragraph. However, you can, in some cases, make your graphic program even more functional by modifying the User's Satellite Routine (USRSAT), which is one of the components that is built into the task image of SATCTL. The simple form of USRSAT that is created by COND can be considerably expanded to "redistribute" the various processor functions between the satellite and host CPUs. For instance, the program DRAWS, shown in Appendix D, is a modified form of USRSAT that allows the satellite processor to handle graphic attentions from the light pen and pushbutton box, rather than forcing the host to assume the burden. Local attention handling of this type is valuable in interactive graphic programs, because a busy host computer will not be slowed down by the necessarily large number of calls to attention-processing subroutines like LPEN, GRATTN, and PBH. Because the communication required on the data link is reduced when graphic attentions are handled by the satellite, the program will respond more rapidly to the user's commands.

5.3.4 Special Precautions for Host-Satellite Programming

A variety of suggestions have been made in previous sections for increasing the efficiency of a host-satellite program. See, for example, the description of the subroutine DISPLY (Section 2.12) and the discussion of local attention handling (Sections 5.3.2 and 5.3.3).

There are also some FORTRAN statements that cannot be used in satellite graphic programs. In general, they are the group of statements that read and write data (and entire files) to the terminal and to mass-storage devices. It is for this reason that the subroutines KBS and TTW are available for communication between the satellite keyboard and the satellite screen, and ACCESS and READWR (see Section 5.3.2) are available for manipulating files on the satellite's mass-storage devices. Examples of FORTRAN statements that are illegal in satellite programs are READ, WRITE, CALL ASSIGN, ENCODE, DECODE, and DEFINE FILE.

The following logical unit numbers (LUNs) are reserved for internal use by the DECgraphic-11 subroutines. Do not attempt to use them for other purposes.

Logical Unit 1: Display Scope (stand-alone RSX-11M only)

Logical Unit 2: Used by SAVE and RSTR

Logical Unit 3: Pushbutton box (stand-alone RSX-11M only)

Logical Unit 5: Terminal interface (TI:)

Similarly, the following event flag numbers are reserved by the subroutines:

Event Flags 9 and 10:	Terminal communication
Event Flag 11:	Light-pen interrupt (stand-alone RSX-11M only)
Event Flag 12:	Pushbutton interrupt (stand-alone RSX-11M only)
Event Flags 17, 18, and 19:	Host-satellite communications

In addition, be sure that you understand the functions of the special control characters that can be typed on the satellite keyboard. They are described in the next section.

5.3.5 Special Uses of Satellite Keyboard Characters

If you run a host-satellite program, you may notice that the satellite terminal "beeps" if you try to type characters on the satellite keyboard. This signal exists to warn you that you are trying to send a message while communication is underway between the host and satellite (including when the host has been delayed by a DHOST call from the satellite).

However, you will also want to use the keyboard at times to send messages to the satellite. For example, the DRAWH/DRAWS program shown in Appendix D has a pair of menu items named SAVE and RSTR for writing and reading display files from the satellite storage devices. When you select one of these options, the message FILE NAME: will appear on the screen, and the satellite program will wait for you to type the name of the file. Clearly, there has to be a method of distinguishing messages of this type, which are intended to communicate with the satellite, from messages intended for the host. The character

CTRL/H

is used for this purpose. (This CTRL/H character is made by holding down the CTRL key while you press the H key.) When you type a CTRL/H, the appearance of the satellite cursor will change. Instead of appearing as a square with a flashing center, the cursor will change to the outline of the square only. All messages you type while this cursor is on the screen will be directed to the satellite. To talk to the host again, type another CTRL/H; the cursor will change back to its regular appearance.

You can abort a graphic display on the satellite screen by typing the character

(CTRL/C)

to the satellite. In other words, to abort a graphic program, type CTRL/H, then CTRL/C to abort the task.

If you also want to abort SATCTL and use your graphic subsystem as a simple remote terminal, type

(CTRL/B)

to the satellite. In other words, type CTRL/H, then CTRL/B. (You may also have to follow with a CTRL/C or CTRL/Z to make the prompt symbol for the host operating system reappear.)

APPENDIX A

DECGRAPHIC-11 SUBROUTINE SUMMARY

This appendix is an alphabetical summary of the graphic subroutines in the DECgraphic-11 FORTRAN Graphics Package.

NOTE

The subroutines FRSAT, TOSAT, FRHOST, TOHOST, DHOST, ACCESS, and READWR, which are specific to host-satellite systems, are not covered in this appendix. See Section 5.3.2.

An asterisk (*) before a subroutine name means that the subroutine functions differently on VT11- and VS60-based subsystems. Usually, the difference is that only one scope is supported on the VT11; the s parameter is ignored in VT11 calls.

A cross (+) identifies subroutines that work only on VS60 graphic subsystems. Calls to these subroutines are legal in programs written for VT11 subsystems, but they will not perform any operation.

An explanation of the l, i, f, and t parameters, shown as optional arguments in many of these calls, is included at the end of this appendix.

DECGRAPHIC-11 SUBROUTINE SUMMARY

Call	Argument List	Effect
ADVANC	(k[,n])	Advances Pointer k (1-20) by n primitives from its current position; n must be positive; if n is omitted, the pointer advances to the next primitive in the display file. (See Section 2.6.2.)
APNT	(x,y[,l,i,f,t])	Positions the beam at the absolute position represented by (x,y) and can display a dot at that position; optionally changes l, i, f, and t parameters. (See Section 2.3.1.)
+ AREA	(n)	Specifies that subsequent graphic calls refer to the main viewing area (n=1) or the menu area (n=2). (See Section 2.2.2.)
* ATTACH	(k[,n,s])	Attaches the primitive identified by Pointer k (1-20) to the tracking object on Scope s. When the primitive is a long vector, it will be attached to the object and will follow it; if n is positive or omitted, the vector's origin is stationary and the destination end will move; however, if n is negative, the destination end is stationary and the origin will move. If the primitive is an absolute point or an absolute vector, you do not have to include n. (See Section 2.8.4.)
+ AVECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the absolute point that is represented by (x,y); optionally changes the l, i, f, and t parameters. (See Section 2.3.4.)
CHANGA	(k,x,y)	Changes the coordinates of the primitive identified by Pointer k (1-20) to the new value specified in (x,y). Like CHANGE, this subroutine can be used for primitives defined by the subroutines AVECT, VECT, SVECT, LVECT, RPNT, APNT, XGRA, and YGRA; but for VECT, SVECT, LVECT, and RPNT, subsequent primitives are adjusted so that they will appear at the same absolute screen positions as before the CHANGA call. (See Section 2.6.5.)
CHANGE	(k,x,y)	Changes the coordinates of the primitive identified by Pointer k (1-20) to the new values specified in (x,y). This subroutine can be used for primitives defined by the subroutines AVECT, VECT, SVECT, LVECT, RPNT, APNT, XGRA, and YGRA. (See Section 2.6.4.)

Call	Argument List	Effect
* CHANGT	(k,'a1'[,'a2',...])	Changes the argument string of the TEXT call identified by Pointer k (1-20) to the new string supplied in the CHANGT call. Characters in the CHANGT string can include control codes for italics, rotated characters, and superscripts or subscripts. (See Section 2.6.6.)
CMPRS		Compresses the display file by removing all erased primitives and subpictures, and reclaiming the space used by them. (See Section 2.13.1.)
CONT		Restarts the display processor, thus restoring the display interrupted by a call to STOP. (See Section 2.1.3.)
COPY	([m1],m2)	Creates a copy of the existing subpicture whose tag is m2 and assigns it the tag m1; if the m1 parameter is omitted, Subpicture m2 is copied to the currently open subpicture. (See Section 2.4.3.)
+ CVSCAL	(m[,ifc,ifv])	Scales the size of displayed characters and vectors in Subpicture m; ifc can be in range 1-4, where 1 is one-half normal character size, and 4 is twice normal size (normal is 2); ifv must be in the range 1-15, where 1 is one-fourth normal size and 15 is three and three-fourths normal size (normal is 4). (See Section 2.4.8.)
* DETACH	[(s)]	Detaches all primitives from the tracking object on Scope s. (See Section 2.8.5.)
DISPLY	(n)	Creates display files quickly by suppressing the display of inserted primitives; n=-1 suppresses the display; n=0 displays all primitives previously inserted, but still suppresses the display of subsequently inserted primitives; n=1 returns to normal display mode, in which the primitives are displayed as soon as they are inserted in the display file. (See Section 2.12.1.)
DPTR	(I)	Returns in I the subscript of the display-file element in which the primitive will be entered. (See Section 2.14.1.)
DPYNOP	(n)	Inserts n "no-operation" instructions at the end of the display file. (See Section 2.14.2.)

	Call	Argument List	Effect
	DPYWD	(iword,idisp)	Inserts the 16-bit data word (iword) at the end of the display file; displays the result of iword on the screen if the value of idisp is 0. (See Section 2.14.3.)
	ERAS	[(m)]	Erases the definition of Subpicture m from the display file; if parameter m is omitted, the tracking object is removed from the screen; the space occupied by the subpicture must be reclaimed by CMPRS or SAVE. (See Section 2.4.6.)
	ERASP	(k)	Erases the primitive identified by Pointer k (1-20), and repositions the pointer at the next primitive in the display file. (See Section 2.6.8.)
*	ESUB	[(m)]	Terminates the definition of a subpicture; if the m argument is included in the call in a VS60 program, the display parameters (e.g., light-pen enable, intensity) that were in effect before the current subpicture was defined are restored; the m argument has no effect in VT11 programs. (See Section 2.4.2.)
	FIGR	(ARRAY,n,m[,l,i,f,t])	Creates a special figure subpicture with tag m. The figure is plotted with long vectors whose displacements are the first n (x,y) pairs specified in ARRAY. Optionally changes the l, i, f, and t parameters. (See Section 2.5.3.)
	FLASH	(k[,f])	Enables (f>0) or disables (f<0) flash mode for the primitive identified by Pointer k. (See Section 2.7.3.)
	FREE		Disconnects the display file from the display processor, thus freeing the area of memory used by the file; this action terminates graphic processing until the next call to INIT. (See Section 2.1.4.)
	GET	(k,X,Y)	Returns in (X,Y) the coordinate positions of the primitive identified by Pointer k. (See Section 2.6.3.)
	GRATTN	(iwait,IRETRN,idev1[,idev2,idev3,...,idevn])	Polls the interactive devices idev1...idevn for graphic attentions; waits for an attention if iwait=1; returns after one poll if iwait=0; returns in IRETRN an index to the list of interactive devices if an attention occurs, where IRETRN=1 if the attention came from idev1, etc.; the values for

Call	Argument List	Effect
		idevn are 1=light pen, 2=pushbutton, and 3=keyboard. GRATTN must be followed with an attention-processing routine, such as LPEN, PBH, or KBC. (See Section 2.9.)
* GRID	(gx,gy[,s])	Moves the tracking object on Scope s to the nearest point on the grid and automatically detaches any detached primitives. Arguments gx and gy define the spacing of points on the grid. (See Section 2.8.6.)
INIT	[(n)]	Clears the display screen, initializes the display file to use the first n words of the COMMON area, DFILE, and sets the initial display parameters. (See Section 2.1.1.)
INSRT	[(k)]	Reopens a subpicture for insertion of primitives (specified in subsequent graphic subroutine calls). Insertion begins just before the primitive identified by Pointer k. If k is omitted, the insert operation is terminated. (See Section 2.6.7.)
INTENS	(k[,i])	Changes the intensity level of the primitive identified by Pointer k to the new value specified in i (1-8) and/or makes the referenced primitive visible. (See Section 2.7.2.)
KBC	(ICHAR)	Reads a single character from the keyboard; returns the ASCII value of the character in ICHAR; does not wait for a carriage return before reading the character (except in RT-11), and thus allows easy use of the keyboard characters as function keys in interactive programs. (See Section 2.11.1.)
KBS	(length,STRING[,NUMBER])	Reads a character string from the keyboard (waits for carriage return in all operating systems); length is the number of characters to read; STRING is a LOGICAL*1 array to return ASCII values of characters; and NUMBER optionally returns the number of characters actually read. (See Section 2.11.2.)
LINTYP	(k[,t])	If t is a legal line-type value (1-4), changes the line type of the primitive identified by Pointer k to the type specified in t. (See Section 2.7.4.)

Call	Argument List	Effect
* LPEN	(IH,IT[,X,Y,IP,IA,IM,IT1,IT2])	Tells whether a light-pen hit has taken place, and returns information about the hit in the following variables. (See Section 2.8.1.)
	IH	Nonzero if light-pen hit has occurred (1 for Scope 1, 2 for Scope 2); always 0 or 1 for a VT11.
	IT	Tag of the subpicture in which the hit occurred.
	X,Y	Coordinates of the hit.
	IP	Number of the primitive within the subpicture at which the hit occurred; IP is undefined if the primitive is not in a subpicture.
	IA	Array that stores the precedents of subpicture IT (subpicture tags are in order, starting with the innermost-nested subpicture).
	IM	Screen area of the light-pen hit (1 for main area, 2 for menu area); always 1 for a VT11.
	IT1,IT2	Status of the light-pen tip switches for Scopes 1 (IT1) and 2 (IT2) -- 0 for off, 1 for on; for a VT11, IT1 is always 1 and IT2 is always 0.
LVECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the relative position represented by (x,y), using the long-vector format; optionally changes l, i, f, and t parameters. (See Section 2.3.6.)
MENU	([x0],y0,dy,m,'a1'[,'a2',...,'a10'])	Displays a list of up to 10 light-pen-sensitive items to be used as a menu. The items are the character strings 'a1','a2', and so on, and the first item is displayed at the coordinate position represented by (x0,y0); if x0 is omitted from a VS60 call, the items are displayed in the hardware menu area of the screen; on the VT11, they begin at the right edge of the viewing area. The dy argument is the vertical spacing between menu items. The m argument is the subpicture tag assigned to the first menu item ('a1'); subsequent menu items are tagged

Call	Argument List	Effect
		sequentially, so that if there are 10 menu items, the 10th item's tag is m+10-1 or m+9. (See Section 2.3.8.)
NMBR	(m,var[,n,'format'])	Creates a special numeric subpicture whose tag is m. This call formats the numeric contents of a FORTRAN variable, var, with a width of n in the specified format. If the format parameter is omitted, the default format is F16.8 for subroutine libraries that take both real and integer arguments, and I6 for libraries that accept only integers. (See Section 2.4.7.)
NOWNDW		Restores the default window (x0=0., y0=0.) to (x1=1023., y1=1023.) for subsequent graphic subroutine calls, and eliminates any user-defined window. (See Section 2.2.4.)
OFF	(m)	Temporarily turns off the subpicture whose tag is m. (See Section 2.4.4.)
ON	(m)	Turns on Subpicture m. (See Section 2.4.5.)
PBH	(IHIT,PUSHED)	Looks for and identifies graphic attentions from the LK-11 pushbuttons; IHIT returns a nonzero value if any button was pushed since the last PBH call; PUSHED is a 16-element LOGICAL*1 array that identifies the hit button, where PUSHED(N)=.TRUE. if Button n was pushed. (See Section 2.10.2.)
PBL	[(DARK,LIT)]	Turns specific button lights on or off on the LK-11 pushbutton box; DARK and LIT are 16-element LOGICAL*1 arrays, where a .TRUE. value in an element turns the corresponding button off or on, respectively; if you call PBL with no arguments, the button lights return to automatic control. (See Section 2.10.3.)
PBS	(STATE)	Reports the .TRUE./.FALSE. status of the LK-11 buttons in the 16-element LOGICAL*1 array STATE. If the button lights are under automatic control, a .TRUE. value in STATE corresponds to a lit button. (See Section 2.10.1.)
POINTR	(k,m[,j])	Points Pointer k at the jth primitive of Subpicture m; if j is omitted, the pointer points to the first primitive of the subpicture. The k argument must be in the range 1-20. (See Section 2.6.1.)

	Call	Argument List	Effect
	RPNT	(x,y[,l,i,f,t])	Moves the beam from its current position to the relative position represented by (x,y), and can display a point at that position. If the current position is (10,20) the beam is moved to (10+x,20+y). Optionally changes l, i, f, and t parameters. (See Section 2.3.2.)
	RSTR	('[dev:] file descriptor')	Restores in the memory area of the current display file the display file stored in the specified mass-storage file. (See Section 2.13.3.)
	SAVE	('[dev:] file descriptor')	Saves the display file by writing it onto the mass-storage file identified in the call. (See Section 2.13.2.)
+	SCOPE	(n)	Specifies that subsequent graphic calls refer to Scope 1 (n=1) or 2 (n=2). (See Section 2.2.1.)
*	SENSE	(k[,l,s])	Enables (l>0) or disables (l<0) light-pen sensitivity for the primitive identified by Pointer k for Scope s (1 or 2). (See Section 2.7.1.)
	STOP		Halts the display processor, stops the display, and clears the display screen. The display can be restored by calling CONT. (See Section 2.1.2.)
	SUBP	(m1[,m2])	Begins the definition of a subpicture whose tag is m1; all primitives specified in subsequent graphic subroutine calls are part of the subpicture, until the occurrence of a terminating call to ESUB. If the m2 parameter is included in the call, a new subpicture tag, m1, repeats an existing subpicture whose tag is m2. (See Section 2.4.1.)
	SVECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the relative position represented by (x,y), using the short-vector format; optionally changes l, i, f, and t parameters. (See Section 2.3.5.)
*	TEXT	([ictrl,]'a1'[[,ictrl],'a2',...])	Displays the character strings supplied in the call ('a1', etc.), beginning at the current beam position. Characters in the call can include control codes for italics, rotated characters (VS60), and superscripts or subscripts. (VS60.) (See Section 2.3.7.)

	Call	Argument List	Effect
*	TRAK	(x,y[,s])	Positions a tracking object on the screen of Scope s at coordinate position (x,y) and centers the object on any light-pen hit within the tracking object area. (See Section 2.8.2.)
*	TRAKXY	(X,Y[,s])	Returns the coordinates of the current position of the tracking object in (X,Y) for Scope s. (See Section 2.8.3.)
	TTW	(number,ICHARS[,number,ICHARS,...])	Writes a character string on the screen; ICHARS is a LOGICAL*1 array containing the characters; the argument number tells how many characters to display; both arguments can be repeated indefinitely. (See Section 2.11.3.)
	VECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the relative position represented by (x,y), using the short-vector format when possible; optionally changes l, i, f, and t parameters. (See Section 2.3.3.)
+	VIEWPT	(x,y)	Defines a viewport on the image definition area of the VS60 by specifying the coordinate positions (x,y) of the lower left corner of the viewport. (See Section 2.2.5.)
	WINDW	(x0,y0,xl,yl[,FX[,FY]])	Defines a new window in which (x0,y0) identifies the lower left corner of the screen, and (xl,yl) identifies the upper right corner of the screen; optionally returns the x and y scaling factors in FX and FY. (See Section 2.2.3.)
	XGRA	(dy,ARRAY,n,m[,l,i,f,t])	Creates a special Graph Subpicture m, which consists of a series of points. The x values of the points to be plotted are the first n elements of ARRAY, and the y values are given by integral multiples of dy. Optionally changes l, i, f, and t parameters. (See Section 2.5.1.)
	YGRA	(dx,ARRAY,n,m[,l,i,f,t])	Creates a special Graph Subpicture m, which consists of a series of points. The y values of the points to be plotted are the first n elements of ARRAY, and the x values are given by integral multiples of dx. Optionally changes l, i, f, and t parameters. (See Section 2.5.2.)

DISPLAY PARAMETERS

Parameter	Meaning
l	Light-pen control Default: disabled l=positive. Light-pen interaction is enabled, and a light-pen attention will occur when the light pen is pointed at a sensitive object on the display screen. l=0 or omitted. The value of the parameter does not change from its previous status. l=negative. Light-pen interaction is disabled; light-pen attentions will not occur.
i	Intensity level Default: 4; with normal adjustment of the BRIGHTNESS or INTENSITY knob, this value makes the primitive bright enough for the light pen to detect. i=1 through 8. The current and subsequent primitives are visible, with intensity level i, where 1 is the faintest intensity and 8 is the brightest. i=0, omitted or greater than 8. The current primitive becomes visible, but the intensity level does not change. i=-8 through -1. The current primitive (except for characters) is invisible, but the intensity level of subsequent primitives is the absolute value of i. i=less than -8. The current primitive is invisible, and the intensity level is unchanged for subsequent primitives.
f	Flash mode Default: off f=positive. Current and subsequent primitives will flash on and off. f=0 or omitted. The value of the parameter does not change from its previous status. f=negative. Flash mode is disabled for current and subsequent primitives.
t	Line type Default: solid (1) t=1. Current and subsequent vectors are displayed as solid lines. t=2. Vectors are displayed as long-dash lines. t=3. Vectors are displayed as short-dash lines. t=4. Vectors are displayed as dot-dash lines. t=0, omitted, negative, or greater than 4. The value of the parameter does not change from its previous status.

APPENDIX B

DECGRAPHIC-11 ERROR MESSAGES

This appendix summarizes all error messages that are generated by the DECgraphic-11 FORTRAN Graphics Package. For each DECgraphic-11 message, the appendix provides the message number (where appropriate), briefly explains the reason for its occurrence, and lists the subroutines in which the error might occur. The appendix also explains messages that relate specifically to host-satellite systems.

You can suppress the display of error-message text when you run the COND program to create new libraries. When you are building your subroutine library (see Chapters 4 and 5), answer N to the question:

```
ERROR MESSAGE TEXT (Y OR N) ?
```

If you have suppressed text output and an error occurs, only the number of the error message will be displayed, as shown below:

```
ERROR #YY XXXXXXXXXX
```

where YY is the 2-digit error-message number (see the list given on the following pages), and the 10-digit X field represents the argument (for example, the subpicture tag) in certain messages. Errors marked HOST-SATELLITE ONLY do not display any text.

In stand-alone systems, error messages specific to DECgraphic-11 are returned using standard FORTRAN subroutine traceback logic. Messages are written in the following form:

```
IN ROUTINE "XXXXXX" LINE YYYY
FROM ROUTINE "XXXXXX" LINE YYYY
     .
     .
     .
```

A message of this kind will be preceded by a false error message (Number 61):

```
ILLEGAL MEMORY REFERENCE
```

Disregard this message.

In host-satellite systems, error messages specific to DECgraphic-11 are also returned using standard FORTRAN subroutine traceback logic in the form shown above. However, in host-satellite systems the ILLEGAL MEMORY REFERENCE line is not written.

Because many DECgraphic-11 subroutines operate internally by calling other subroutines, some traceback information (in either a stand-alone or host-satellite system) will concern subroutine calls that you have not made. Note only the references to your subroutine calls.

DECGRAPHIC-11 ERROR MESSAGES

Number	Error Message	Subroutine(s)
1	UNABLE TO LINK TO SCOPE You do not have a scope, or it is being used by another task. If you are using RSX-11M, be sure that the option statement ASG=GR0:1 was included in the Task Builder commands (see Section 5.2.1).	INIT
2	INIT NOT CALLED You have forgotten to call the INIT subroutine to initialize your display file (see Section 2.1.1).	All subroutines
3	MORE ESUB'S THAN SUBP'S There are more calls to ESUB in your program than corresponding SUBP calls (see Sections 2.4.1 and 2.4.2).	ESUB
4	[SATELLITE I/O ERROR] HOST-SATELLITE ONLY An error has occurred in input or output to the satellite's storage devices.	SAVE, RSTR, ACCESS, READWR
5	[CHANGT TOO LONG] HOST-SATELLITE ONLY You have given an argument string in the subroutine CHANGT that is too long to fit in the host's communication buffer (see Chapter 5).	CHANGT
6	SYSTEM FAULT An internal software error has occurred; report to DIGITAL with Software Performance Report.	All subroutines
7	[RESERVED FOR FUTURE USE]	
8	DISPLAY FILE FULL There is not enough space left in the display file for any more primitives. You may be able to correct the condition with the CMPRS subroutine (see Section 2.13.1).	All subroutines
9	FILE TOO BIG The display file being restored from a mass-storage device is too big to fit in the present display file.	RSTR
10	XXXXX IS IN USE The tag XXXXX is already assigned to a subpicture.	SUBP, COPY, XGRA, YGRA, FIGR, NMBR, MENU

Number	Error Message	Subroutine(s)
11	MORE THAN 8 NESTED SUBP You have specified more than eight nested calls to SUBP.	All subpicture subroutines
12	XXXXX IS NOT A SUBPICTURE The tag XXXXX is not defined as a subpicture.	SUBP, COPY
13	XXXXX IS STILL OPEN You have tried to CALL SUBP (m1,m2) for a subpicture that has not been "closed" by an ESUB call.	SUBP, COPY
14	ILLEGAL INCREMENT The dx or dy value specified in the call is less than 1 or exceeds 63 raster units.	XGRA, YGRA
15	XXXXX ILLEGAL/UNDEF POINTER The value of Pointer XXXXX is not in the range 1 through 20, or a referenced pointer has not been defined with a call to POINTR.	All subroutines that use pointers
16	ILLEGAL DURING INSRT You have attempted an illegal operation while in insert mode.	CMPRS, SAVE, RSTR, SUBP, FLASH, SENSE, INTENS, CVSCAL, ESUB, LINTYP, INSRT
17	TOO FEW ARGUMENTS You have omitted one or more required arguments from a subroutine call.	All subroutines
18	CAN'T COPY NESTED SUBPS You have tried to copy a subpicture that contains another subpicture.	COPY
19	VERSION MISMATCH You have mismatched versions of the components in the DECgraphic-11 FORTRAN Graphics Package kit, or you have tried to restore a VT11 display file on a VS60 screen.	INIT, SAVE, RSTR

The error messages explained below relate specifically to host-satellite systems.

? ERROR XX

In host-satellite configurations, this error message indicates a FORTRAN error at the host end. The XX refers to the number of a standard FORTRAN error. Refer to the appropriate FORTRAN reference manual.

? SATELLITE ERROR XXX

Unless XXX is one of the numbers given in the following two paragraphs, a FORTRAN error has occurred at the satellite end. The XXX refers to the number of a standard FORTRAN error. Refer to the appropriate FORTRAN reference manual.

If XXX is 63, an unrecognized interrupt has occurred. This message usually indicates a hardware problem. Contact your DIGITAL Field Service Representative.

If XXX is in the range 200 through 210, the error is caused by an internal software problem (not a FORTRAN problem). Report to DIGITAL with a Software Performance Report.

COMMUNICATIONS FAILURE

This message, which is usually preceded by random information ("garbage"), indicates a communications problem (often a line failure or other hardware problem) between host and satellite. Try to run your program a few more times. If failure continues, contact your DIGITAL Field Service Representative.

APPENDIX C

DISPLAY-FILE STRUCTURE

This appendix describes the internal structure of the DECgraphic-11 display file as it is constructed in memory. Consult this appendix before attempting to alter a display file (with the DPYWD and DPYNOP subroutines) or before you change any of the source modules that make up the DECgraphic-11 FORTRAN Graphics Package.

1. Overall Structure

   a. Root portion

      This is controlled internally by the DECgraphic-11 Display File Handler, which also sets up initial display parameters.

   b. User Space

      Display data ("active display list"), set up by DECgraphic-11 in the COMMON area called DFILE that ends with the instruction DHALT, 0 (173400,0).

2. Subpictures

   a. A repetition of an existing subpicture is formatted in five words:

VT11: 173400 (DHALT)	VS60: 162000 (DJSR)
Address of rest of file	Address of subpicture
Address of subpicture	161002 (DJMPR)
tag	tag
link	link

   b. A subpicture header, which begins every subpicture definition, is also formatted in five words:

VT11: Same as for a repetition	VS60: 163004 (DJSRR)
	160000 (DJMPA)
	Address of rest of file
	tag
	link

   c. The end of a subpicture has the following format:

VT11: 173400 (DHALT)	VS60: 165000 for ESUB
0	(return, no restore)
	166000 for ESUB (n)
	(return, restore
	display parameters)

d. When subpictures are turned off, their formats are changed to the following:

Repeated subpictures:

VT11: 160000 (DJMPA) in first word	VS60: 161004 (DJMPR) in first word

Subpicture definitions:

VT11: 160000 (DJMPA) in first word	VS60: 161000 (DJMPR next word) in first word

e. When subpictures are erased, the first two words of the formats are changed to the following:

VT11 and VS60: 160000 (DJMPA)
Address of rest of buffer

The link is also removed from the tag list.

f. The pointer to the first tag is in ILST in COMMON GRDAT; the pointer to the last tag (the one that is 0) is in IEND.

3. Display stops are serviced by the Display File Handler.

4. Light-pen hits are serviced by the Display File Handler.

5. Display time-outs are serviced by the Display File Handler.

6. Calls to XGRA, YGRA, and FIGR:

a. XGRA and YGRA have the following format:

```
Five-word subpicture header
Load graphic increment
Enter graph mode
X or Y data (one word per point)
.
.
.
End of subpicture
```

b. FIGR has the following format:

```
Five-word subpicture header
164000 (DNOP)
LVECT
Data (two words per point)
.
.
.
End of subpicture
```

7. Calls to NMBR:

```
Five-word subpicture header
100001 (TEXT MODE)
Data (two characters for each word)
.
.
.
End of subpicture
```

8. File structure as a "saved file":

```
Total of n+3 words:
n+2=number of words to follow (first word)
n words of display data
.
.
.
173400,0
```

9. Display-file organization on sequential calls to the same module (vector/point/graph plot module):

The example provided below makes sequential calls to the module LVECT.

A single call to LVECT is normally mapped into three words in the display file.

```
Word 1:  mode (M)
Word 2:  x coordinate
Word 3:  y coordinate
```

But on a sequential call to LVECT such as:

```
.
.
.
CALL LVECT(X1, Y1)
CALL LVECT(X2,Y2)
CALL LVECT(X3,Y3)
.
.
.
```

the LVECT calls will be mapped into the display file as shown below:

```
.
.
.
M
X1
Y1
X2
Y2
X3
Y3
.
.
.
```

This result occurs because the display-parameter word (1,i,f,t) remains unaltered in the second and third call.

APPENDIX D

FORTRAN PROGRAMMING EXAMPLES

This appendix contains two forms of a figure-drawing program that uses the Decgraphic-11 FORTRAN Graphics Package. Both forms are included as FORTRAN sources in your DECgraphic-11 FORTRAN Graphics Package kit.

D.1 DRAW.FOR

The first form, DRAW.FOR, can be used in RT-11 and RSX-11M stand-alone systems. It can also be run on host-satellite systems, but will force the host computer to handle graphic attentions; see instead the DRAWH/DRAWS program described in the next two sections.

DRAW.FOR must be used with real-valued libraries. After you have built such a library (see Chapter 4 or Chapter 5), follow these procedures to prepare DRAW:

RT-11:

```
.FOR dev:DRAW (RET)
.
```

where dev: is the storage device (e.g., DK1:) that contains the source DRAW.FOR.

```
.LINK dev:DRAW,GLIB (RET)
.R DRAW (RET)
```

The DRAW program will run as described in Section D.4.

RSX-11M (mapped stand-alone system):

```
>FLX SY:/RS=dev:DRAW.FOR/os (RET)
>FOR DRAW=DRAW.FOR (RET)
>TKB (RET)
TKB>DRAW=DRAW (RET)
TKB>GLIB/LB (RET)
TKB>/ (RET)
ENTER OPTIONS:
TKB>ASG=GR0:1 (RET)
TKB>COMMON=DFILE (RET)
TKB>MAXBUF=512 (RET)
TKB>// (RET)
>RUN DRAW (RET)
```

Notice that the Task Builder commands use a global common for the display file (DFILE). For details on global commons (also called shared regions), which must be created at the time you generate your RSX-11M system, see the RSX-11M

System Generation Manual, the RSX-11M Task Builder Reference Manual, the IAS/RSX-11 FORTRAN IV User's Guide, and Section 1.2 of this manual.

In the FLX command, the letters dev: stand for the device that contains the source DRAW.FOR (an RK05 or RK06 disk in RT-11 format or a magnetic tape in DOS-11 format). The letters os represent the file format of the device, either RT or DO.

D.2 DRAWH.FTN (HOST-SATELLITE ONLY)

DRAWH is the host portion of the host-satellite drawing program. It must be compiled (in FORTRAN IV or IV-PLUS) and linked to the host library (GLIBH.OLB).

This version of DRAWH uses integer-only arguments, and therefore must be used with a GLIBH that was built with that feature; see Chapter 5. In the sample procedures, the FORTRAN IV-PLUS option is also required. To compile and link DRAWH, follow these procedures:

RSX-11M/RSX-11D:

```
>FLX SY:/RS=dev:*.FTN/os (RET)
```

[dev: is the device containing DRAWH.FTN; os is either RT or DO.]

```
>F4P USER.DRAWH=DRAWH (RET)
>TKB @GRHBLD (RET)
>PIP DRAW.TSK=USER.TSK/RE (RET)
```

Now build the satellite portion of the program; see Section D.3.

IAS:

```
PDS>MOUNT/FOREIGN/NOOPERATOR (RET)
DEVICE? dev: (RET)
VOLUME-ID? XXX (RET)
PDS>COPY (RET)
FROM?dev:*.FTN/os (RET)
TO? *.* (RET)
PDS>FORTRAN/OBJ:USER DRAWH (RET)
PDS>@GRHBLD (RET)
PDS>RENAME USER.TSK DRAW.TSK (RET)
```

In the COPY commands, dev: is the storage device containing the source files DRAWH.FTN and DRAWS.FTN (an RK05 or RK06 disk in RT-11 format or a magnetic tape in DOS-11 format). The letters os are either RT11 or DOS.

Now proceed to the next section and build the satellite portion of DRAW.

D.3 DRAWS.FTN (HOST-SATELLITE ONLY)

DRAWS is the satellite portion of the program, actually a modified version of USRSAT that handles graphic attentions from the light pen, pushbutton box, and satellite keyboard.

Therefore, you must create a new satellite control program (SATCTL), substituting DRAWS for USRSAT:

RSX-11:

```
>FOR USRSAT.DRAWS=DRAWS (RET)
>TKB @GRSBLD (RET)
```

IAS:

```
PDS>FORTRAN/OBJ:USRSAT DRAWS (RET)
PDS>@GRSBLD (RET)
```

When both DRAWH and DRAWS have been compiled and linked, use the Graphic Loader (LGR) to run SATCTL. After the satellite cursor appears (about one minute), you can run DRAW:

RSX-11M:

```
>RUN LGR (RET)
LGR>SATCTL (RET)
[Satellite cursor appears] RUN DRAW (RET)
```

RSX-11D:

```
MCR>RUN LGR (RET)
LGR>SATCTL (RET)
STARTING ADDRESS? 1000 (RET)
[Satellite cursor appears] RUN DRAW (RET)
```

IAS:

```
PDS>RUN LGR (RET)
LGR>SATCTL (RET)
STARTING ADDRESS> 1000 (RET)
[Satellite cursor appears] RUN DRAW (RET)
```

D.4 USING THE DRAW PROGRAM

The program DRAW.FOR uses the light pen to interact with a menu of graphic options. The options, and their functions, are as follows:

Menu Item	Function
DRAW	Draw lines to make a subpicture.
MOVE	Move a subpicture.
COMBINE	Combine subpictures into one.
SCALE	Change the size of a subpicture.
COPY	Copy a subpicture.
ERASE	Erase a subpicture.
MODIFY	Split a line; make a line visible or invisible; move a corner.
HIDE	Turn off a subpicture.
SEEK	Turn on all subpictures.
SEEKCOPY	Turn on and copy subpictures.
ROTATE	Rotate a subpicture on the screen.
SAVE	Save the display (i.e., the entire picture).
RECALL	Recall a display file from storage.
EXIT	Terminate the DRAW program.

Select a function by pointing to a menu item with the light pen. The selected item will brighten, and then the main menu will disappear, to be replaced with a "secondary" menu.

Secondary menus give you more detailed control of an individual function. For example, when you point to DRAW on the main menu, a secondary menu will appear with the options POSITION, LINE, CLOSE, and DONE, and the tracking object will appear in the middle of the screen. To see how this function works, touch the word LINE with the light pen. Then touch the tracking object and move the light pen, allowing the tracking object to follow it. A line appears, with its origin at the first position of the tracking object and its destination the present position. The line's destination will move with the tracking object, a process called "rubber-banding." This process is performed by attaching a long vector to the tracking object using the LVECT and ATTACH subroutines (see Sections 2.3.6 and 2.8.4).

The other secondary DRAW functions are POSITION, which moves the tracking object to a new starting position without drawing a line; CLOSE, which creates a closed figure by supplying a missing line (e.g., the third line of a triangle); and DONE, which returns to the main menu. You can also select LINE repeatedly, in which case the current line's destination will be forced to a grid point (see the GRID subroutine in Section 2.8.6) and this point will be the origin of the next line.

The DRAW and MODIFY menu items both have secondary menus that include DONE. The secondary menus will stay on the screen until you select DONE. In addition, after selecting a secondary item such as POSITION, you must perform the stated operation before you can select any new item from the secondary menu.

The other main-menu items usually switch to their secondary menu and then immediately back to the main menu after you request an operation (ROTATE is such an example).

The SAVE and RECALL items on the main menu store or retrieve the entire display file you have created, including subpictures that are turned off. Both items, when selected, will display the message FILENAME: on the screen; when this message appears, type the name (6 characters or less) by which the display file is to be stored or retrieved. SAVE and RECALL also automatically remember the subpicture tags used in a particular picture, so that they will not be duplicated accidentally.

The DRAWH/DRAWS program functions in the same way, with the additional features of pushbutton support and keyboard interaction. In other words, you can select the menu items with the light pen, pushbutton box, or satellite keyboard. On the keyboard, the 14 menu items are represented, top to bottom, by the letters A through N. The same menu items are also represented by the first 14 buttons on the LK-11 pushbutton box. Any time you select a menu item, the corresponding button will light briefly on the LK-11. The LK-11 is not a requirement for running DRAWH/DRAWS, but only an option.

You can learn more about DRAW and DECgraphic-11 by running the program and watching the DECgraphic-11 FORTRAN Graphics Package in operation.

D.5 PROGRAM LISTINGS

```
C
C       SIMPLE DRAWING PROGRAM USING THE NEW GRAPHICS PACKAGE
C
C               STAND ALONE VERSION
C               USES REAL MODE, LIGHT PEN ATTENTIONS ONLY
C
        COMMON/DFILE/IBUF(4096)
        COMMON/PDATA/NPRIM(209),NVPRIM(209),NDEF,NHID
        REAL SF(4)
        LOGICAL*1 FILE1(16),FILE2(16),USED(5)
        DATA SF/.25,.5,2.,4./
        DATA USED/48,0,0,37,0/
        SIZE=4096.
        GRD=50.
        SM=50.
        YTOP=750.
        IWARN=15
        MARGIN=10
        NDEF=0
        NHID=0
        ISAC=0
        DO 10 I=1,200
        NVPRIM(I)=0
10      NPRIM(I)=0
C
C       SET UP MENU AREAS
C
        CALL INIT(4096)
        CALL SUBP(1000)
        CALL OFF(1000)
        CALL SUBP(1001)
        CALL MENU(,YTOP,-SM,2010,'DRAW','MOVE','COMBINE','SCALE',
     X  'COPY','ERASE','MODIFY','HIDE','SEEK')
        CALL MENU(,YTOP-9.*SM,-SM,2019,'SEEK & COPY','ROTATE','SAVE',
     X  'RECALL','EXIT')
        CALL ESUB
        CALL SUBP(1002)
        CALL MENU(,300.,-SM,1020,'POSITION','LINE','CLOSE','DONE')
        CALL ESUB
        CALL MENU(,300.,0.,1003,'DONE')
        CALL SUBP(1004)
        CALL MENU(,300.,-SM,1030,'1/4','1/2','2 X','4 X')
        CALL ESUB
        CALL SUBP(1005)
        CALL MENU(,300.,-SM,1040,'ERASE LINE','SPLIT LINE',
     X  'MOVE CORNER','SHOW ALL','DONE')
        CALL ESUB
        CALL SUBP(1006)
        CALL MENU(,300.,-SM,1050,'90 CW','180','90 CCW')
        CALL ESUB
        CALL SUBP(2006)
        CALL AREA(2)
        CALL APNT(0.,0.,,-4)
        CALL TEXT('      ')
        CALL ESUB(2006)
C
C       MAIN LOOP -- WAIT FOR MENU HIT AND BRANCH TO SERVICE IT
C
100     DO 110 I=1002,1006
110     CALL OFF(I+0)
        CALL ON(1001)
```

```
      CALL ON(1000)
      CALL DPTR(I)
      CALL POINTR(2,2006,2)
      I=(SIZE-I)/SIZE*100.
      CALL FLASH(2,IWARN-I)
      USED(3)=I-I/10*10+48
      USED(2)=I/10+48
      CALL CHANGT(2,USED)
      CALL MENUH(IT,2010,2023)
      CALL OFF(1001)
      GOTO (1100,1600,1700,1800,1900,2000,2100,2200,2300,2600,2700,
     X  2400,2500,5000),IT
C
C     DRAW A NEW OBJECT
C
1100  IF(I.LT.MARGIN)GOTO 4000
      CALL ON(1002)
      CALL MAKOBJ(NOBJ)
      CALL SUBP(NOBJ)
      CALL APNT(500.,500.,1,-4)
      CALL POINTR(2,NOBJ)
      XX=500.
      YY=500.
1110  CALL ATTACH(2)
      CALL TRAK(XX,YY)
      CALL MENUH(IT,1020,1023)
      CALL GRID(GRD,GRD)
      CALL TRAKXY(XX,YY)
      CALL ERAS
      CALL GET(2,X,Y)
      IF(ABS(X).LT.GRD.AND.ABS(Y).LT.GRD.AND.NPRIM(NOBJ).NE.0)
     X  GOTO 1110
      II=-4
      GOTO(1140,1120,1160,1180),IT
1120  II=4
      NVPRIM(NOBJ)=NVPRIM(NOBJ)+1
1140  NPRIM(NOBJ)=NPRIM(NOBJ)+1
      CALL LVECT(0.,0.,,II)
      CALL ADVANC(2)
      GOTO 1110
1160  II=4
1180  CALL POINTR(2,NOBJ)
      CALL GET(2,X0,Y0)
      X=X0-XX
      Y=Y0-YY
      IF(ABS(X).LT.GRD.AND.ABS(Y).LT.GRD)GOTO 1185
      CALL LVECT(X,Y,,II)
      IF(II.GT.0)NVPRIM(NOBJ)=NVPRIM(NOBJ)+1
      NPRIM(NOBJ)=NPRIM(NOBJ)+1
1185  CALL ESUB
      IF(NVPRIM(NOBJ).EQ.0)GOTO 1190
      NDEF=NDEF+1
      GOTO 100
1190  CALL ERAS(NOBJ)
      NPRIM(NOBJ)=0
      GOTO 100
C
C     MOVE AN OBJECT
C
1600  IF(NDEF.EQ.0)GOTO 100
      CALL ON(1003)
      CALL PICKOB(IT,2)
      CALL POINTR(2,IT)
      CALL GET(2,XX,YY)
```

```
        CALL ATTACH(2)
        CALL TRAK(XX,YY)
        CALL MENUH(IT,1003,1003)
        CALL GRID(GRD,GRD)
        CALL ERAS
        GOTO 100
C
C       COMBINE TWO OBJECTS
C
1700    IF(I.LT.MARGIN)GOTO 4000
        IF(NDEF.LT.2)GOTO 100
        CALL PICKOB(IT,2)
1710    CALL PICKOB(IT2,3)
        IF(IT2.EQ.IT)GOTO 1710
        CALL MAKOBJ(NOBJ)
        CALL SUBP(NOBJ)
        CALL COPY(,IT)
        CALL GET(2,X1,Y1)
        CALL GET(3,X2,Y2)
        CALL LVECT(X2-X1,Y2-Y1,,-4)
        CALL OFF(IT2)
        CALL ERASP(3)
        CALL COPY(,IT2)
        CALL LVECT(X1-X2,Y1-Y2,,-4)
        CALL ESUB
        NPRIM(NOBJ)=NPRIM(IT)+NPRIM(IT2)+2
        NVPRIM(NOBJ)=NVPRIM(IT)+NVPRIM(IT2)
        NDEF=NDEF-1
        CALL ERAS(IT)
        CALL ERAS(IT2)
        NPRIM(IT)=0
        NPRIM(IT2)=0
        NVPRIM(IT)=0
        NVPRIM(IT2)=0
        GOTO 100
C
C       SCALE AN OBJECT
C
1800    IF(NDEF.EQ.0)GOTO 100
        CALL ON(1004)
        CALL MENUH(IT2,1030,1033)
        CALL PICKOB(IT,2)
        CALL OFF(IT)
        XX=0.
        YY=0.
        DO 1830 I=1,NPRIM(IT)
        CALL ADVANC(2)
        CALL GET(2,X,Y)
        CALL CHANGE(2,X*SF(IT2),Y*SF(IT2))
        CALL GET(2,X,Y)
        XX=XX+X
1830    YY=YY+Y
1840    CALL GET(2,X,Y)
        CALL CHANGE(2,X-XX,Y-YY)
        CALL ON(IT)
        GOTO 100
C
C       COPY AN OBJECT
C
1900    IF(I.LT.MARGIN)GOTO 4000
        IF(NDEF.EQ.0)GOTO 100
        CALL ON(1003)
        CALL PICKOB(IT,2)
1910    CALL MAKOBJ(NOBJ)
```

```
      CALL COPY(NOBJ,IT)
      CALL POINTR(2,NOBJ)
      CALL GET(2,X,Y)
      CALL ATTACH(2)
      CALL TRAK(X,Y)
      CALL MENUH(IT2,1003,1003)
      CALL GRID(GRD,GRD)
      CALL ERAS
      NDEF=NDEF+1
      NPRIM(NOBJ)=NPRIM(IT)
      NVPRIM(NOBJ)=NVPRIM(IT)
      IF(ISAC.EQ.0)GOTO 100
      ISAC=0
      GOTO 2210
C
C     ERASE AN OBJECT
C
2000  IF(NDEF.EQ.0)GOTO 100
      CALL PICKOB(IT,2)
      CALL ERAS(IT)
      NDEF=NDEF-1
      NVPRIM(IT)=0
      NPRIM(IT)=0
      GOTO 100
C
C     MODIFY AN OBJECT
C
2100  IF(NDEF.EQ.0)GOTO 100
      CALL ON(1005)
      CALL MENUH(IT2,1040,1044)
2105  IF(IT2.EQ.5)GOTO 100
2110  CALL GRATTN(1,IT,1)
      CALL LPEN(IH,IT,,,IP)
      IF(IH.EQ.0.OR.IT.LT.1.OR.IT.GT.209)GOTO 2110
      CALL POINTR(5,IT,IP)
      GOTO (2120,2140,2130,2170),IT2
C
C     ERASE A LINE
C
2120  CALL INTENS(5,-10)
      NVPRIM(IT)=NVPRIM(IT)-1
      IF(NVPRIM(IT).GT.0)GOTO 2100
      CALL ERAS(IT)
      NPRIM(IT)=0
      NDEF=NDEF-1
      GOTO 2100
C
C     MOVE A CORNER
C
2130  IF(IP.NE.NPRIM(IT)+1)GOTO 2150
      CALL POINTR(4,IT)
      CALL ATTACH(4)
      CALL POINTR(6,IT,2)
      GOTO 2155
C
C     SPLIT A LINE
C
2140  CALL GET(5,X,Y)
      CALL OFF(1000)
      CALL CHANGE(5,X/2.,Y/2.)
      CALL POINTR(2,IT,IP+1)
      CALL INSRT(2)
      CALL LVECT(X/2.,Y/2.)
      CALL INSRT
```

```
        CALL ON(1000)
        NPRIM(IT)=NPRIM(IT)+1
        NVPRIM(IT)=NVPRIM(IT)+1
2150    CALL POINTR(6,IT,IP+1)
2155    CALL ATTACH(5)
        CALL ATTACH(6,-1)
        CALL POINTR(2,IT)
        CALL GET(2,X,Y)
        DO 2160 I=1,IP-1
        CALL ADVANC(2)
        CALL GET(2,XX,YY)
        X=X+XX
2160    Y=Y+YY
        CALL TRAK(X,Y)
        CALL MENUH(IT2,1040,1044)
        CALL GRID(GRD,GRD)
        CALL ERAS
        GOTO 2105
C
C       SHOW ALL LINES
C
2170    CALL POINTR(5,IT)
        DO 2180 I=1,NPRIM(IT)
        CALL ADVANC(5)
2180    CALL INTENS(5)
        NVPRIM(IT)=NPRIM(IT)
        GOTO 2100
C
C       HIDE AN OBJECT
C
2200    IF(NDEF.EQ.0)GOTO 100
        CALL  PICKOB(IT,2)
2210    CALL OFF(IT)
        NVPRIM(IT)=-NVPRIM(IT)
        NDEF=NDEF-1
        NHID=NHID+1
        GOTO 100
C
C       SEEK AN OBJECT
C
2300    IF(NHID.EQ.0)GOTO 100
2305    DO 2310 I=1,200
        IF(NVPRIM(I).LT.0)CALL ON(I+0)
2310    IF(NVPRIM(I).GT.0)CALL OFF(I+0)
        CALL PICKOB(IT,2)
        NVPRIM(IT)=-NVPRIM(IT)
        NDEF=NDEF+1
        NHID=NHID-1
        DO 2320 I=1,200
        IF(NVPRIM(I).LT.0)CALL OFF(I+0)
2320    IF(NVPRIM(I).GT.0)CALL ON(I+0)
        IF(ISAC)1910,100,1910
C
C       SAVE THE DISPLAY
C
2400    CALL INFILE(FILE1,FILE2)
        CALL STOP
        CALL ASSIGN(2,FILE2)
        DEFINE FILE 2(2,256,U,INDX)
        WRITE(2'1)(NPRIM(I),I=1,256)
        WRITE(2'2)(NPRIM(I),I=257,420),(J,J=421,512)
        CALL CLOSE(2)
        CALL SAVE(FILE1)
        CALL LPEN(IH,IT)
```

```
        GOTO 100
C
C       RECALL A DISPLAY FILE
C
2500    CALL INFILE(FILE1,FILE2)
        CALL STOP
        CALL ASSIGN(2,FILE2)
        DEFINE FILE 2(2,256,U,INDX)
        READ(2'1)(NPRIM(I),I=1,256)
        READ(2'2)(NPRIM(I),I=257,420),(K,I=421,512)
        CALL CLOSE(2)
        CALL INIT
        CALL RSTR(FILE1)
        CALL LPEN(IH,IT)
        GOTO 100
C
C       SEEK AND COPY
C
2600    IF(I.LT.MARGIN)GOTO 4000
        IF(NHID.EQ.0)GOTO 100
        CALL ON(1003)
        ISAC=1
        GOTO 2305
C
C       ROTATE
C
2700    IF(NDEF.EQ.0)GOTO 100
        CALL ON(1006)
        CALL MENUH(IT2,1050,1052)
        CALL PICKOB(IT,2)
        CALL OFF(IT)
        DO 2750 I=1,NPRIM(IT)
        CALL ADVANC(2)
        CALL GET(2,X,Y)
        GOTO(2710,2720,2730),IT2
2710    CALL CHANGE(2,Y,-X)
        GOTO 2750
2720    CALL CHANGE(2,-X,-Y)
        GOTO 2750
2730    CALL CHANGE(2,-Y,X)
2750    CONTINUE
        CALL ON(IT)
        GOTO 100
4000    CALL CMPRS
        GOTO 100
5000    CALL FREE
        STOP
        END
        SUBROUTINE MENUH(IT,M1,M2)
C
C       WAIT FOR MENU HIT
C
100     CALL GRATTN(1,I,1)
        CALL LPEN(IH,IT)
        IF(IH.EQ.0.OR.IT.LT.M1.OR.IT.GT.M2)GOTO 100
        CALL POINTR(10,IT)
        CALL INTENS(10,8)
        CALL WAIT(5000)
        CALL LPEN(IH,IX)
        CALL INTENS(10,4)
        IT=IT+1-M1
        RETURN
        END
        SUBROUTINE PICKOB(IT,IP)
```

```
C
C       PICK AN OBJECT
C
        COMMON/DFILE/IBUF(4096)
        COMMON/PDATA/NPRIM(209),NVPRIM(209),NDEF,NHID
100     CALL GRATTN(1,I,1)
        CALL LPEN(IH,IT)
        IF(IH.EQ.0.OR.IT.LT.1.OR.IT.GT.209)GOTO 100
        CALL POINTR(IP,IT)
        CALL INTENS(IP,-8)
        CALL WAIT(5000)
        CALL INTENS(IP,-4)
        RETURN
        END
        SUBROUTINE INFILE(FILE1,FILE2)
C
C       INPUT A FILE NAME
C
        LOGICAL*1 FILE1(16),FILE2(16),DSP(5),DAT(5)
        DATA DSP,DAT/'.','D','S','P',0,'.','D','A','T',0/
1       CALL TTW(0,'FILENAME : ',-1)
        CALL KBS(16,FILE1,N)
        IF(N.EQ.0)GOTO 1
        DO 100 I=1,N
100     FILE2(I)=FILE1(I)
        DO 200 I=1,5
        FILE1(I+N)=DSP(I)
200     FILE2(I+N)=DAT(I)
        RETURN
        END
        SUBROUTINE MAKOBJ(NOBJ)
        COMMON/DFILE/IBUF(4096)
        COMMON/PDATA/NPRIM(209),NVPRIM(209),NDEF,NHID
        DO 100 NOBJ=1,209
        IF(NVPRIM(NOBJ).EQ.0)RETURN
100     CONTINUE
        STOP
        END
*
```

This FORTRAN example shows the DRAW program "distributed" between the host and satellite computers in a host-satellite system. DRAWH performs most of the same tasks done by the stand-alone DRAW program. DRAWS is a modified form of the User's Satellite Routine (USRSAT) that features local handlng of graphic attentions and local storage of display and data files on the satellite's floppy disks.

```
C
C       SIMPLE DRAWING PROGRAM USING DECGRAPHIC-11 SOFTWARE
C
C               HOST SIDE OF HOST/SATELLITE VERSION
C               USES INTEGER MODE, LOCAL SAVE/RSTR,
C               AND LK11 PUSH BUTTON BOX
C
        COMMON/PDATA/NPRIM(209),NVPRIM(209),NDEF,NHID
        LOGICAL*1 FILE1(16),FILE2(16),USED(5)
        INTEGER GRD,SM,YTOP,X,Y,XX,YY,X0,Y0,X1,Y1,X2,Y2,
     X  ZERO,TWO,FOUR,THREEC,SIZE,FIVEC
        DATA USED/48,0,0,37,0/
        SIZE=4096
        GRD=50
        SM=50
        YTOP=750
        ZERO=0
        TWO=2
        FOUR=4
        ONEC=100
        THREEC=300
        FIVEC=500
        IWARN=15
        MARGIN=10
        NDEF=0
        NHID=0
        ISAC=0
        DO 10 I=1,209
        NVPRIM(I)=0
10      NPRIM(I)=0
C
C       SET UP MENU AREAS
C
        CALL INIT(4096)
        CALL SUBP(1000)
        CALL OFF(1000)
        CALL SUBP(1001)
        CALL MENU(,YTOP,-SM,2010,'DRAW','MOVE','COMBINE','SCALE',
     X  'COPY','ERASE','MODIFY','HIDE','SEEK')
        CALL MENU(,YTOP-9*SM,-SM,2019,'SEEK & COPY','ROTATE','SAVE',
     X  'RECALL','EXIT')
        CALL ESUB
        CALL SUBP(1002)
        CALL MENU(,THREEC,-SM,1020,'POSITION','LINE','CLOSE','DONE')
        CALL ESUB
        CALL MENU(,THREEC,ZERO,1003,'DONE')
        CALL SUBP(1004)
        CALL MENU(,THREEC,-SM,1030,'1/4','1/2','2 X','4 X')
        CALL ESUB
        CALL SUBP(1005)
        CALL MENU(,THREEC,-SM,1040,'ERASE LINE','SPLIT LINE',
     X  'MOVE CORNER','SHOW ALL','DONE')
        CALL ESUB
        CALL SUBP(1006)
        CALL MENU(,THREEC,-SM,1050,'90 CW','180','90 CCW')
        CALL ESUB
```

```
      CALL SUBP(2006)
      CALL AREA(2)
      CALL APNT(ZERO,ZERO,,-4)
      CALL TEXT('     ')
      CALL ESUB(2006)
C
C     MAIN LOOP -- WAIT FOR MENU HIT AND BRANCH TO SERVICE IT
C
100   DO 110 I=1002,1006
110   CALL OFF(I)
      CALL ON(1001)
      CALL ON(1000)
      CALL DPTR(I)
      CALL POINTR(2,2006,2)
      I=(SIZE-I)/(SIZE/100)
      CALL FLASH(2,IWARN-I)
      USED(3)=I-I/10*10+48
      USED(2)=I/10+48
      CALL CHANGT(2,USED)
      CALL MENUH(IT,2010,2023)
      CALL OFF(1001)
      GOTO (1100,1600,1700,1800,1900,2000,2100,2200,2300,2600,2700,
     X  2400,2500,5000),IT
C
C     DRAW A NEW OBJECT
C
1100  IF(I.LT.MARGIN)GOTO 4000
      CALL ON(1002)
      CALL MAKOBJ(NOBJ)
      CALL SUBP(NOBJ)
      CALL APNT(FIVEC,FIVEC,1,-4)
      CALL POINTR(2,NOBJ)
      XX=FIVEC
      YY=FIVEC
1110  CALL ATTACH(2)
      CALL TRAK(XX,YY)
      CALL MENUH(IT,1020,1023)
      CALL GRID(GRD,GRD)
      CALL TRAKXY(XX,YY)
      CALL ERAS
      CALL GET(2,X,Y)
      IF(IABS(X).LT.GRD.AND.IABS(Y).LT.GRD.AND.NPRIM(NOBJ).NE.0)
     X  GOTO 1110
      II=-10
      GOTO(1140,1120,1160,1180),IT
1120  II=10
      NVPRIM(NOBJ)=NVPRIM(NOBJ)+1
1140  NPRIM(NOBJ)=NPRIM(NOBJ)+1
      CALL LVECT(ZERO,ZERO,,II)
      CALL ADVANC(2)
      GOTO 1110
1160  II=10
1180  CALL POINTR(2,NOBJ)
      CALL GET(2,X0,Y0)
      X=X0-XX
      Y=Y0-YY
      IF(IABS(X).LT.GRD.AND.IABS(Y).LT.GRD)GOTO 1185
      CALL LVECT(X,Y,,II)
      IF(II.GT.0)NVPRIM(NOBJ)=NVPRIM(NOBJ)+1
      NPRIM(NOBJ)=NPRIM(NOBJ)+1
1185  CALL ESUB
      IF(NVPRIM(NOBJ).EQ.0)GOTO 1190
      NDEF=NDEF+1
      GOTO 100
```

```
1190    CALL ERAS(NOBJ)
        NPRIM(NOBJ)=0
        GOTO 100
C
C       MOVE AN OBJECT
C
1600    IF(NDEF.EQ.0)GOTO 100
        CALL ON(1003)
        CALL PICKOB(IT,2)
        CALL POINTR(2,IT)
        CALL GET(2,XX,YY)
        CALL ATTACH(2)
        CALL TRAK(XX,YY)
        CALL MENUH(IT,1003,1003)
        CALL GRID(GRD,GRD)
        CALL ERAS
        GOTO 100
C
C       COMBINE TWO OBJECTS
C
1700    IF(I.LT.MARGIN)GOTO 4000
        IF(NDEF.LT.2)GOTO 100
        CALL PICKOB(IT,2)
1710    CALL PICKOB(IT2,3)
        IF(IT2.EQ.IT)GOTO 1710
        CALL MAKOBJ(NOBJ)
        CALL SUBP(NOBJ)
        CALL COPY(,IT)
        CALL GET(2,X1,Y1)
        CALL GET(3,X2,Y2)
        CALL LVECT(X2-X1,Y2-Y1,,-10)
        CALL OFF(IT2)
        CALL ERASP(3)
        CALL COPY(,IT2)
        CALL LVECT(X1-X2,Y1-Y2,,-10)
        CALL ESUB
        NPRIM(NOBJ)=NPRIM(IT)+NPRIM(IT2)+2
        NVPRIM(NOBJ)=NVPRIM(IT)+NVPRIM(IT2)
        NDEF=NDEF-1
        CALL ERAS(IT)
        CALL ERAS(IT2)
        NPRIM(IT)=0
        NPRIM(IT2)=0
        NVPRIM(IT)=0
        NVPRIM(IT2)=0
        GOTO 100
C
C       SCALE AN OBJECT
C
1800    IF(NDEF.EQ.0)GOTO 100
        CALL ON(1004)
        CALL MENUH(IT2,1030,1033)
        CALL PICKOB(IT,2)
        CALL OFF(IT)
        XX=ZERO
        YY=ZERO
        DO 1830 I=1,NPRIM(IT)
        CALL ADVANC(2)
        CALL GET(2,X,Y)
        GOTO (1805,1810,1815,1820),IT2
1805    CALL CHANGE(2,X/FOUR,Y/FOUR)
        GOTO 1825
1810    CALL CHANGE(2,X/TWO,Y/TWO)
        GOTO 1825
```

```
1815    CALL CHANGE(2,X*TWO,Y*TWO)
        GOTO 1825
1820    CALL CHANGE(2,X*FOUR,Y*FOUR)
1825    CALL GET(2,X,Y)
        XX=XX+X
1830    YY=YY+Y
1840    CALL GET(2,X,Y)
        CALL CHANGE(2,X-XX,Y-YY)
        CALL ON(IT)
        GOTO 100
C
C       COPY AN OBJECT
C
1900    IF(I.LT.MARGIN)GOTO 4000
        IF(NDEF.EQ.0)GOTO 100
        CALL ON(1003)
        CALL PICKOB(IT,2)
1910    CALL MAKOBJ(NOBJ)
        CALL COPY(NOBJ,IT)
        CALL POINTR(2,NOBJ)
        CALL GET(2,X,Y)
        CALL ATTACH(2)
        CALL TRAK(X,Y)
        CALL MENUH(IT2,1003,1003)
        CALL GRID(GRD,GRD)
        CALL ERAS
        NDEF=NDEF+1
        NPRIM(NOBJ)=NPRIM(IT)
        NVPRIM(NOBJ)=NVPRIM(IT)
        IF(ISAC.EQ.0)GOTO 100
        ISAC=0
        GOTO 2210
C
C       ERASE AN OBJECT
C
2000    IF(NDEF.EQ.0)GOTO 100
        CALL PICKOB(IT,2)
        CALL ERAS(IT)
        NDEF=NDEF-1
        NVPRIM(IT)=0
        NPRIM(IT)=0
        GOTO 100
C
C       MODIFY AN OBJECT
C
2100    IF(NDEF.EQ.0)GOTO 100
        CALL ON(1005)
        CALL MENUH(IT2,1040,1044)
2105    IF(IT2.EQ.5)GOTO 100
2110    CALL GRATTN(1,IT,1)
        CALL LPEN(IH,IT,,,IP)
        IF(IH.EQ.0.OR.IT.LT.1.OR.IT.GT.209)GOTO 2110
        CALL POINTR(5,IT,IP)
        GOTO (2120,2140,2130,2170),IT2
C
C       ERASE A LINE
C
2120    CALL INTENS(5,-10)
        NVPRIM(IT)=NVPRIM(IT)-1
        IF(NVPRIM(IT).GT.0)GOTO 2100
        CALL ERAS(IT)
        NPRIM(IT)=0
        NDEF=NDEF-1
        GOTO 2100
```

```
C
C       MOVE A CORNER
C
2130    IF(IP.NE.NPRIM(IT)+1)GOTO 2150
        CALL POINTR(4,IT)
        CALL ATTACH(4)
        CALL POINTR(6,IT,2)
        GOTO 2155
C
C       SPLIT A LINE
C
2140    CALL GET(5,X,Y)
        CALL OFF(1000)
        CALL CHANGE(5,X/TWO,Y/TWO)
        CALL POINTR(2,IT,IP+1)
        CALL INSRT(2)
        CALL LVECT(X-X/TWO,Y-Y/TWO)
        CALL INSRT
        CALL ON(1000)
        NPRIM(IT)=NPRIM(IT)+1
        NVPRIM(IT)=NVPRIM(IT)+1
2150    CALL POINTR(6,IT,IP+1)
2155    CALL ATTACH(5)
        CALL ATTACH(6,-1)
        CALL POINTR(2,IT)
        CALL GET(2,X,Y)
        DO 2160 I=1,IP-1
        CALL ADVANC(2)
        CALL GET(2,XX,YY)
        X=X+XX
2160    Y=Y+YY
        CALL TRAK(X,Y)
        CALL MENUH(IT2,1040,1044)
        CALL GRID(GRD,GRD)
        CALL ERAS
        GOTO 2105
C
C       SHOW ALL LINES
C
2170    CALL POINTR(5,IT)
        DO 2180 I=1,NPRIM(IT)
        CALL ADVANC(5)
2180    CALL INTENS(5)
        NVPRIM(IT)=NPRIM(IT)
        GOTO 2100
C
C       HIDE AN OBJECT
C
2200    IF(NDEF.EQ.0)GOTO 100
        CALL  PICKOB(IT,2)
2210    CALL OFF(IT)
        NVPRIM(IT)=-NVPRIM(IT)
        NDEF=NDEF-1
        NHID=NHID+1
        GOTO 100
C
C       SEEK AN OBJECT
C
2300    IF(NHID.EQ.0)GOTO 100
2305    DO 2310 I=1,200
        IF(NVPRIM(I).LT.0)CALL ON(I)
2310    IF(NVPRIM(I).GT.0)CALL OFF(I)
        CALL PICKOB(IT,2)
        NVPRIM(IT)=-NVPRIM(IT)
```

```
        NDEF=NDEF+1
        NHID=NHID-1
        DO 2320 I=1,200
        IF(NVPRIM(I).LT.0)CALL OFF(I)
2320    IF(NVPRIM(I).GT.0)CALL ON(I)
        IF(ISAC)1910,100,1910
C
C       SAVE THE DISPLAY
C
2400    CALL INFILE(1,FILE1,FILE2,I)
        IF(I.NE.0)GOTO 2450
        CALL STOP
        CALL ASSIGN(2,FILE2)
        DEFINE FILE 2(2,256,U,INDX)
        WRITE(2'1)(NPRIM(I),I=1,256)
        WRITE(2'2)(NPRIM(I),I=257,420),(J,J=421,512)
        CALL CLOSE(2)
        CALL SAVE(FILE1)
        CALL LPEN(IH,IT)
        GOTO 100
2450    DO 2460 I=1,420,20
2460    CALL TOOSAT(40,NPRIM(I))
        CALL LPEN(IH,IT)
        GOTO 100
C
C       RECALL A DISPLAY FILE
C
2500    CALL INFILE(0,FILE1,FILE2,I)
        IF(I.NE.0)GOTO 2550
        CALL STOP
        CALL ASSIGN(2,FILE2)
        DEFINE FILE 2 (2,256,U,INDX)
        READ(2'1)(NPRIM(I),I=1,256)
        READ(2'2)(NPRIM(I),I=257,420),(K,J=421,512)
        CALL CLOSE(2)
        CALL INIT
        CALL RSTR(FILE1)
        CALL LPEN(IH,IT)
        GOTO 100
2550    DO 2560 I=1,420,20
2560    CALL FRSAT(1,40,NPRIM(I))
        CALL LPEN(IH,IT)
        GOTO 100
C
C       SEEK AND COPY
C
2600    IF(I.LT.MARGIN)GOTO 4000
        IF(NHID.EQ.0)GOTO 100
        CALL ON(1003)
        ISAC=1
        GOTO 2305
C
C       ROTATE
C
2700    IF(NDEF.EQ.0)GOTO 100
        CALL ON(1006)
        CALL MENUH(IT2,1050,1052)
        CALL PICKOB(IT,2)
        CALL OFF(IT)
        DO 2750 I=1,NPRIM(IT)
        CALL ADVANC(2)
        CALL GET(2,X,Y)
        GOTO(2710,2720,2730),IT2
2710    CALL CHANGE(2,Y,-X)
```

```
        GOTO 2750
2720    CALL CHANGE(2,-X,-Y)
        GOTO 2750
2730    CALL CHANGE(2,-Y,X)
2750    CONTINUE
        CALL ON(IT)
        GOTO 100
4000    CALL CMPRS
        GOTO 100
5000    CALL FREE
        STOP
        END
        SUBROUTINE MENUH(IT,M1,M2)
C
C       GET A MENU HIT
C
        CALL TOSAT(102,2,2,M1,2,M2)
        CALL FRSAT(1,1,IT)
        RETURN
        END
        SUBROUTINE PICKOB(IT,IP)
C
C       PICK AN OBJECT
C
        CALL TOSAT(103,1,1,IP)
        CALL FRSAT(1,2,IT)
        RETURN
        END
        SUBROUTINE INFILE(IFUN,FILE1,FILE2,I)
C
C       INPUT A FILE NAME
C
        CALL TOSAT(101,1,1,IFUN)
        CALL FRSAT(3,11,FILE1,11,FILE2,1,I)
        RETURN
        END
        SUBROUTINE MAKOBJ(NOBJ)
        COMMON/PDATA/NPRIM(209),NVPRIM(209),NDEF,NHID
        DO 100 NOBJ=1,209
        IF(NVPRIM(NOBJ).EQ.0)RETURN
100     CONTINUE
        STOP
        END
        SUBROUTINE USRSAT(B,I)
C
C       SIMPLE DRAWING PROGRAM USING DECGRAPHIC-11 SOFTWARE
C
C               SATELLITE END OF HOST/SATELLITE VERSION
C               USES INTEGER MODE, LOCAL SAVE/RSTR,
C               AND LK11 PUSH BUTTON BOX
C
        LOGICAL*1 B(60),FILE1(16),FILE2(16)
        IF(I.EQ.0.OR.B(1).LT.101.OR.B(1).GT.103)RETURN
        GOTO (10100,10200,10300),B(1)-100
10100   K2=B(2)
        CALL INFILE(K2,FILE1,FILE2,I)
        RETURN
10200   CALL DHOST(1)
        CALL MENUH(IT,INTEGR(B(2)),INTEGR(B(4)))
        CALL TOHOST(1,IT)
        RETURN
10300   CALL DHOST(2)
        K=B(2)
        CALL PICKOB(IT,K)
```

```
      CALL TOHOST(2,IT)
      RETURN
      END
      SUBROUTINE MENUH(IT,M1,M2)
C
C     WAIT FOR MENU HIT
C
      LOGICAL*1 ALL(16),TEMP(16)
      DATA ALL/16*.TRUE./
      DO 10 I=1,16
10    TEMP(I)=.FALSE.
      CALL PBL(ALL,TEMP)
100   CALL GRATTN(1,IDEV,1,2,3)
      GOTO (130,120,110),IDEV
110   CALL KBC(IT)
      IT=IT-65+M1
      IH=1
      GOTO 135
120   CALL PBH(IH,TEMP)
      DO 125 IT=M1,M1+15
      IF(TEMP(IT-M1+1))GOTO 135
125   CONTINUE
      GOTO 100
130   CALL LPEN(IH,IT)
135   IF(IH.EQ.0.OR.IT.LT.M1.OR.IT.GT.M2)GOTO 100
      CALL POINTR(10,IT)
      CALL INTENS(10,8)
      IT=IT-M1+1
      TEMP(IT)=.TRUE.
      CALL PBL(ALL,TEMP)
      TEMP(IT)=.FALSE.
      CALL WAIT(5000)
      CALL LPEN(IH,IX)
      CALL INTENS(10,4)
      CALL PBL(ALL,TEMP)
      RETURN
      END
      SUBROUTINE PICKOB(IT,IP)
C
C     PICK AN OBJECT
C
100   CALL GRATTN(1,IT,1)
      CALL LPEN(IH,IT)
      IF(IH.EQ.0.OR.IT.LT.1.OR.IT.GT.209)GOTO 100
      CALL POINTR(IP,IT)
      CALL INTENS(IP,-8)
      CALL WAIT(5000)
      CALL INTENS(IP,-4)
      RETURN
      END
      SUBROUTINE INFILE(IFUN,FILE1,FILE2,II)
      COMMON/PDATA/NPRIM(209),NVPRIM(209),NDEF,NHID
      LOGICAL*1 FILE1(16),FILE2(16),DAT(5),DSP(5),D,X
      DATA DSP,DAT/'.','D','S','P',0,'.','D','A','T',0/
      DATA D,X/'D','X'/
      CALL DHOST(23)
1     CALL TTW(0,'FILENAME : ',-1)
      CALL KBS(16,FILE1,N)
      IF(N.EQ.0)GOTO 1
      DO 100 I=1,N
100   FILE2(I)=FILE1(I)
      DO 200 I=1,5
      FILE1(I+N)=DSP(I)
200   FILE2(I+N)=DAT(I)
```

```
      IF(FILE1(1).EQ.D.AND.FILE1(2).EQ.X)GOTO 300
      CALL TOHOST(11,FILE1,11,FILE2,1,0)
      RETURN
300   CALL TOHOST(22,FILE1,1,1)
      IF(IFUN.EQ.0)GOTO 800
      DO 400 I=1,420,20
400   CALL FRHOST(NPRIM(I))
      CALL ACCESS(1,FILE2,1,IR)
      IF(IR.LT.0)CALL ACCESS(2,FILE2,1,IR,2)
      CALL READWR(1,1,0,420,NPRIM,IR2)
      CALL SAVE(FILE1)
500   IF(IR.LT.0.OR.IR2.LT.0)CALL TTW(0,'IO ERROR')
      RETURN
800   CALL ACCESS(1,FILE2,1,IR)
      CALL READWR(0,1,0,420,NPRIM,IR2)
      DO 900 I=1,420,20
900   CALL TOHOST(40,NPRIM(I))
      CALL INIT
      CALL RSTR(FILE1)
      GOTO 500
      END
```

GLOSSARY

ABSOLUTE POINT
A point on the display screen that is identified by absolute x and y values in the current coordinate system (e.g., x=1023., y=32.). Regardless of the current beam position, the statement CALL APNT(100.,100.) will always move the beam to the coordinates (100.,100.) -- if they are legal coordinates -- and may also display a visible point at that location.

ABSOLUTE VECTOR
A line segment drawn from the current beam position to an absolute point; available only on VS60 display subsystems.

ACTIVE DISPLAY LIST
The list of instructions in the display file that is currently available to the display processor. The length of the active display list is always equal to or less than that of the total display file. New information is added to the active display list by the primitive-generating subroutines and by the DISPLY subroutine (see also DISPLAY FILE, DISPLAY PROCESSING UNIT, FRAME, INACTIVE DISPLAY LIST).

ATTENTION See GRAPHIC ATTENTION

ATTENTION FLAG
An indicator set by the DECgraphic-11 software to mark the occurrence of a graphic attention. The GRATTN subroutine reports the status of the attention flag and identifies the attention source (device). After the attention flag has been set, it is cleared by a call to an attention-handling subroutine (LPEN, PBS, PBH, or KBC). One of these subroutines should always be called after GRATTN reports a graphic attention so that the attention flag properly reflects the current status of your interactive devices.

BEAM
A stream of electrons directed at a position on the display screen; points, vectors, and other primitives are usually displayed relative to the current beam position. When you initialize the display file, the beam points to the lower left corner of the viewing area of the screen [the point (0.,0.) in the default coordinate system].

BLANK MODE See FLASH MODE

BRIGHTNESS See INTENSITY

CATHODE RAY TUBE <CRT>
An evacuated glass tube in which a beam of electrons is emitted and focused onto a phosphor-coated tube surface. A beam-deflection system moves the beam so that an image is traced out on the surface. The scopes used by the VT11 and VS60 processors are CRT units.

DISPLAY
Contents of the display screen at a given time. See also FRAME.

DISPLAY ELEMENT See PRIMITIVE

DISPLAY FILE
A discrete area of PDP-11 memory that is allocated to a COMMON area by the user program and is used to store the graphic instructions and data used in creating displays. The nonempty part of the display file (see also ACTIVE DISPLAY LIST) contains instructions and data that are retrieved and executed by the display processor to create images on the screen. The display file can be saved as an RT-11 or Files-11 data file and later restored and used by other programs.

DISPLAY LIST See ACTIVE DISPLAY LIST, INACTIVE DISPLAY LIST

DISPLAY PROCESSING UNIT (DISPLAY PROCESSOR, DPU)
The processing unit of a VT11 or VS60 graphic subsystem. The DPU retrieves instructions and data directly from the display file, without using the PDP-11 central processor.

DISPLAY SUBSYSTEM
The required hardware -- not including the PDP-11 computer -- for running programs with the DECgraphic-11 FORTRAN Graphics Package. A display subsystem therefore consists of a VT11 or VS60 display processor, a CRT display scope with light pen, and a keyboard. The LK-11 pushbutton box is an optional interactive device available on all DECgraphic-11 display subsystems. The terms "VS60" and "VT11" are often used to refer to entire display subsystems.

DRAWING AREA See IMAGE DEFINITION AREA

FLASH MODE
The mode in which a picture or part of a picture on the display screen blinks on and off. You can select this mode by specifying a positive integer value for the f display parameter (included in many subroutine calls) or with the FLASH subroutine. When you initialize the display file, flash mode is disabled.

FLICKER
An unsteadiness in the displayed image, caused in refreshed displays when the display processor does not have enough time to complete one frame before the phosphor at the beginning of the frame begins to fade out.

FONT
A kind of type in which characters are displayed. You can display characters in the normal (Roman) type font or in italics. To specify the italic font, call the TEXT subroutine and precede the character string to be displayed with a special control code.

FRAME

The picture created by one pass of the display processing unit through the active display list. To avoid flicker in a refreshed display, 30 or more frames per second must typically be executed. Between frames, primitives and subpictures are added and deleted according to your programmed instructions.

GRAPHIC ARRAY

A collection of values stored in an array and plotted as points on the x or y axis by the XGRA or YGRA subroutines. Graphic arrays can also be plotted as sets of long vectors with the FIGR subroutine.

GRAPHIC ATTENTION

An event created when the light pen is pointed at a primitive on the display screen that has been made light-pen sensitive, or by a keystroke on the LK-11 pushbutton box or terminal keyboard.

To be properly recognized as graphic attentions, these actions must be detected by subroutines such as GRATTN, LPEN, PBH, or KBC.

GRID

A logical construct of imaginary points evenly spaced at user-defined intervals on the display screen. If you call the GRID subroutine, the tracking object automatically moves to the nearest point on the grid. This action allows you to adjust the coordinates of primitives to predefined increments, a useful feature in drawing rectilinear objects.

HOST-SATELLITE GRAPHIC SYSTEM

A hardware/software configuration in which graphic processing is distributed between a large host computer (such as a PDP-11/70) and a satellite graphic terminal driven by a smaller satellite computer (e.g., a PDP-11/34). The satellite terminal in such a system closely resembles a stand-alone system, except that the display subsystem is connected to the host computer by a serial data link rather than by the UNIBUS. Because the satellite terminal can be used as a general-purpose terminal to the host computer, host-satellite systems are sometimes more cost-efficient than stand-alone systems. However, if the same ease of programming is desired in a host-satellite system as exists in stand-alone systems, the host-satellite system cannot execute a graphic program as quickly because of the communication overhead required. In the DECgraphic-11 FORTRAN Graphics Package, parts of the host-satellite software can be modified to distribute graphic processing between host and satellite in a manner that is most efficient for a given application (see Chapter 5 of this manual). Host-satellite systems can be configured with RSX-11M as the host operating system; RSX-11M can also drive a stand-alone graphic system. RSX-11D and IAS can also be used as host-satellite operating systems but not as stand-alone systems.

IMAGE DEFINITION AREA

The total VS60 area in which you can define primitives. In the default (unit-scaled) coordinate system, it extends from a lower left corner at coordinate position (x=-4095., y=-4095.) to an upper right corner at (x=4095., y=4095.). You can define a viewport on this area to examine different parts of it.

INACTIVE DISPLAY LIST

A list of graphic instructions and data, residing in the display file, that is not included in the current frame by the display processor. The subroutine DISPLY can create inactive display lists and can also add the inactive display list to the active display list. Creating an inactive list, and then activating it as a block, is a good method for creating graphic pictures quickly, particularly in host-satellite graphic systems.

INTENSITY

The apparent brightness of objects on the display screen. You can select the intensity level of a picture or part of a picture by specifying a new value for the i parameter (included in many subroutine calls) or with the INTENS subroutine. When you initialize the display file, the intensity level is set to 4, which is normally bright enough for the object to be detected by a light pen. If the specified intensity is negative, the display will be invisible. The absolute brightness of a display is dependent on the setting of the INTENSITY or BRIGHTNESS knob on the display scope.

KEYBOARD INTERACTION

An optional interactive feature of DECgraphic-11 software, available on all operating systems. The ASCII values of keyboard characters are read by the subroutine KBC and can be programmed to represent program branches, menu items, or large-scale graphic functions. Each distinct ASCII code represents a distinct graphic attention.

LIGHT BUTTON

A name sometimes given to an item in the menu area of the display screen. The item is usually a character string that has been made sensitive to the light pen. Light buttons can also be selected with an LK-11 pushbutton box or (using the KBC subroutine) by striking a particular key on the terminal keyboard. The keyboard and pushbutton box can select light buttons whether or not they are sensitive to the light pen.

LIGHT PEN

A solid-state light-detecting device consisting of a photosensitive diode. It is attached by a cord to the VT11 or VS60 display scope. If a primitive or subpicture has been made light-pen sensitive, pointing to it with the light pen creates a light-pen hit, or graphic attention. Pressing the tip switch of the VS60 light pen against the screen creates a second type of graphic attention, giving you an additional interactive control.

LIGHT-PEN HIT

An event, or graphic attention, created when the light pen is pointed at a primitive or subpicture on the display screen that has been made light-pen sensitive. A hit is internally recognized as a interrupt from the light-pen device. The subroutine GRATTN can wait for and respond to light-pen hits, and detailed information on the hit is returned by the LPEN subroutine. LPEN also returns information about graphic attentions created by the tip switches on the VS60 light pen. If you use GRATTN to handle light-pen hits, you should always follow the GRATTN call by a call to LPEN to clear the ATTENTION flag.

LIGHT-PEN SENSITIVITY
A characteristic of a primitive or subpicture. If a primitive or subpicture is light-pen sensitive, a graphic attention will occur when you point to the object with the pen. You can enable light-pen sensitivity by specifying a positive value for the l parameter (included in many subroutine calls) or by using the SENSE subroutine. When you initialize the display file, light-pen sensitivity is disabled.

LINE TYPE
The type of line used to display vectors on the screen. You can select one of four types:

1. solid line
2. long-dash line
3. short-dash line
4. dot-dash line

You can specify a new value for the t parameter (included in many subroutine calls) or you can use the LINTYP subroutine. When you initialize the display file, the line type is solid.

LONG VECTOR
A vector stored in long-vector format, occupying two words. A long vector may not exceed 1023 rasters in length.

MAIN AREA **See VIEWING AREA**

MENU
A list of character strings or small pictures, also called light buttons. You can select an option from this list by touching the desired character string with the light pen, by pressing the proper pushbutton on the optional LK-11 box, or by pressing a programmed keyboard character. You can specify 10 light buttons in a single call to the MENU subroutine; by calling MENU repeatedly, you can display as many light buttons as can fit on the desired area of the screen.

MENU AREA
A hardware area in the VS60 display processor used to display a list of options called light buttons, or menu items. The area is the rightmost one and two-third inch vertical strip on the screen (4 centimeters by 30 centimeters). This area can accommodate a horizontal capacity of 128 raster units, or 14 normal-sized characters. With the MENU subroutine, you can display the menu anywhere on the VS60 screen, although the hardware menu area is chosen by default. The VT11 screen does not have a hardware menu area; the area chosen by default is simply the extreme right edge of the VT11 viewing area.

POINTER
One of 20 special elements used to identify primitives in the display file.

PRECEDENT
In nested subpicture calls, the precedent of Subpicture m1 is the "calling" subpicture, i.e., another subpicture that references m1. Because up to eight subpictures can be nested, a subpicture could have as many as seven precedents. In calls to LPEN, the tags of precedent subpictures are returned in array IA.

PRIMITIVE

A basic display element, such as a point, vector, or character string, that can be defined in a single subroutine call and stored in the display file.

PUSHBUTTON BOX (LK-11)

An optional DECgraphic-11 interactive device, available on all systems. The box has 16 buttons mounted in a small keypad. Each button can be programmed (with the PBS, PBH, and PBL subroutines) to represent a program branch (for example, to select and respond to a menu item). Each button creates a distinct graphic attention when pressed. The buttons are "two-state" switches and are assigned LOGICAL*1 .TRUE. and .FALSE. values by the DECgraphic-11 software. Each button has an internal light that normally is lit when the button's state is .TRUE.

RASTER UNIT

The smallest resolvable distance between two adjacent points. There are 1024 x 1024 raster units in the viewing area of the VT11 and VS60 display screens, and 8192 x 8192 raster units in the image definition area of the VS60.

RELATIVE POINT

A point on the display screen that is defined in relation to the current beam position. If the current beam position is at (10.,20.), the statement CALL RNPT(12.,12.) moves the beam (and may also display a point) at absolute position (22.,32.). Compare with ABSOLUTE POINT.

RELATIVE VECTOR

A line segment drawn from the current beam position to a coordinate position relative to the beam position. Compare with ABSOLUTE VECTOR and RELATIVE POINT.

SCALING

Defining a user window in which physical distances and locations on the screen are measured according to a nonunit scale -- for example, in increments of ten raster units rather than one. Scaling is accomplished with the WINDW subroutine. On the VS60, the WINDW subroutine operates on the total image definition area, not just the visible portion on the screen. VS60 users can also scale (enlarge or shrink) the sizes of vectors and characters with the CVSCAL subroutine.

SHORT VECTOR

A vector stored in short-vector format, occupying one memory word. A short vector cannot exceed 63 rasters in length. Short vectors are displayed by the SVECT subroutine. The VECT subroutine uses the short-vector format whenever possible. Short vectors cannot be attached to the tracking object. Compare with LONG VECTOR.

STAND-ALONE GRAPHIC SYSTEM

A graphic hardware/software configuration in which the VT11 or VS60 display processor is connected to the PDP-11 central processor by the UNIBUS. Stand-alone systems have the advantage of higher speed compared with host-satellite systems, at the expense of dedicating computer hardware more or less totally to graphic programming. All DECgraphic-11 systems that use the RT-11 operating system are stand-alone systems. If you use RSX-11M, you can use the DECgraphic-11 software in either a stand-alone or host-satellite configuration.

SUBPICTURE
An entity defined by grouping together several primitive definitions. A subpicture is analogous to a subroutine and is used for the same reasons -- primarily modularity and efficiency. By referencing a subpicture, you often save the overhead that would otherwise be required to specify each primitive included in the subpicture definition. For a discussion of the "break-even" point for using subpictures instead of individual primitives, see Chapter 3 of this manual.

TAG
A unique name assigned to a subpicture. A tag must be a positive integer in the range 1 through 32767.

TIP SWITCH
A switch on the tip of the VS60 light pen that is set when the pen is pushed against the screen. It provides an additional graphic attention, separate from the light-pen hit, that is useful in interactive graphics.

TRACKING OBJECT
An octagonal image that can be displayed on the screen by the TRAK subroutine. It moves automatically to center itself on any light-pen hit in its area. If you call GRID, the tracking object will move to the nearest point on the logical grid. The tracking object is stored internally as a subpicture of relative vectors.

UNIT-SCALED COORDINATES
Coordinates in the default (unscaled) coordinate system, in which adjacent positions on an axis are separated by a single raster unit.

VECTOR
A line segment extending from one coordinate position to another on the display screen. The length of a relative vector cannot exceed 1023 raster units. See also ABSOLUTE VECTOR, LONG VECTOR, RELATIVE VECTOR, SHORT VECTOR.

VIEWING AREA
On the VT11, the 9.25-by-9.25-inch area of the screen in which images can be displayed. On the VS60, the 12-by-12-inch screen area that forms the current viewport on the total image definition area of the display processor. On the VS60, the viewport can be shifted to reveal different parts of the image definition area, but the default viewport is the area extending from a lower left corner at the (unit-scaled) coordinate position (0.,0.) to an upper right corner at (1023.,1023.). The viewing area on either a VT11 or VS60 consists of 1024 raster units along the x and y axes.

VIEWPORT
On the VS60, the part of the image definition area that is visible in the viewing area. A viewport always consists of 1024 x 1024 raster units. The position of the viewport can be shifted on the image definition area by the VIEWPT subroutine; the default viewport extends from the (unit-scaled) coordinate (0.,0.) to (1023.,1023.).

INDEX

DECGRAPHIC-11
FORTRAN Programming
Manual
AA-5428C-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 152
MARLBOROUGH, MA
01752

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
200 Forest Street MR1-2/E37
Marlborough, Massachusetts 01752