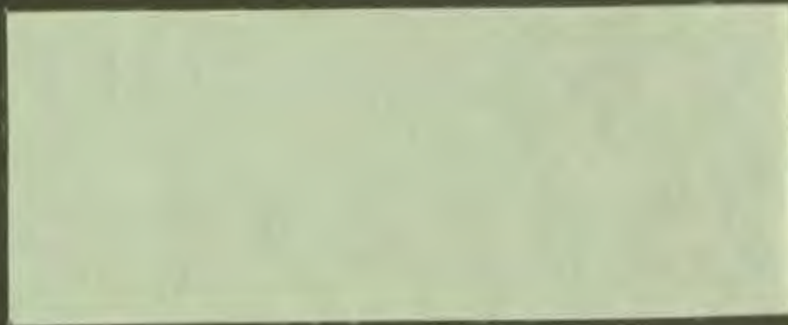


Professional™
300 series



Developer's Tool Kit

SHREWSBURY LIBRARY
DIGITAL EQUIPMENT CORPORATION
SHR1 3/G18
DTN 237-3400

DECLIT AA PRO N619C
CORE graphics library
manual : developer's tool
kit

DECLIT
AA
PRO
N619C

Digital Equipment Corporation
SHREWSBURY LIBRARY

digital
software

94-003/049/24

Digital Equipment Corporation
SHREWSBURY LIBRARY

CORE Graphics Library Manual

Order No. AA-N619C-TK

April 1984

This document describes the Professional 300 series CORE Graphics Library. It is intended to be used as a reference manual and user guide for programmers developing graphics applications with the Professional Host Tool Kit or PRO/Tool Kit.

DEVELOPMENT SYSTEM: Professional Host Tool Kit V2.0
PRO/Tool Kit V2.0

SOFTWARE VERSION: CORE Graphics Library V2.0

DECLIT
AA
PRO
N619C

32743

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754

First Printing, December 1982
Revised, September 1983
Revised, April 1984

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 or equipment that is not supplied by DIGITAL or its affiliated companies.

The specifications and drawings, herein, are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for manufacture or sale of items without written permission.

Copyright © 1982, 1983, 1984 by Digital Equipment Corporation
All Rights Reserved

The following are trademarks of Digital Equipment Corporation:

CTIBUS	MASSBUS	Rainbow
DEC	PDP	RSTS
DECmate	P/OS	RSX
DECsystem-10	PRO/BASIC	Tool Kit
DECSYSTEM-20	PRO/Communications	UNIBUS
DECUS	Professional	VAX
DECwriter	PRO/FMS	VMS
DIBOL	PRO/RMS	VT
digital	PROSE	Work Processor
	PROSE PLUS	

CONTENTS

PREFACE	ix
CHAPTER 1	OVERVIEW
1.1	RELATIONSHIP TO OTHER GRAPHICS TOOLS 1-1
1.2	THE CORE GRAPHICS STANDARD 1-2
1.3	GRAPHICS PROGRAMMING 1-4
1.4	CONTROLLING THE GRAPHICS SYSTEM 1-5
1.5	DESCRIBING THE GRAPHICAL WORLD 1-7
1.5.1	WORLD COORDINATES 1-8
1.5.1.1	THE WINDOW 1-10
1.5.1.2	THE CURRENT POSITION 1-11
1.5.1.3	ABSOLUTE VS. RELATIVE POSITIONS 1-12
1.5.2	NORMALIZED DEVICE COORDINATES 1-12
1.5.2.1	NDC SPACE 1-13
1.5.2.2	THE VIEWPORT 1-13
1.5.3	PHYSICAL DEVICE COORDINATES 1-14
1.5.4	THE VIEWING TRANSFORMATION 1-14
1.6	OUTPUT PRIMITIVES - THE GRAPHICAL "BUILDING BLOCKS" 1-16
1.6.1	CURRENT POSITION INSTRUCTIONS 1-17
1.6.2	MARKER PRIMITIVE INSTRUCTIONS 1-17
1.6.3	LINE PRIMITIVE INSTRUCTIONS - THE GRAPHICAL "PEN" 1-18
1.6.4	TEXT PRIMITIVE INSTRUCTIONS 1-19
1.7	ATTRIBUTES - CONTROLLING THE WAY OUTPUT PRIMITIVES LOOK 1-21
1.7.1	LINE ATTRIBUTES 1-21
1.7.2	MARKER ATTRIBUTES 1-24
1.7.3	TEXT ATTRIBUTES 1-25
1.7.4	COLOR - THE BITMAP ARCHITECTURE 1-30
1.7.4.1	HOW COLORS ARE FORMED 1-31
1.7.4.2	THE COLOR MAP 1-32
1.7.4.3	THE BITMAP/COLOR MAP INTERFACE 1-34
1.7.5	THE WRITING MODE 1-36
1.7.6	THE GLOBAL ATTRIBUTE LIST 1-38
CHAPTER 2	PROGRAMMING WITH THE CORE GRAPHICS LIBRARY
2.1	CALLING CGL ROUTINES FROM HIGH-LEVEL LANGUAGES . . 2-1
2.1.1	THE BASIC-PLUS-2 INTERFACE 2-1
2.1.2	THE COBOL-81 INTERFACE 2-2
2.1.3	THE DIBOL INTERFACE 2-3
2.1.4	THE FORTRAN INTERFACE 2-4
2.1.5	THE PASCAL INTERFACE 2-5
2.2	CALLING CGL ROUTINES FROM MACRO-11 2-6
2.3	TERMINAL INPUT/OUTPUT 2-7
2.4	USING CGL WITH THE P/OS USER INTERFACE LIBRARY . . 2-7

2.5	BUILDING YOUR CGL PROGRAM	2-8
2.6	INSTALLING YOUR CGL PROGRAM	2-10

CHAPTER 3 CONTROL INSTRUCTIONS

3.1	INITIALIZE_CORE - PREPARE GRAPHICS SYSTEM FOR USE	3-1
3.2	TERMINATE_CORE - GRAPHICS SYSTEM USAGE FINISHED .	3-2
3.3	NEW_FRAME - REFRESH VIEW SURFACE	3-2
3.4	INITIALIZE_VIEW_SURFACE - ENABLE ACCESS TO DEVICE	3-3
3.5	TERMINATE_VIEW_SURFACE - DISABLE ACCESS TO DEVICE	3-4
3.6	SELECT_VIEW_SURFACE - ENABLE GRAPHICS OUTPUT TO DEVICE	3-5
3.7	DESELECT_VIEW_SURFACE - DISABLE GRAPHICS OUTPUT TO DEVICE	3-7
3.8	PLAYBACK_FILE - EXECUTE FILE OF GRAPHICS COMMANDS	3-8
3.9	BEGIN_BATCH - BEGIN STORING VIEW SURFACE UPDATES .	3-9
3.10	END_BATCH - END BATCH OF UPDATES	3-9
3.11	CGL_WAIT - SUSPEND EXECUTION	3-10
3.12	ERASE_VIEWPORT - ERASE IMAGES IN VIEWPORT . . .	3-10
3.13	PRINT_SCREEN - SEND SCREEN IMAGE TO PRINTER . .	3-11
3.14	REPORT_MOST_RECENT_ERROR - IDENTIFY EXECUTION ERROR	3-12

CHAPTER 4 VIEWING TRANSFORMATION INSTRUCTIONS

4.1	SET_WINDOW - SPECIFY WORLD COORDINATE SPACE . . .	4-1
4.2	SET_ORIGIN - SPECIFY ORIGIN OF WINDOW	4-2
4.3	SET_WINDOW_CLIPPING - ENABLE OR DISABLE WINDOW CLIPPING	4-3
4.4	SET_NDC_SPACE_2 - DEFINE SIZE OF NDC SPACE	4-3
4.5	SET_VIEWPORT_2 - SPECIFY USABLE AREA OF VIEW SURFACE	4-5
4.6	SCROLL - MOVE SCREEN CONTENTS	4-6
4.7	SCROLL_VIEWPORT - MOVE VIEWPORT CONTENTS	4-

CHAPTER 5 GLOBAL ATTRIBUTE INSTRUCTIONS

5.1	SET_WRITING_INDEX - SELECT COLOR MAP INDEX FOR IMAGES	5-1
5.2	SET_BACKGROUND_INDEX - SET BACKGROUND COLOR MAP INDEX	5-2
5.3	SET_COLOR_MAP_ENTRY - SET COLOR MAP ENTRY RGB VALUES	5-3
5.4	SET_COLOR_MAP - SET ALL COLOR MAP RGB VALUES . . .	5-4
5.5	SET_WRITING_PLANES - SELECT COMBINATION OF PLANES	5-6
5.6	SET_WRITING_MODE - SET WRITING CHARACTERISTICS . .	5-7
5.7	SET_GLOBAL_ATTRIBUTES - SET GLOBAL ATTRIBUTE LIST	5-8

CHAPTER 6

CURRENT POSITION AND MARKER INSTRUCTIONS

- 6.1 CURRENT POSITION INSTRUCTIONS 6-1
 - 6.1.1 MOVE_ABS_2 - Move to Absolute Position 6-1
 - 6.1.2 MOVE_REL_2 - Move Relative to Current Position . 6-2
 - 6.1.3 INQUIRE_CURRENT_POSITION_2 - Get Current Position 6-2
 - 6.1.4 SET_CURSOR - Specify Cursor Characteristics . . 6-3
- 6.2 MARKER PRIMITIVE INSTRUCTIONS 6-4
 - 6.2.1 MARKER_ABS_2 - Draw Marker at Absolute Position 6-4
 - 6.2.2 MARKER_REL_2 - Draw Marker Relative to Current Position 6-5
 - 6.2.3 POLYMARKER_ABS_2 - Draw Markers at Absolute Positions 6-5
 - 6.2.4 POLYMARKER_REL_2 - Draw Markers at Relative Positions 6-6
- 6.3 MARKER ATTRIBUTE INSTRUCTIONS 6-7
 - 6.3.1 SET_MARKER_SYMBOL - Select New Marker Symbol . . 6-7

CHAPTER 7

LINE INSTRUCTIONS

- 7.1 STRAIGHT LINE PRIMITIVE INSTRUCTIONS 7-1
 - 7.1.1 LINE_ABS_2 - Draw Line to Absolute Position . . 7-1
 - 7.1.2 LINE_REL_2 - Draw Line to Relative Position . . 7-1
 - 7.1.3 POLYLINE_ABS_2 - Draw Lines to Absolute Positions 7-2
 - 7.1.4 POLYLINE_REL_2 - Draw Lines to Relative Positions 7-3
 - 7.1.5 POLYGON_ABS_2 - Draw Polygon by Absolute Positions 7-4
 - 7.1.6 POLYGON_REL_2 - Draw Polygon by Relative Positions 7-5
 - 7.1.7 RECTANGLE_ABS_2 - Draw Rectangle by Absolute Position 7-6
 - 7.1.8 RECTANGLE_REL_2 - Draw Rectangle by Relative Position 7-7
- 7.2 CURVED LINE PRIMITIVE INSTRUCTIONS 7-8
 - 7.2.1 ARC_ABS_2 - Draw Arc Based on Absolute Position 7-8
 - 7.2.2 ARC_REL_2 - Draw Arc Based on Relative Position 7-9
 - 7.2.3 CURVE_ABS_2 - Draw Curve by Absolute Positions 7-11
 - 7.2.4 CURVE_REL_2 - Draw Curve by Relative Positions 7-12
- 7.3 LINE ATTRIBUTE INSTRUCTIONS 7-13
 - 7.3.1 SET_LINestyle - Set Line Drawing Style 7-13
 - 7.3.2 SET_LINEWIDTH - Set Line Drawing Width 7-14
 - 7.3.3 SET_LINEWIDTH_ORIENTATION - Set Line Endpoint Offset 7-16
 - 7.3.4 SET_FILL_MODE - Enable or Disable Area Fill . 7-18
 - 7.3.5 SET_FILL_ENTITY - Specify Line or Point for Fill Reference 7-19
 - 7.3.6 SET_FILL_CHAR - Specify Character for Fill . . 7-20

CHAPTER 8 TEXT INSTRUCTIONS

8.1	TEXT PRIMITIVE INSTRUCTIONS	8-1
8.1.1	TEXT - Draw Line of Text	8-1
8.1.2	INQUIRE_TEXT_EXTENT_2 - Report Position at End of String	8-2
8.1.3	LOAD_FONT - Load User-defined Font	8-2
8.1.4	LOAD_CHARACTER - Load User-defined Character	8-3
8.1.5	BEGIN_DEFINE_CHARACTER	8-4
8.1.6	END_DEFINE_CHARACTER	8-5
8.2	TEXT ATTRIBUTE INSTRUCTIONS	8-6
8.2.1	SET_CHARSIZE - Set Character Size	8-6
8.2.2	SET_CHARSPACE - Set Character Spacing	8-7
8.2.3	SET_CHARPATH - Set Text Writing Direction	8-8
8.2.4	SET_CHARJUST - Set Text Justification	8-10
8.2.5	SET_CHARITALIC - Set Character Slant	8-10
8.2.6	SET_FONT - Select Character Font	8-11
8.2.7	SET_FONT_SIZE - Define Size of Character Font	8-12

APPENDIX A ERROR MESSAGES

APPENDIX B OPTIONAL VIEW SURFACES

B.1	HEWLETT-PACKARD HP7470A AND HP7475A GRAPHICS PLOTTERS	B-1
B.1.1	Hardware Requirements	B-1
B.1.2	Setting Up the Plotter	B-2
B.1.3	Physical Device Coordinate Space	B-2
B.1.4	Inoperative Instructions	B-2
B.1.5	SET_WRITING_INDEX	B-3
B.1.6	SET_BACKGROUND_INDEX	B-4
B.1.7	SET_WRITING_MODE	B-4
B.1.8	SET_MARKER_SYMBOL	B-4
B.1.9	SET_LINESTYLE	B-5
B.1.10	SET_LINEWIDTH	B-5
B.1.11	SET_FILL_CHAR	B-5
B.1.12	SET_FONT	B-8
B.1.13	SET_FONT_SIZE	B-8
B.1.14	Plotter Errors	B-9
B.1.15	HP-GL Features Not Accessible from CGL	B-9

APPENDIX C INCLUDE FILES

C.1	BASIC-PLUS-2	C-1
C.2	DIBOL	C-4
C.3	FORTRAN-77	C-6
C.4	PASCAL	C-9

APPENDIX D

EXAMPLE PROGRAMS

D.1	COLORMAP.PAS - COLOR MAP EDITOR	D-1
D.1.1	COLORMAP.HLP - HELP FRAME	D-4
D.2	GEDIT.B2S - GRAPHICS SKETCHPAD	D-5
D.3	FONT.B2S - DISPLAY A FONT	D-12
D.3.1	SUITS.FNT - SAMPLE USER-DEFINED FONT	D-14
D.4	MODE.B2S - DEMONSTRATE WRITING MODES	D-15

APPENDIX E

SUMMARY OF INSTRUCTIONS

APPENDIX F

GLOSSARY

INDEX

FIGURES

1-1	A Graphics System	1-3
1-2	A Cartesian Coordinate System	1-8
1-3	The Window In World Coordinate Space	1-9
1-4	The Origin of the Window	1-10
1-5	Default Normalized Device Coordinate Space	1-11
1-6	The Viewport	1-12
1-7	The Viewing Transformation	1-13
1-8	One Image in Several Viewports	1-15
1-9	Polygon Fill (Drawn on HP7470 Plotter)	1-22
1-10	Open Area Fill Modes	1-23
1-11	Character Size	1-24
1-12	Character Spacing	1-25
1-13	Character Path	1-26
1-14	Character Justification	1-27
1-15	Character Italic	1-28
1-16	Font 0	1-29
1-17	The Bitmap with Extended Bitmap Option	1-30
1-18	The Color Map with Default Values	1-32
1-19	The Color Map/Bitmap Interface	1-34
1-20	The Writing Modes (Shown with Line Style)	1-35
4-1	The SCROLL Instruction with Default Window Origin	4-7
7-1	An Arc in World Coordinate Space	7-9
7-2	The Standard Line Styles	7-14
7-3	Line Width Orientations: Bottom-left and Centered	7-17
B-1	Hatch Patterns 1 through 18	B-7
B-2	Pie Chart (Drawn with HP7470 Plotter)	B-8

TABLES

5-1	Integer Attribute List	5-8
5-2	Real Attribute List	5-9
B-1	Hatch Patterns	B-6

PREFACE

Document Objectives

This manual describes the Professional 300 Series CORE Graphics Library. It provides both reference and user information.

Intended Audience

This document is intended to be used as a reference manual and user guide for programmers developing graphics applications with the Professional Developer's Tool Kit.

Document Structure

This manual describes a graphics system consisting of approximately 100 instructions.

Chapter 1 is devoted to an overall "user guide" description of the system. It describes what a graphics system is, what the instructions do, and how they interact with each other.

Chapter 2 describes how to use the CORE Graphics Library with various programming languages and how to task build graphics programs.

Chapters 3 through 8 make up a "reference manual" for individual instructions. The various types of output primitive instructions and their associated attribute instructions are grouped together by function.

Within each chapter, the instructions are ordered from simplest to most complex. For some users, this is equivalent to ordering them from most often used to least often used. Testing has shown this order to provide the quickest access.

Each CORE Graphics Library instruction is documented in the following format:

- **Instruction Name and Description**

CORE Graphics Library instruction names were selected for compatibility with the ACM SIGGRAPH CORE Standard. Where appropriate, CORE Standard names are used. For example, names that end with "_2" specify a two-dimensional instruction. The CORE Standard provides for three-dimensional graphics.

A brief description of each instruction is provided, except for symmetric INQUIRE instructions. All SET instructions have a corresponding INQUIRE instruction that returns the current values of the instruction's parameters.

PREFACE

- **CORE Standard**

To facilitate program portability, the CORE Standard "function" that corresponds to each CORE Graphics Library instruction is shown. For the most part, CORE Standard parameters indicate nonspecific data types and structures, and, in some cases, do not match the CORE Graphics Library parameters.

- **CORE Graphics Library**

Each CORE Graphics Library instruction name is shown with the data types, positions, and semantic meanings of the parameters. For example:

```
LINE_ABS_2 (X, Y)
```

For symmetric (SET/INQUIRE) instructions, respective data types are shown. For example:

```
SET_WRITING_INDEX (index)
```

```
INQUIRE_WRITING_INDEX (index)
```

index is an integer expression/variable that specifies
...

The term "expression" corresponds to the SET instruction and the term "variable" corresponds to the INQUIRE instruction. A SET instruction does not return a value; thus, in theory, you should be able to pass an expression such as:

```
SET_WRITING_INDEX (((X + Y) * Z) MOD 8)
```

Some programming languages allow expressions as reference parameters by evaluating the expression and storing the result in a temporary location. Others do not. If your programming language does not support this feature, please read "expression" as "value."

- **Notes**

Additional information relating to the usage of the instruction is provided.

- **Errors**

The most likely execution errors are provided.

PREFACE

- **Example**

Where appropriate, example program fragments are provided. For clarity, the examples use language-independent CORE Graphics Library instruction names.

Appendix A contains the CORE Graphics Library error messages.

Appendix B contains information about non-default view surfaces, such as Hewlett-Packard graphics plotters.

Appendix C contains listings of language-specific constant declaration files.

Appendix D contains several PASCAL and BASIC-PLUS-2 example programs, provided for instructional purposes.

Appendix E provides an alphabetic summary of the CORE Graphics Library instructions.

Appendix F provides a glossary of commonly used terms.

How to Use this Manual

Read Chapter 1 carefully, particularly if you are not an experienced graphics programmer. A sound understanding of the coordinate systems, viewing transformation, Extended Bitmap Option, writing modes, and so forth is essential. Read Chapter 2 when you are ready to begin writing programs. Use Chapters 3 through 8 for reference. When you have become familiar with most of the instructions, you may prefer to use Appendix E for reference.

Documentation Conventions

- The term "CGL" is used in place of "CORE Graphics Library."
- The term "CORE Standard" is used in place of "ACM SIGGRAPH CORE Standard."
- In CORE Graphics Library instruction descriptions, the items in UPPER CASE are to be used exactly as shown. The items in lower case must be replaced by language-specific elements as described.

Associated Professional 300 Series Documentation

- Tool Kit User's Guide

PREFACE

- Terminal Subsystem Manual
- BASIC Reference Manual
- Tool Kit COBOL-81 Documentation Supplement
- Tool Kit DIBOL User's Guide
- Tool Kit FORTRAN-77 Documentation Supplement
- Tool Kit PASCAL User's Guide

CHAPTER 1

OVERVIEW

The Professional 300 Series Terminal Subsystem has two modes: text mode and graphics mode. Text mode is directly accessible to Tool Kit application programs and is described in the Terminal Subsystem Manual. Graphics mode is accessible to application programs only through one of several software development tools.

The Professional 300 Series CORE Graphics Library (CGL from this point on) is a general purpose graphics subroutine package that was designed with two objectives:

- To be compatible with CORE Graphics Standard created by the Association for Computing Machinery (ACM) Special Interest Group on Graphics (SIGGRAPH).
- To make available to the high-level language programmer the powerful functions provided by the Professional 300 Series video bitmap architecture.

This chapter compares CGL to the other graphics programming tools, explains why the CORE Standard was selected as the basis for the Professional 300 series high-level graphics interface, and provides an extensive description of how to use CGL.

1.1 RELATIONSHIP TO OTHER GRAPHICS TOOLS

Other Professional 300 Series graphics software development tools include:

- **PRO/GIDIS (General Image Display Instruction Set)**

PRO/GIDIS is the lowest-level, virtual device interface to the Professional's graphics hardware. Applications that use PRO/GIDIS are optimized for speed and compactness; thus PRO/GIDIS is well-suited to applications like interactive drawing packages, graphics terminal emulators, and

RELATIONSHIP TO OTHER GRAPHICS TOOLS

scientific/engineering data display packages. Both CGL and ReGIS have been implemented with PRO/GIDIS.

Although it will not usually provide the same speed or compactness, CGL provides more functionality than PRO/GIDIS. For example, you can use a single CGL instruction to draw a curve connecting a list of coordinate pairs. With PRO/GIDIS, you must construct the curve yourself, using your own curve interpolation algorithm and lower-level instructions.

NOTE

Curve interpolation can product jagged lines and strange loops in some cases. To correct the problem, add additional points and/or space the points differently.

Applications implemented using CGL are transportable to other CORE implementations with a minimum of conversion effort. PRO/GIDIS is a Digital-specific protocol, and is not easily transportable to other devices or graphics systems.

The application-level interface to CGL is the standard PDP-11 R5 subroutine call. PRO/GIDIS is accessed as a device driver, using the P/OS QIO mechanism to transmit a stream of binary op-codes and parameter data.

- ReGIS (Remote Graphics Instruction Set)

ReGIS is a Digital-developed, ASCII-based protocol used to transmit graphics instructions from a host computer to a remote Professional, or VT125 or GIGI graphics terminal. ReGIS cannot currently be used by applications that reside on the Professional itself; it can only be used when communicating to the Professional over a communications line.

1.2 THE CORE GRAPHICS STANDARD

Most computer applications are designed to create, manipulate, or analyze data. A graphics system is a tool used by an application program to create pictures that show relationships among data.

To accomplish this, the program must describe each part of a picture to the graphics system. The graphics system then creates an image by converting the description from the program into commands that affect the view surfaces (display components) of one or more graphics output devices (see Figure 1-1).

THE CORE GRAPHICS STANDARD

Before 1976, no standard for device-independent graphics systems existed. Although some were very good, none of the existing systems were compatible. A program that used one graphics system would have had to be completely rewritten in order to use another.

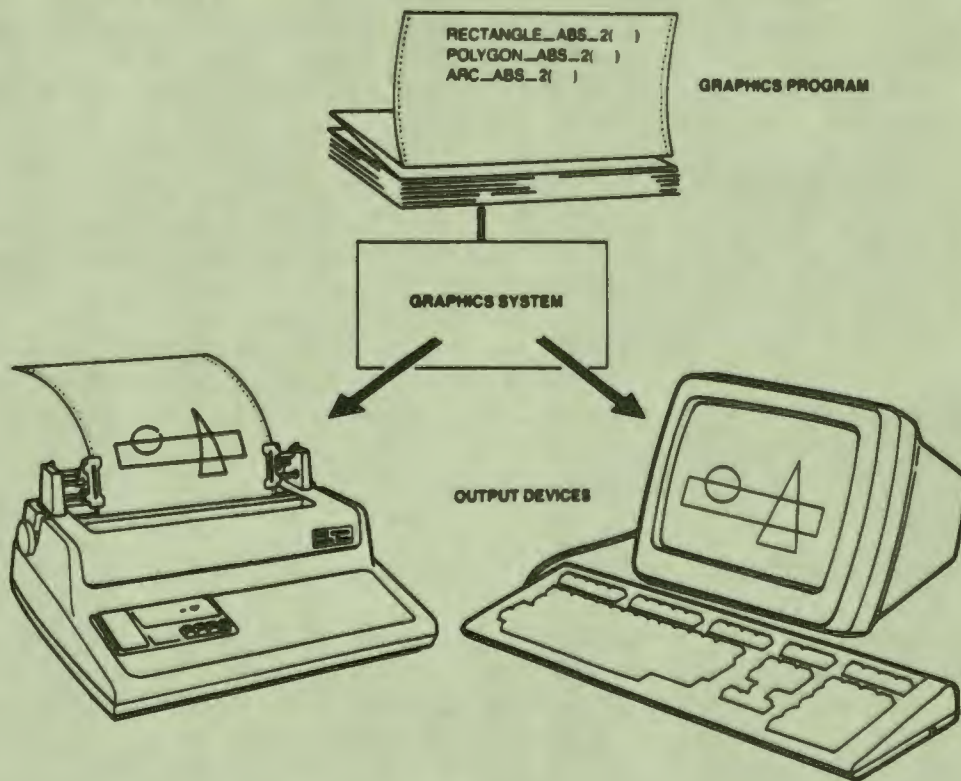


Figure 1-1: A Graphics System

The Association for Computing Machinery (ACM) Special Interest Group on Graphics (SIGGRAPH) recognized the need for a computer graphics standard and, in 1974, began the long process of defining one. Four themes influenced the subsequent design:

1. Portability of programs (and programmers) is the most significant purpose of a standard.
2. The issues that affect portability are those which affect program structure and deserve the most attention.
3. The methodology of both design and use of a standard is as important as its functional capability.

THE CORE GRAPHICS STANDARD

4. The functions of constructing and manipulating an object, and of producing a picture of the object, should be cleanly separated.

The CORE Standard places a great deal of emphasis on portability because factors that limit a program's portability (device dependence, readability, structure,) are the same factors that limit its maintainability. As any software manager knows, a maintainable program is more profitable in the long run. You should strive for portability, even if your programs will never run on anything but a Professional.

The key requirement for program portability is device-independence: the graphics system must make it unnecessary for an application program to contain information specific to a particular graphical output device.

Portability also applies to personnel. A portable programmer can learn to program new or different graphics systems without extensive retraining. The desirability of that characteristic needs no elaboration.

1.3 GRAPHICS PROGRAMMING

The first step in the graphics programming process is to design pictorial representations of the data you want to display. This step is similar to designing a program: you don't begin writing code until you know exactly what you intend to do.

The second step is to write the instructions that create the desired images. CGL has four general types of instructions:

- **Control Instructions**

Control instructions start and stop the graphics system, initialize, select, deselect, and refresh view surfaces, print the screen, and report errors.

- **Viewing Transformation Instructions**

Viewing transformation instructions provide the graphics system with a description of the graphical world (the data manipulated by your program) and control how the graphics system displays it on the current view surfaces.

- **Output Primitive Instructions**

These instructions draw the actual lines, curves, markers, and text that make up images.

GRAPHICS PROGRAMMING

- **Attribute Instructions**

These instructions control colors, styles, modes, and fonts.

The remaining steps are to compile, task build, transfer, install, and test your program on the Professional. CGL requires that you make some edits (see Chapter 2) to the PAB (task builder) command file before task building. The overall development cycle is described in detail in the Developer's Tool Kit User Guide.

1.4 CONTROLLING THE GRAPHICS SYSTEM

CGL requires a minimum of program control. The only requirement is that all programs execute the INITIALIZE_CORE instruction before using any other instructions.

The instructions that control CGL are described in detail in Chapter 3. They are:

- **INITIALIZE_CORE**

The INITIALIZE_CORE instruction allocates the resources used by CGL, initializes and selects the video view surface, sets the default values of the viewing transformation, attributes, color map, and so forth.

- **TERMINATE_CORE**

The TERMINATE_CORE instruction releases the resources used by CGL.

- **NEW_FRAME**

The NEW_FRAME instruction erases currently selected view surfaces. It is recommended that you execute a NEW_FRAME instruction immediately after INITIALIZE_CORE and INITIALIZE_VIEW_SURFACE in order to ensure fresh view surfaces.

- **INITIALIZE_VIEW_SURFACE**

The INITIALIZE_VIEW_SURFACE instruction tells CGL to make a specific view surface ready to accept graphics output. However, CGL does not start sending information to the view surface until you select it. The video monitor is initialized by default.

CONTROLLING THE GRAPHICS SYSTEM

- **TERMINATE_VIEW_SURFACE**

The **TERMINATE_VIEW_SURFACE** instruction terminates access to and releases a specific output device.

- **SELECT_VIEW_SURFACE**

The **SELECT_VIEW_SURFACE** instruction adds a specific device to the set of view surfaces to which CGL performs output. It does not affect the current attribute and viewing transformation values. For example, suppose that you select a view surface, draw an image, deselect that view surface, and select another view surface. The viewport is the same for the second view surface as it was for the first. CGL conveys current state information (except font descriptions) to each view surface when you select it. The video monitor is selected by default.

- **DESELECT_VIEW_SURFACE**

The **DESELECT_VIEW_SURFACE** instruction removes a specific device from the set of selected devices.

- **PLAYBACK_FILE**

The **PLAYBACK_FILE** instruction reads a file of display commands and sends them to all currently selected view surfaces.

- **BEGIN_BATCH**

The **BEGIN_BATCH** instruction begins storing all subsequent view surface updates in a buffer and continues to do so until it executes an **END_BATCH** instruction. Normally, CGL updates the view surfaces each time it executes an instruction. Use of this instruction can considerably improve performance.

- **END_BATCH**

The **END_BATCH** instruction empties the buffer and performs all of the view surface updates that have been stored since the last **BEGIN_BATCH** instruction. CGL no longer stores view surface updates after **END_BATCH** executes.

- **CGL_WAIT**

The **CGL_WAIT** instruction suspends graphics execution, leaving the view surfaces unchanged for a specific number of seconds.

CONTROLLING THE GRAPHICS SYSTEM

- **ERASE_VIEWPORT**

The ERASE_VIEWPORT instruction erases the viewport leaving the remainder of each view surface unchanged.

- **PRINT_SCREEN**

The PRINT_SCREEN instruction sends a specified portion of the image on the video monitor screen to the printer. You can specify horizontal and vertical margins.

- **REPORT_MOST_RECENT_ERROR**

The REPORT_MOST_RECENT_ERROR instruction reports the number of the most recent execution error and the number of the instruction that caused it. It is used primarily for debugging.

1.5 DESCRIBING THE GRAPHICAL WORLD

The graphical world is two-dimensional; we visualize it as a plane. The Cartesian coordinate system provides a convenient way of describing points on a plane. Cartesian coordinates are specified in the format:

X,Y

where X is the horizontal axis and Y is the vertical axis.

A coordinate pair specifies a discrete point on the plane. The finite area of the plane that can be specified by coordinate pairs is called the coordinate space. Figure 1-2 shows a typical coordinate system with axes intersecting at (0,0) and space defined as:

$-1 \leq X \leq 2$ and $-2 \leq Y \leq 1$

The point (-2,1) is outside of the coordinate space.

CGL deals with three different Cartesian coordinate systems:

1. The world coordinate system

Your program uses this coordinate system to represent its database or simply for programming convenience.

DESCRIBING THE GRAPHICAL WORLD

2. The normalized device coordinate system

CGL uses this coordinate system as a device-independent way of describing a view surface.

3. The physical device coordinate system

The Professional terminal subsystem uses this coordinate system to address individual locations on a specific view surface.

The following sections will discuss each coordinate system individually.

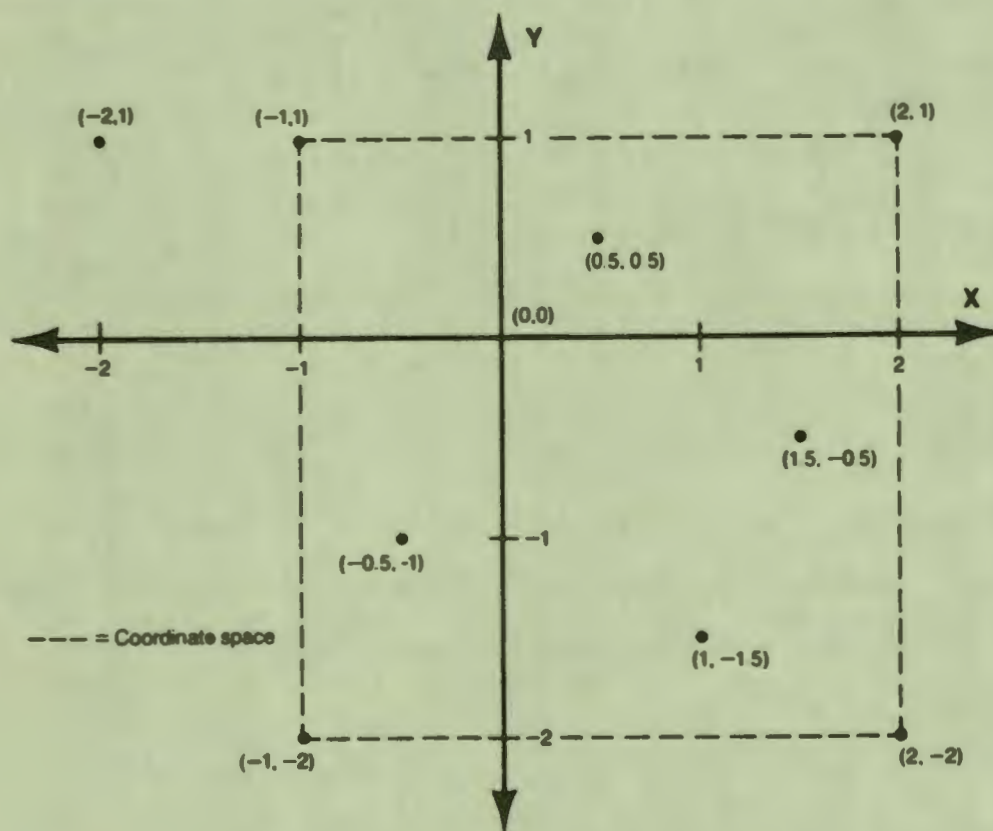


Figure 1-2: A Cartesian Coordinate System

1.5.1 WORLD COORDINATES

Graphical world coordinates (world coordinates from this point on) are device-independent Cartesian coordinates defined by your application program to describe locations and sizes to CGL. You can adjust the graphical world to whatever size and shape is the most convenient.

DESCRIBING THE GRAPHICAL WORLD

If you are working with a database, you can adjust the graphical world to match the data. For example, an application program might deal with sales of amblihelical hexnuts in thousands against time in months, while another program might deal with peaches in bushels against rainfall in inches.

If you are creating visual images, you can adjust the graphical world to match the image. For example, a chess program could draw a chessboard by making the graphical world eight squares by eight squares.

The mapping of world coordinates onto a view surface (or some portion of a view surface) is called the viewing transformation. This is described in detail in Section 1.5.4.

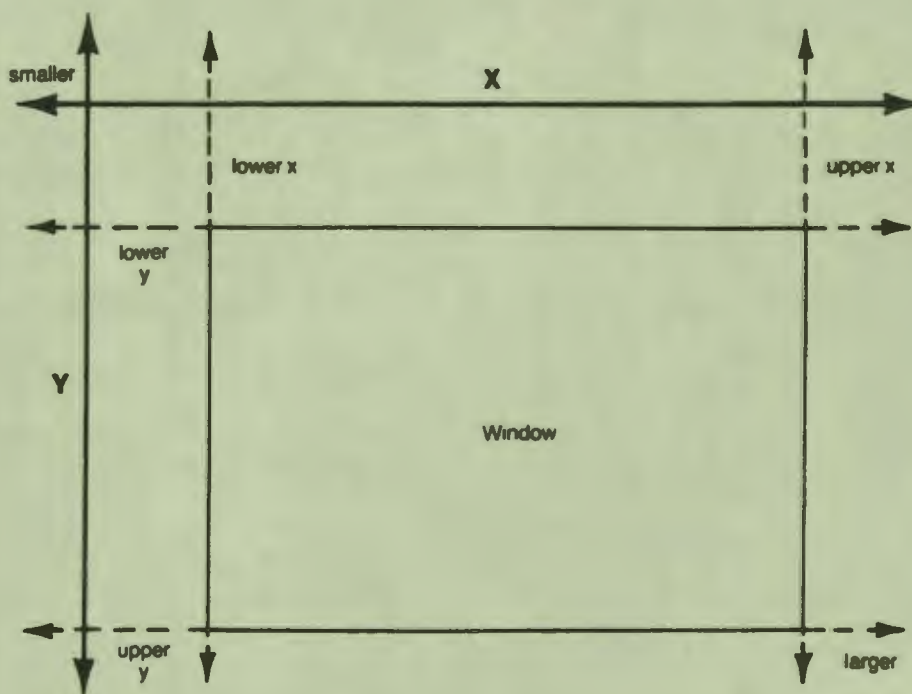


Figure 1-3: The Window In World Coordinate Space

World coordinates can represent any unit of measure. The only requirement imposed by CGL is that world coordinate values must be supplied as real (floating point) numbers. (According to the CORE Standard, most data are available as real numbers.) World coordinate space, then, is bounded only by the set of PDP-11 single-precision real numbers.

DESCRIBING THE GRAPHICAL WORLD

1.5.1.1 THE WINDOW - The `SET_WINDOW` instruction (described in Chapter 4) defines the window, which is the rectangular portion of world coordinate space that is currently used by your program. You provide the lower and upper bounds of the X (horizontal) and Y (vertical) dimensions of the window. Figure 1-3 shows a window in world coordinate space.

The X and Y axes in Figure 1-3 are shown in arbitrary locations and do not necessarily represent zero. The edges of a window can be positive or negative coordinates. Zero on the X or Y axis can be inside or outside of the window.

The origin of the window represents which directions on the view surfaces correspond to increases and decreases in world coordinate values. The origin is defined to be the corner addressed by the smallest world coordinate pair. For example, a window defined as $(-2,3,7,9)$ has as its origin the point $(-2,7)$.

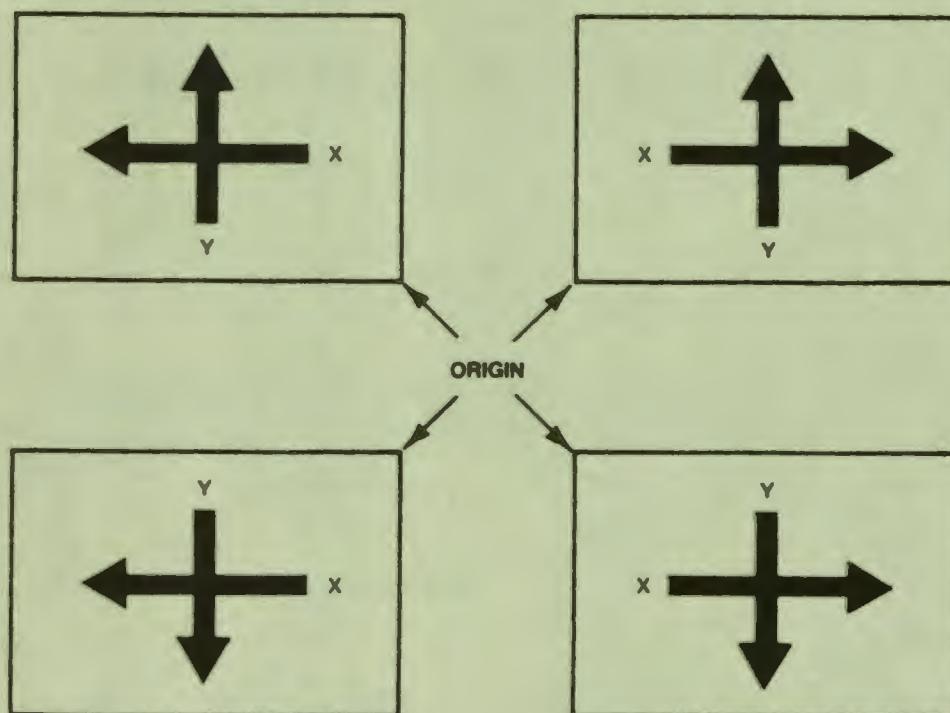


Figure 1-4: The Origin of the Window

If the origin is set to the top-left (the default), X values increase toward the right of the view surfaces and Y values increase toward the bottom. You may decide that the bottom-right is a more convenient origin for your program. By changing the

DESCRIBING THE GRAPHICAL WORLD

origin, you can cause the X value to increase toward the left of the view surfaces and/or the Y value to increase toward the top.

By default, the origin is the top-left corner (the corner that appears at the top-left corner of the view surfaces). The `SET_ORIGIN` instruction (described in Chapter 4) selects any of the four corners as the origin. Figure 1-4 shows the relationship between the origin and world coordinate values.

1.5.1.2 THE CURRENT POSITION - CGL maintains a coordinate pair called the current position that corresponds to the current drawing location in world coordinate space. The visual representation of the current position is the cursor, a symbol that blinks in complement mode. The default cursor is a crosshair symbol. You can use the `SET_CURSOR` instruction (described in Chapter 6) to specify your own cursor.

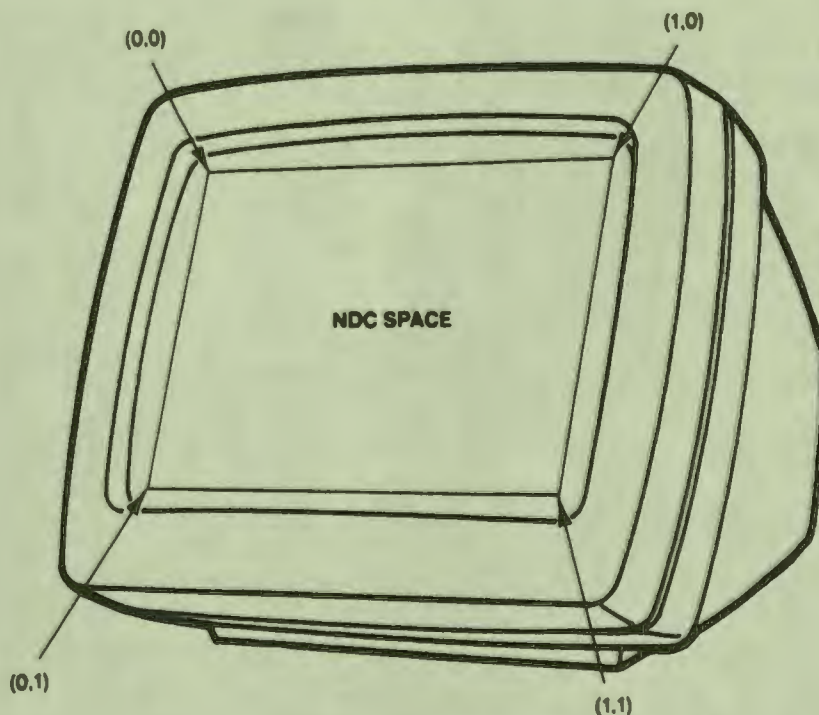


Figure 1-5: Default Normalized Device Coordinate Space

Some output primitive instructions use the current position as the starting position. All output primitive instructions affect the current position (or don't affect it) in a well-defined way. For example, a call to the `LINE_ABS_2` instruction draws a line from the current position to a specified location and makes that

DESCRIBING THE GRAPHICAL WORLD

location the current position. Thus, repeated calls to the `LINE_ABS_2` instruction would result in a set of connected lines. `CGL` also provides `MOVE_ABS_2` and `MOVE_REL_2` instructions to change the current position without drawing anything.

1.5.1.3 ABSOLUTE VS. RELATIVE POSITIONS - You can specify a position in world coordinate space in one of two ways: as an absolute position (independent from the current position) or as a relative position (an offset or displacement from the current position). Output primitive instructions have two versions, "`ABS`" and "`REL`", for absolute coordinates and relative coordinates, respectively.

1.5.2 NORMALIZED DEVICE COORDINATES

Normalized device coordinate (NDC) space is the CORE Standard method of describing the dimensions of any view surface in a device-independent way.

NDC coordinates are real numbers in the range 0 to 1, with default bounds (0,1,0,1) that map to the entire view surface. Your application can specify the upper bounds of NDC space in order to change the aspect ratios of view surfaces.

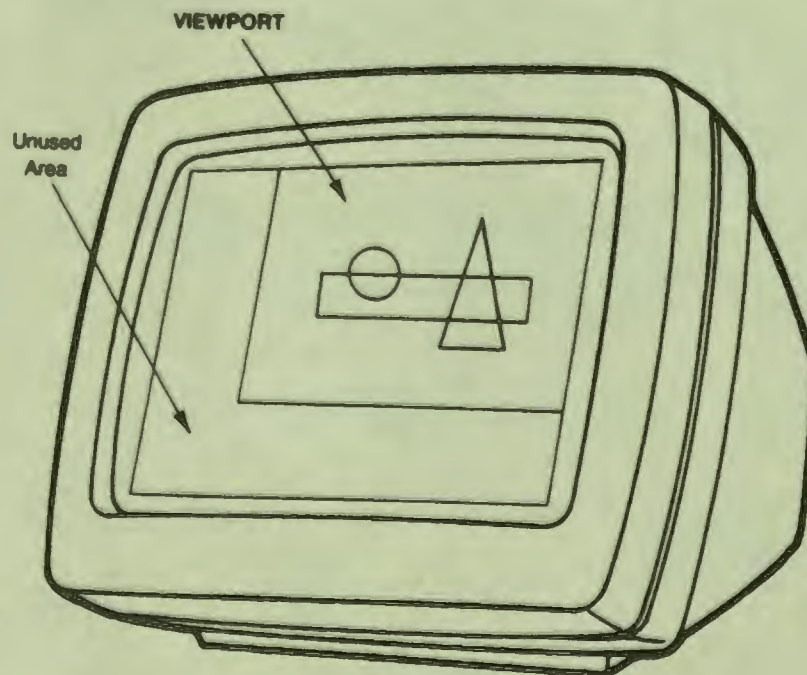


Figure 1-6: The Viewport

DESCRIBING THE GRAPHICAL WORLD

1.5.2.1 NDC SPACE - The default NDC space is rectangular, not square, because its aspect ratio corresponds to that of the Professional video monitor (960 x 600 or 8:5). To change this ratio, use the `SET_NDC_SPACE_2` instruction. Figure 1-5 is a picture of the default NDC space.

1.5.2.2 THE VIEWPORT - Your program can use all of normalized device coordinate space or any rectangular portion of it that you desire. The portion used by your program is called the viewport and is shown in Figure 1-6. The `SET_VIEWPORT_2` instruction (described in Chapter 4) specifies the exact bounds of the viewport.

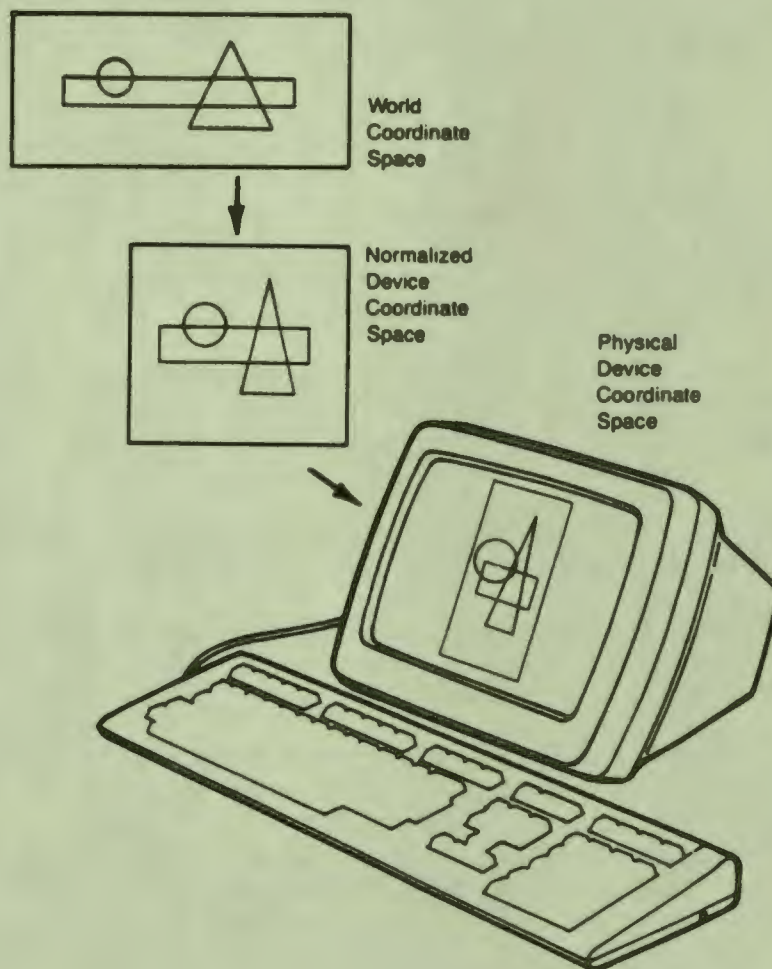


Figure 1-7: The Viewing Transformation

DESCRIBING THE GRAPHICAL WORLD

1.5.3 PHYSICAL DEVICE COORDINATES

Physical device coordinates are device-dependent Cartesian coordinates for specifying positions on the view surface of a particular output device. Each type of output device has its own physical coordinate space. Some CGL instructions accept numeric values where each bit corresponds to one physical device coordinate unit.

The Professional video monitor has a physical device coordinate space of 960 (horizontal) by 600 (vertical) units, a rectangle lying on its side. As a matter of interest, each horizontal device coordinate corresponds to a single pixel (picture element). That is not true of the vertical coordinates; there are only 240 vertical pixels. The mapping of physical device coordinate units into pixels is a function of the terminal subsystem.

1.5.4 THE VIEWING TRANSFORMATION

The process of creating an image on a view surface can be thought of as a three-step process, as shown in Figure 1-7.

1. CGL (optionally) clips the world coordinate objects to be viewed so that the portions that would fall outside the window are removed from view. The `SET_WINDOW_CLIPPING` instruction controls this function.
2. CGL maps the contents of the window (world coordinates) to the viewport (normalized device coordinates).
3. CGL maps the contents of the viewport (normalized device coordinates) to each currently selected view surface (physical device coordinates).

The viewport can have any aspect (X to Y) ratio you wish. If the aspect ratio of the window does not match the viewport, CGL "squeezes" or "stretches" the window to fit. Changing the viewport affects different output primitives in different ways. Some of these effects are controlled by CGL; others are characteristics of the terminal subsystem.

- Straight lines and arcs appear as you would expect. Arcs retain their shape but not their size (a circle does not transform to an ellipse).

DESCRIBING THE GRAPHICAL WORLD

- Curved lines vary somewhat in shape, depending on the physical device coordinate positions available to draw them.
- Text (character size and spacing) is adjusted to fit the required number of characters into the viewport.

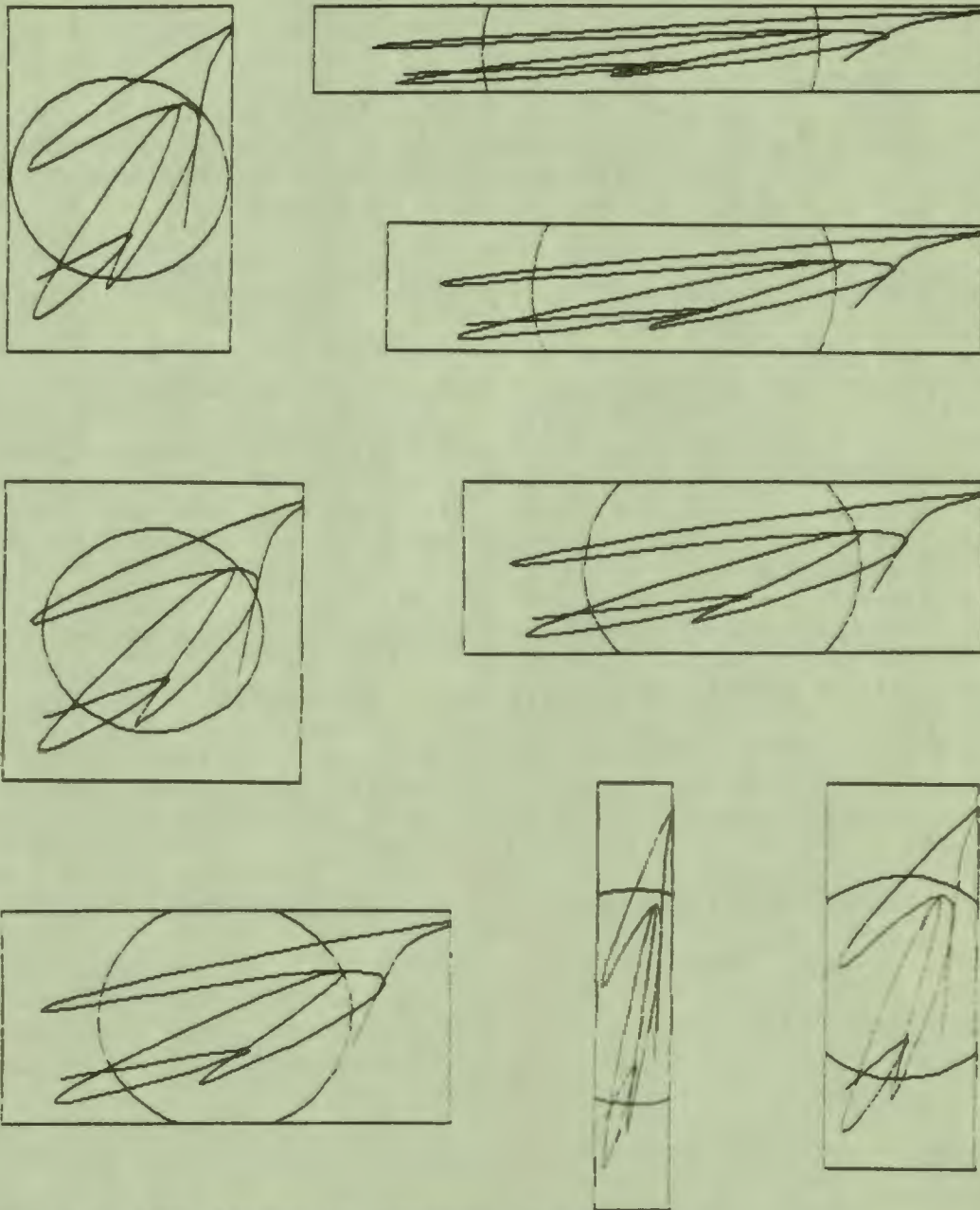


Figure 1-8: One Image in Several Viewports

Figure 1-8 shows the image of a rectangle (drawn along the edges of the viewport), a circle (360 degree arc) and a curve drawn in several different viewports. The circle remains round, regardless of the shape of the viewport.

OUTPUT PRIMITIVES - THE GRAPHICAL "BUILDING BLOCKS"

1.6 OUTPUT PRIMITIVES - THE GRAPHICAL "BUILDING BLOCKS"

Output primitives are the fundamental visible images that you combine to make up pictures. You can draw straight lines, curved lines, markers, and text, or just move the current position.

Your application program creates output primitives by calling CGL instructions (described in detail in Chapters 6 and 7). Output primitive instruction names end in "ABS 2" (absolute) or "REL 2" (relative) which specifies whether the parameters are absolute positions or offsets. The suffix "2" is the CORE Standard syntax for "two dimensional" and is included for compatibility with future software that may support three dimensional output primitives.

The way output primitives appear on the view surfaces is determined by:

- The parameters passed with the instruction call

Most of the parameters passed with output primitive instructions specify where in world coordinate space to draw the output primitive. If the parameters are absolute coordinates, you specify either a single coordinate pair or an array containing a list of coordinate pairs. If the parameters are relative coordinates, you specify either a single offset or an array containing a list of offsets.

- The current global attribute value settings

The global attributes (which affect all output primitives) are: the writing index, the background index, the writing planes, and the writing mode. These are explained in Section 1.7.6.

- The current primitive-specific attribute value settings

Each type of output primitive has a set of attributes that determine style, color, and so forth. For example, line primitives have a special attribute called fill that you can use to draw "solid" objects. These are explained in Section 1.7.

- The viewing transformation

The shapes of the window and the viewport affect the way output primitives appear. If the shapes are different, CGL "squeezes" or "stretches" the window to fit the viewport. The distortion of the window affects different output primitives in different ways and is described in Section 1.5.4

OUTPUT PRIMITIVES

1.6.1 CURRENT POSITION INSTRUCTIONS

Current position instructions cause no change to the view surfaces. They simply change or report on the value of the current position.

- **MOVE_ABS_2**

The **MOVE_ABS_2** instruction changes the current position to a point specified as an absolute position in world coordinate space.

- **MOVE_REL_2**

The **MOVE_REL_2** instruction changes the current position to a point specified as an offset in world coordinate space.

- **INQUIRE_CURRENT_POSITION_2**

The **INQUIRE_CURRENT_POSITION_2** instruction returns the current position in world coordinate space.

- **SET_CURSOR**

The **SET_CURSOR** instruction controls the appearance of the cursor, the visual representation of the current position.

1.6.2 MARKER PRIMITIVE INSTRUCTIONS

These instructions change the current position and draw markers or series of markers. Markers are symbols such as dots or bullets that represent points in world coordinate space. They appear on the view surfaces centered on the new current position.

- **MARKER_ABS_2**

The **MARKER_ABS_2** instruction draws a character at a point specified as an absolute position in world coordinate space.

- **MARKER_REL_2**

The **MARKER_REL_2** instruction draws a character at a point specified as an offset in world coordinate space.

- **POLYMARKER_ABS_2**

The **POLYMARKER_ABS_2** instruction draws a character at each of a list of points specified as absolute positions in world coordinate space.

OUTPUT PRIMITIVES

- **POLYMARKER_REL_2**

The **POLYMARKER_REL_2** instruction draws a character at each of a list of points specified as offsets in world coordinate space.

1.6.3 LINE PRIMITIVE INSTRUCTIONS - THE GRAPHICAL "PEN"

These instructions draw one or more lines. You supply the point(s) that describe the line(s) that you want to draw.

These instructions draw straight lines:

- **LINE_ABS_2**

The **LINE_ABS_2** instruction draws a straight line from the current position to a point specified as an absolute position in world coordinate space.

- **LINE_REL_2**

The **LINE_REL_2** instruction draws a straight line from the current position to a point specified as an offset in world coordinate space.

- **POLYLINE_ABS_2**

The **POLYLINE_ABS_2** instruction draws a series of lines from the current position to a list of points specified as absolute positions in world coordinate space.

- **POLYLINE_REL_2**

The **POLYLINE_REL_2** instruction draws a series of lines from the current position to a list of points specified as offsets in world coordinate space.

- **POLYGON_ABS_2**

The **POLYGON_ABS_2** instruction draws a series of lines connecting a list of points specified as absolute positions in world coordinate space.

- **POLYGON_REL_2**

The **POLYGON_REL_2** instruction draws a series of lines connecting a list of points specified as offsets in world coordinate space.

OUTPUT PRIMITIVES

- **RECTANGLE_ABS_2**

The **RECTANGLE_ABS_2** instruction draws a series of lines forming a four-sided, perpendicular, polygon with the current position at one corner and a point specified as an absolute position in world coordinate space at the other.

- **RECTANGLE_REL_2**

The **RECTANGLE_REL_2** instruction draws a series of lines forming a four-sided, perpendicular, polygon with the current position at one corner and a point specified as an offset in world coordinate space at the other.

These instructions draw curved lines by a process called "interpolation." CGL computes the shape of the curve from the supplied points and provides the missing points.

- **ARC_ABS_2**

The **ARC_ABS_2** instruction draws a section of a circle based on absolute positions in world coordinate space.

- **ARC_REL_2**

The **ARC_REL_2** instruction draws a section of a circle based on offsets in world coordinate space.

- **CURVE_ABS_2**

The **CURVE_ABS_2** instruction draws a smooth curve through a list of points specified as absolute positions in world coordinate space.

- **CURVE_REL_2**

The **CURVE_REL_2** instruction draws a smooth curve through a list of points specified as offsets in world coordinate space.

1.6.4 TEXT PRIMITIVE INSTRUCTIONS

Graphics text is independent from and more flexible than the text available when the Terminal Subsystem is in text mode. Although it is possible to have both output primitives and text mode text on a view surface at the same time, it is recommended that you use only one at a time. The aspects of using text mode and graphics simultaneously are discussed in the Terminal Subsystem Manual.

OUTPUT PRIMITIVES

- **TEXT**

The TEXT instruction draws a line of graphical text.

- **INQUIRE_TEXT_EXTENT_2**

The INQUIRE_TEXT_EXTENT_2 instruction does not draw anything. It reports the amount of world coordinate space that would be used to draw a string of a specified length.

- **LOAD_FONT**

The LOAD_FONT instruction loads multiple characters into the current user-defined font from a region in memory created by your application. This is much faster than loading individual characters.

- **LOAD_CHARACTER**

The LOAD_CHARACTER instruction loads a character description into the current user-defined font. You provide the character description in the form of an array of integers. Each integer in the array describes a horizontal row of 16 physical device coordinate positions. Each set bit specifies an "on" position and each clear bit specifies an "off" position (see Section 1.7.4.3). The first element describes the "top" row of the character; the next element describes the next row; and so forth.

In order to set the bits in each of the elements, a program called a "font editor" is very useful. The algorithm for a simple font editor is shown in the example program "FONT" in Appendix D. It reads font description data from a terminal-format file and performs a string-to-integer conversion for each line in the array. You can use an ordinary text editor to create the terminal-format file.

- **BEGIN/END_DEFINE_CHARACTER**

You can load a character by using the following sequence of instructions:

OUTPUT PRIMITIVES

BEGIN_DEFINE_CHARACTER

: : :

output primitives and attributes

: : :

END_DEFINE_CHARACTER

The output primitives and attributes describe the character to be loaded. World coordinates and attribute sizes are mapped to the character dimensions specified in SET_FONT_SIZE.

1.7 ATTRIBUTES - CONTROLLING THE WAY OUTPUT PRIMITIVES LOOK

Attributes are characteristics of appearance, color, style, mode, width, and so forth. Attribute values stay the same until they are explicitly changed. For example, the default line style is SOLID. If you change the line style to DASHED, all subsequent lines will be drawn DASHED until you change it again.

Each type of output primitive has a set of unique attributes. For example, line attributes have no effect on text primitives. The "global" output primitive attributes are the writing index and the writing mode.

The background has an attribute called the background index. The background is defined to be all areas of the view surfaces not covered by the image of an output primitive. Some programmers think of the background on a video monitor as a permanent output primitive that fills the entire screen.

1.7.1 LINE ATTRIBUTES

The line primitives (line, polyline, polygon, rectangle, arc, and curve) have three attributes. The instructions that set them are:

- SET_LINestyle

The line style is the pattern used to draw lines, except when fill is enabled. You can use a standard line style or specify your own. The standard styles are: SOLID, DASHED, DOTTED, and combinations and variations of the above.

ATTRIBUTES

- **SET_LINEWIDTH**

The line width is the width of line primitives in world coordinate units. You can control the vertical and horizontal line width independently. The width of a diagonal line varies according to the angle in which it is drawn (see Figure 7-3). The line width is "squeezed" or "stretched" in the viewing transformation; thus you should adjust it accordingly.

- **SET_LINEWIDTH_ORIENTATION**

The line width orientation controls the way CGL draws the ends of lines. You can think of the end of a line as a rectangle described by the vertical and horizontal line width (see Figure 7-3). You can control the offset between one corner of that rectangle and the point in world coordinate space specified as the starting position in the line primitive instruction.

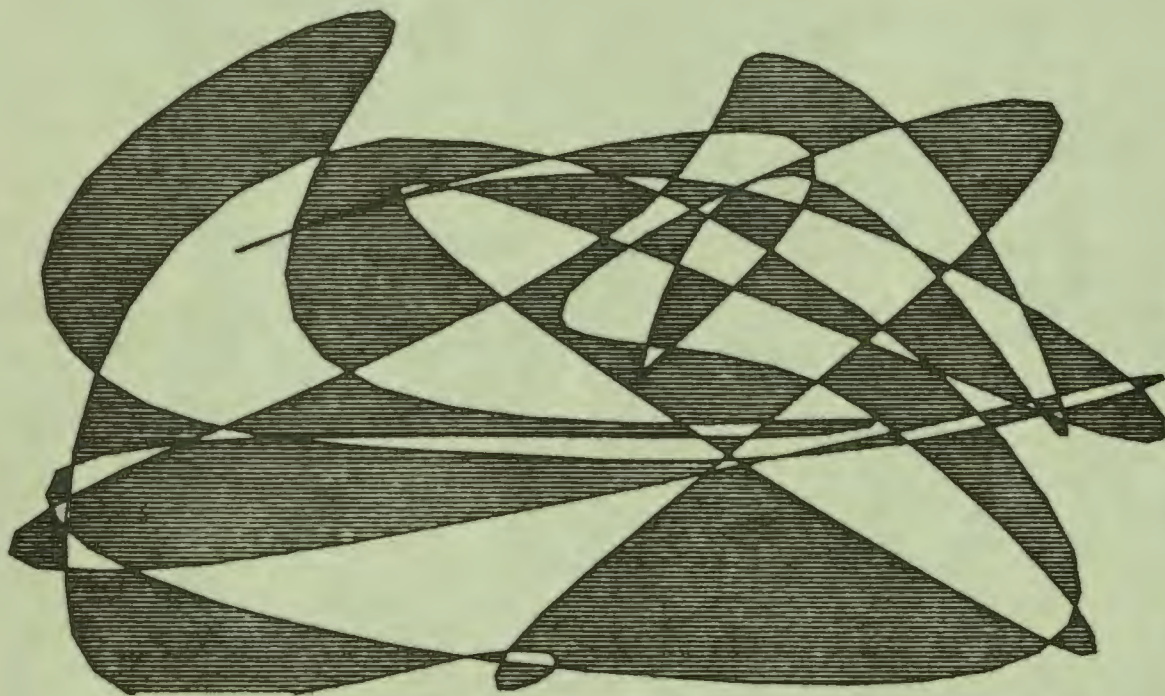


Figure 1-9: Polygon Fill (Drawn on HP7470 Plotter)

ATTRIBUTES

• SET_FILL_MODE

Fill creates solid images by filling in areas with a pattern consisting of a standard or user-defined character. When fill is off, CGL draws lines are drawn using the linestyle. When fill is on, CGL does not actually draw lines. Instead, it causes the space described by a line primitive to be "shaded" or "flooded" with the fill pattern.

If the line describes a closed area such as a polygon, rectangle, closed arc, or closed curve, CGL (in polygon fill mode) "shades" the area with the fill pattern. Figure 1-9 shows a random curve drawn with polygon fill.

If the line primitive describes an open area, CGL "shades" the area between the undrawn line and a predefined entity. The entity can be a horizontal line, a vertical line, or a point. Figure 1-10 shows the open area fill modes.

Fill patterns are self-aligning. When two adjacent or overlapping areas are filled, the patterns align "seamlessly."

• SET_FILL_ENTITY

The fill entity specifies a reference for filling open areas. The reference can be a horizontal line, a vertical line, or a point.

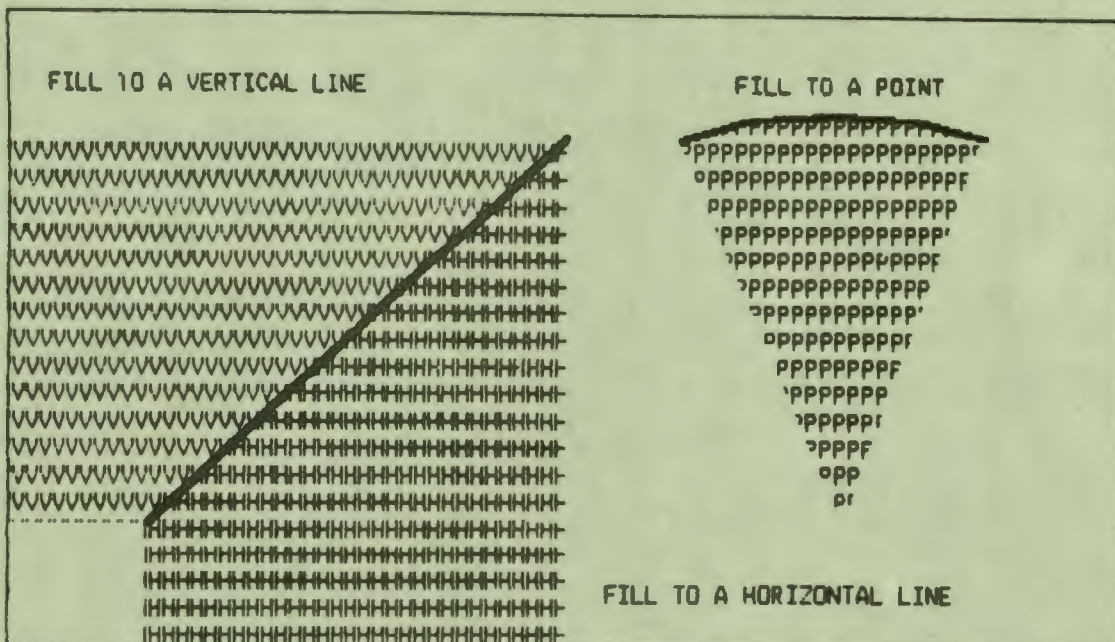


Figure 1-10: Open Area Fill Modes

ATTRIBUTES

- **SET_FILL_CHAR**

The fill character specifies the character (from any font) used for area fill. You can change the size of the fill character by specifying a multiplier on the character height and/or width.

The default fill character is a special case; in fact, it's not a character at all. Character zero (the default) causes CGL to use a vertically-oriented version of the current line style, rather than a character. The default line style is solid, thus the default fill is also solid.

1.7.2 MARKER ATTRIBUTES

The marker primitives (marker and polymarker) have one attribute. The instruction that sets it is:

- **SET_MARKER_SYMBOL**

The marker symbol is the character used to draw markers. You can use a standard symbol or any other character. The (CORE-defined) standard symbols are: period, plus sign, asterisk, upper-case O, and upper-case X.

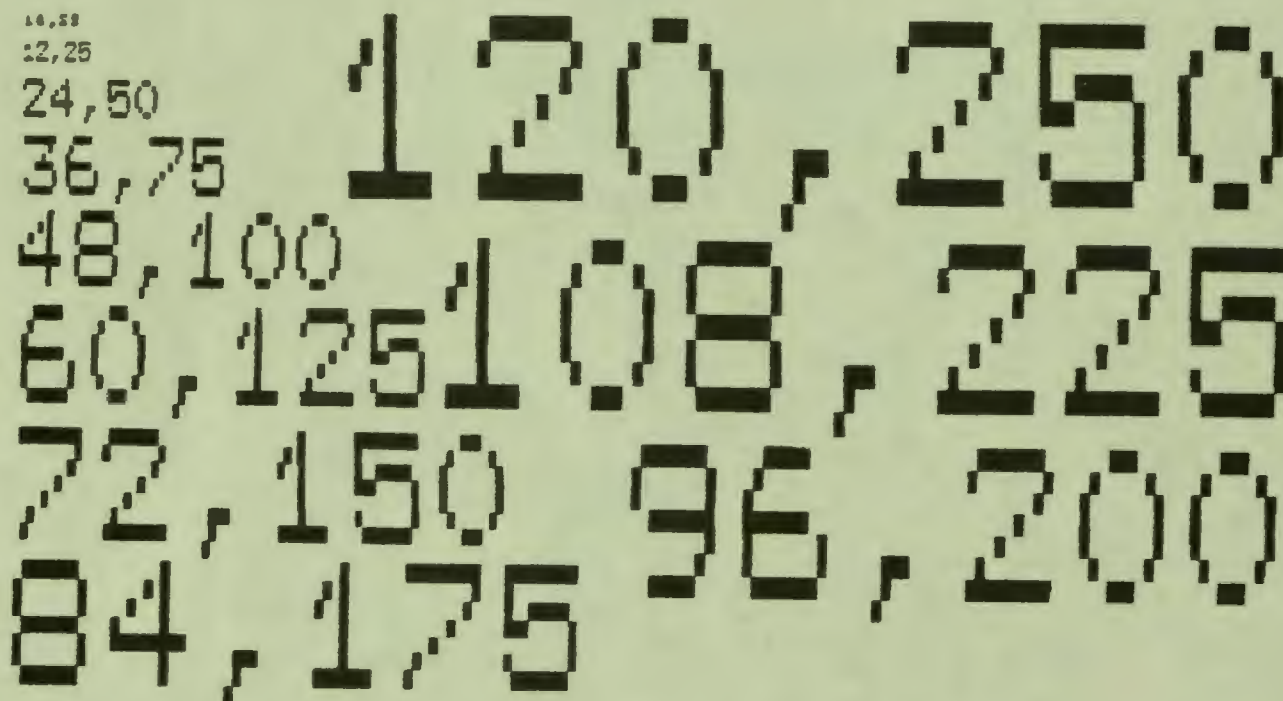


Figure 1-11: Character Size

ATTRIBUTES

For example, if the window is ten world coordinate units wide and you specify a character width of one unit, CGL selects the largest available size that will fit ten characters to a line. If you specify a character size of two units, CGL selects the largest size that will fit five characters to a line.

The "standard" size for Font 0 characters is 12 X 25 physical device coordinate units. The video monitor screen is 960 X 600 physical device coordinate units, thus it can contain 24 rows of 80 characters. The "standard" size for user-defined characters varies from one to 16 physical device coordinates in both height and width (see the SET_FONT_SIZE instruction).

The SET_CHARPATH and SET_CHARITALIC instructions can change the available character sizes. character sizes. Characters drawn in a horizontal path are smaller than those drawn diagonally and larger than those drawn vertically. Characters drawn with a slant are larger than those with no slant.

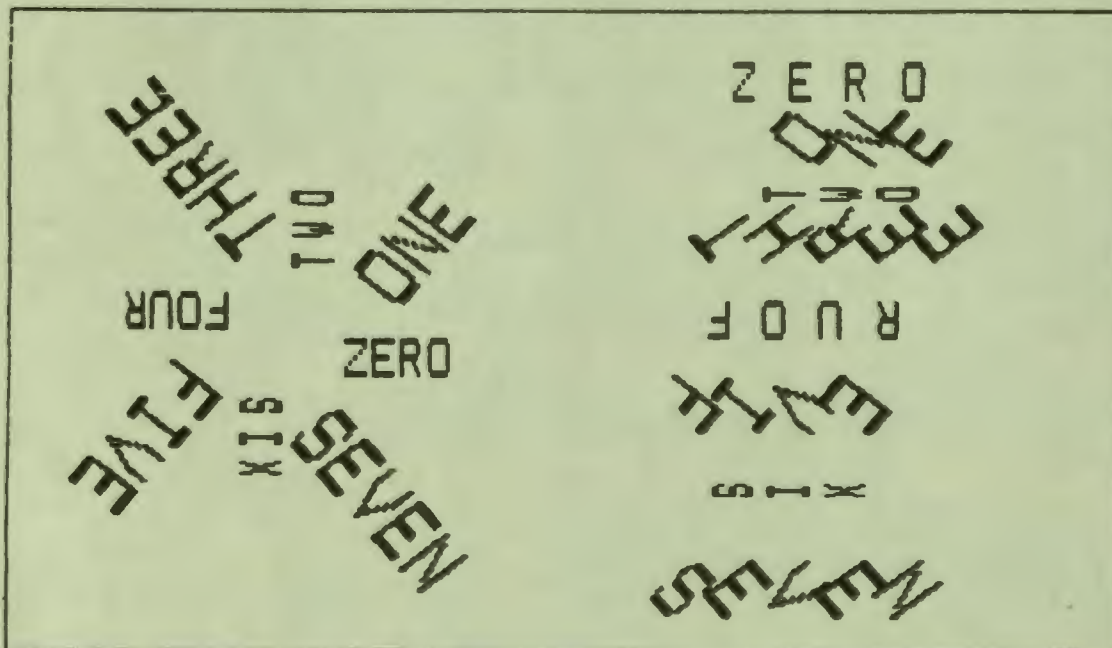


Figure 1-13: Character Path

ATTRIBUTES

- **SET_CHARSPLACE**

The `SET_CHARSPLACE` instruction specifies the displacement between the starting points of adjacent letters. The displacement can be horizontal, vertical, or both. Figure 1-12 shows some examples of character spacing.

The `SET_CHARSPLACE` instruction affects the relative position (not the direction) of individual characters in a string. The direction is specified by the `SET_CHARPATH` instruction.

`CGL` modifies the character spacing to maintain string rotation when the character path mode is "string."

- **SET_CHARPATH**

Character path is the direction in which text is drawn. It can apply to individual characters or to entire strings. There are eight possible directions as shown in Figure 1-13.

The `SET_CHARPATH` instruction has two modes: character and string.

In character mode, `SET_CHARPATH` rotates (changes the direction relative to horizontal) the individual characters in a text string, and sets the spacing to that explicitly defined by the last `SET_CHARSPLACE` call.

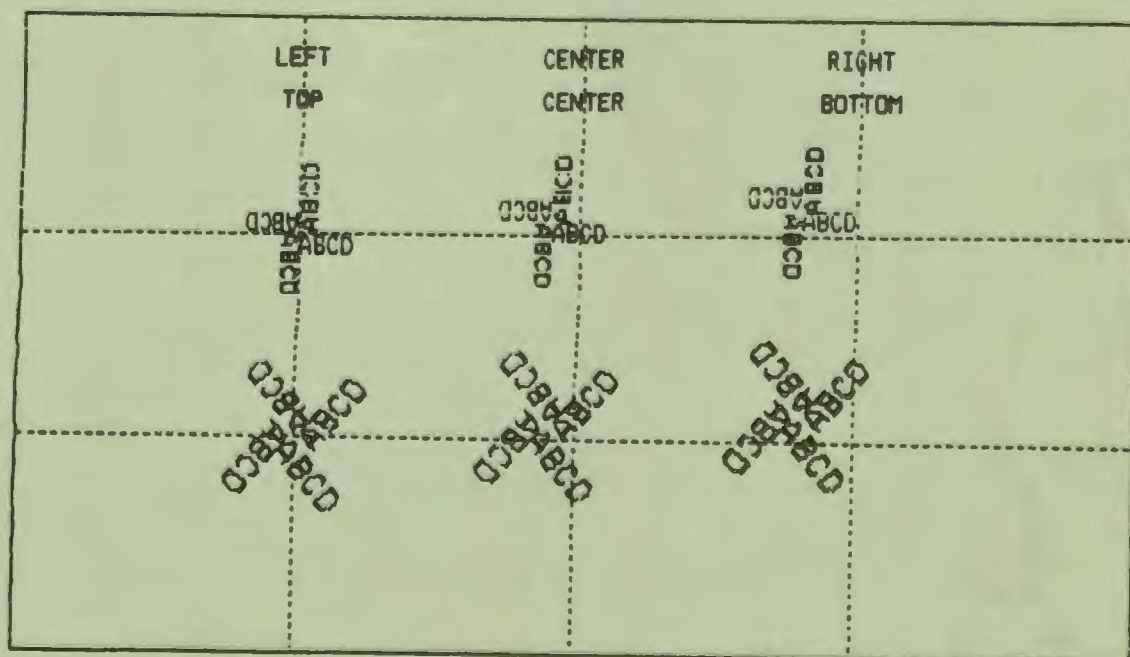


Figure 1-14: Character Justification

ATTRIBUTES

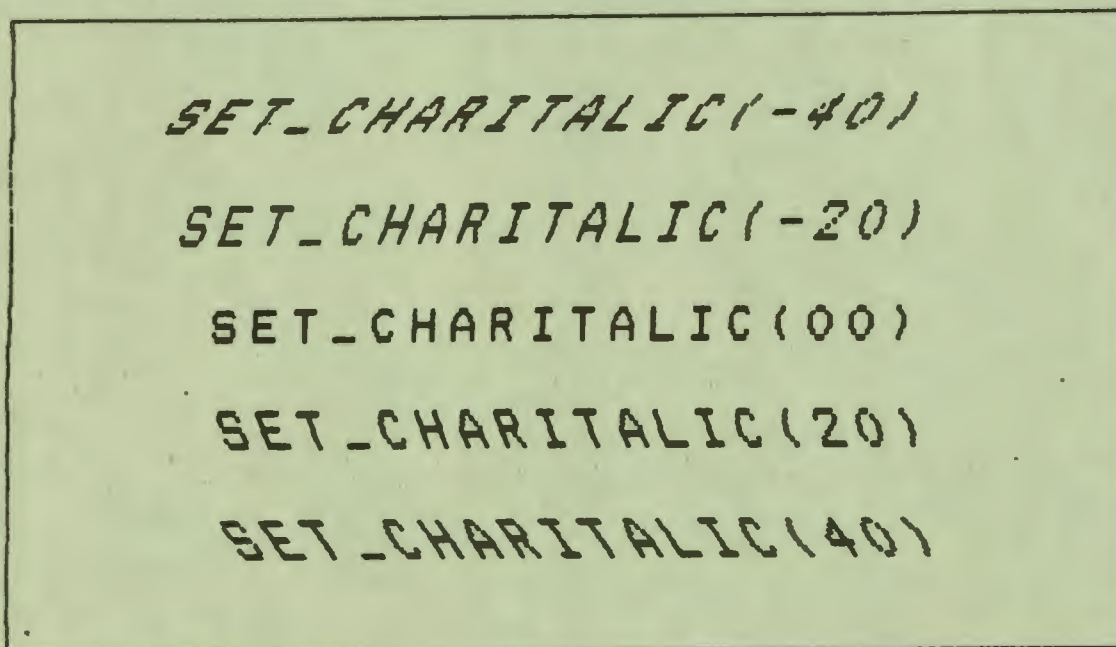
In string mode, (the default) `SET_CHARPATH` rotates the entire string to the specified direction by changing the character spacing and the direction of the individual characters.

`SET_CHARPATH` can be used in conjunction with `SET_CHARSPLACE` to create virtually any desired combination of direction and spacing. Subsequent calls to `SET_CHARSPLACE` cause CGL to modify the spacing so as to maintain string rotation. Likewise, subsequent calls that set the `CHARPATH` to character mode also explicitly set the character spacing to that last specified.

- `SET_CHARJUST`

The `SET_CHARJUST` instruction specifies the starting position of text primitives relative to the current position. It allows horizontal and vertical justification and centering. Figure 1-14 shows some examples of character justification.

Justification and centering are defined in terms of horizontal text drawn left to right. Justification means that an edge of the string is aligned along an X (horizontal) or Y (vertical) line. The edges of a string are the top, the bottom, the leftmost side of the first character, and the rightmost side of the last character. Centering means that the string is bisected exactly by an X or Y line.



```
SET_CHARITALIC(-40)  
SET_CHARITALIC(-20)  
SET_CHARITALIC(00)  
SET_CHARITALIC(20)  
SET_CHARITALIC(40)
```

Figure 1-15: Character Italic

ATTRIBUTES

If the character spacing is not horizontal, CGL computes the position at which it would draw the first character of a horizontal string and draws the first character of the string at that position. The character spacing then determines the position of the second and subsequent characters.

- **SET_CHARITALIC**

Character italic is a forward or backward slant that makes characters in a text string resemble italic type. Figure 1-15 shows some examples of character italic.

- **SET_FONT**

The **SET_FONT** instruction specifies the current font. Font 0 contains the DEC Multinational Character Set (except for C0, C1, and the delete character) which has 190 "printing" characters and cannot be redefined (see Figure 1-16). Fonts 1, 2, and 3 are user-defined fonts that can each contain up to 190 characters that you load yourself.

0	Q	P	'	p	•	À	?	à	?
1	A	Q	a	q	±	Á	Ñ	á	ñ
2	B	R	b	r	¢	Â	Ò	â	ò
3	C	S	c	s	£	Ã	Ó	ã	ó
4	D	T	d	t	¥	Ä	Ô	ä	ô
5	E	U	e	u	¥	Å	Õ	å	õ
6	F	V	f	v	¥	Æ	Ö	æ	ö
7	G	W	g	w	¥	Ç	È	ç	è
8	H	X	h	x	¥	È	É	é	ê
9	I	Y	i	y	¥	É	Ú	é	ú
:	J	Z	j	z	¥	Ê	Û	ê	û
,	K	[k	[¥	«	»	ë	ü
<	L	\	l	l	¥	?	¼	ì	ü
=	M]	m]	¥	?	½	í	ÿ
>	N	^	n	^	¥	?	¾	î	?
/	?	_	o	i	¥	?	?	ï	?

Figure 1-16: Font 0

ATTRIBUTES

• SET_FONT_SIZE

The `SET_FONT_SIZE` instruction initializes a user-defined font. It establishes the size of the font (the highest decimal character code) and the size of the characters in physical device coordinate units.

When you execute `SET_FONT_SIZE`, CGL passes the font size and subsequent character definitions to all currently selected view surfaces. If a view surface is not selected at the time the font is defined, it cannot access the font.

Thus, in theory, you can have different fonts, with characters of different aspect ratios, simultaneously defined for different view surfaces.

1.7.4 COLOR - THE BITMAP ARCHITECTURE

The terminal subsystem has an internal data structure called the bitmap that stores the information currently being displayed on the view surfaces. The bitmap consists of one or three planes. Each plane, in terms of a high-level language data structure, is a two-dimensional array of bits. Each bit corresponds to one physical device coordinate position and represents some information about brightness or color. The basic Professional (with no Extended Bitmap Option) has only one bitmap plane and thus can display only monochromatic images. The value of each bit represents a light or dark point on the screen. The actual color of a monochromatic image depends on the phosphor used in the monitor.

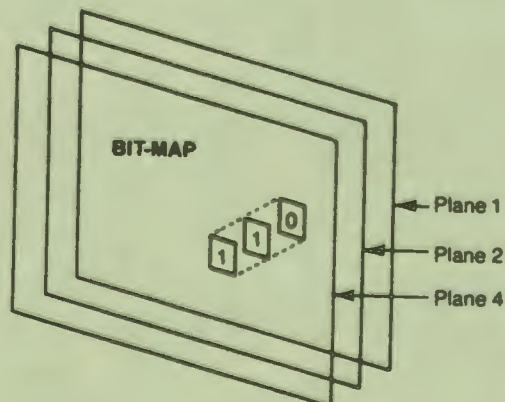


Figure 1-17: The Bitmap with Extended Bitmap Option

ATTRIBUTES

The Extended Bitmap Option (shown in Figure 1-17) provides two additional bitmap planes, making a total of three planes available to your program. The planes are numbered 1, 2, and 4. Each plane doubles the number of colors available to your program at any given time.

With one plane, there are only two colors: dark and light. Two planes provide four colors. Three planes provide eight colors.

In order to use the full color graphics capabilities of the Professional, an output device with appropriate capabilities, such as color video monitor or multi-pen plotter is required. The Extended Bitmap Option with a monochrome output device can simulate colors with varying shades of lightness.

The Extended Bitmap Option also provides a data structure called the color map. In order to understand how the bitmap and the color map work together to produce colors on the view surfaces, consider how colors in light are formed.

1.7.4.1 HOW COLORS ARE FORMED - The Professional forms colors by addition. Red, green, and blue, the primary colors in light, can be added together in various proportions to approximate any color of the spectrum.

The red and blue primary colors of light are similar, but not identical, to the red and blue primary colors of paint. In light, the blue primary is less green and the red primary is more orange. Equal amounts of the light primaries can be combined to form white light.

Complementary colors in light are any two colors that form white light when combined. The three most important complementary colors can be formed by combining primaries:

- Cyan (the complement of red) is formed by combining green and blue.
- Magenta (the complement of green) is formed by combining red and blue.
- Yellow (the complement of blue) is formed by combining red and green.

You can combine any one of these three complementary colors with the third primary color to produce white light. For example, yellow added to blue forms white.

ATTRIBUTES

1.7.4.2 THE COLOR MAP - The Extended Bitmap Option provides an internal data structure called the color map that has eight entries, each of which represents a color available to your program. Figure 1-18 shows a picture of the color map.

Each color map entry consists of three values, one each for red, green, and blue. These "RGB" values specify how much of each primary color is used to form a color. RGB values have the range zero to seven. Zero is the minimum amount of color and seven is the maximum.

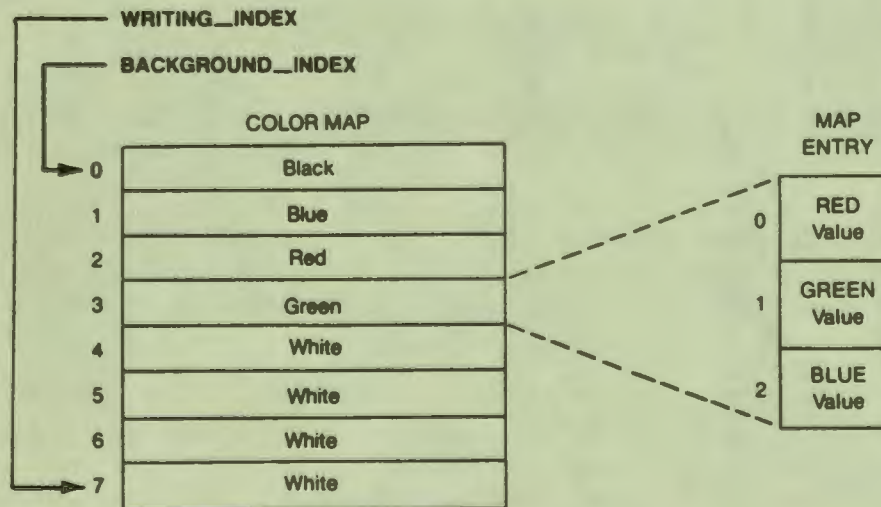


Figure 1-18: The Color Map with Default Values

Each entry in the color map is eight bits wide; three bits for red, three for green but only two for blue, since it is difficult for the human eye to distinguish shades of blue. The blue value range is only (0, 2, 4, 6). Blue values are rounded down to the next lowest even number.

Thus, a color map entry can contain one of 256 ($8 * 8 * 4$) possible colors ranging from black (0,0,0) to white (7,7,6).

CGL provides two instructions that access the color map:

- **SET_COLOR_MAP**

The SET_COLOR_MAP instruction sets up the RGB values of the entire color map.

ATTRIBUTES

- **SET_COLOR_MAP_ENTRY**

The **SET_COLOR_MAP_ENTRY** instruction sets up the RGB values of an individual entry in the color map.

NOTE

When you change the values in a color map entry, you instantaneously change the color of any image on the video monitor screen that was drawn with that entry. This dynamic screen update feature can be used in some very sophisticated ways.

CGL provides two global color attributes. The instructions that set them are:

- **SET_BACKGROUND_INDEX**

This specifies the color map entry generally used to indicate the absence of an image.

- **SET_WRITING_INDEX**

This specifies the color map entry generally used to indicate the presence of an image.

These definitions say "generally" because the exact manner in which CGL draws output primitives depends on the writing mode (described in Section 1.7.5).

The background index and the writing index do not actually specify a color; they specify a color map entry. The color depends on the values stored in the color map.

For example, suppose that you want to draw a yellow circle. If one of the color map entries contains the desired color, just set the writing index to that entry. Otherwise, choose a color map entry and set its red, green, blue values to the desired color, then set the writing index to that entry.

CGL provides the following (VT125 compatible) default color map values:

ATTRIBUTES

Entry	Color	R	G	B	Entry	Color	R	G	B
0	black	0	0	0	4	white	7	7	7
1	blue	0	0	7	5	white	7	7	7
2	red	7	0	0	6	white	7	7	7
3	green	0	7	0	7	white	7	7	7

If only a monochrome monitor is present, CGL uses the following formula to convert RGB values to shades of grey:

$$\text{grey value} = ((R * 2) + (G * 4) + B) / 7$$

Thus, grey values also fall in the range zero to seven.

1.7.4.3 THE BITMAP/COLOR MAP INTERFACE - With the Extended Bitmap Option present, the values stored in the three bitmap planes form a three-bit binary number (decimal value 0 to 7). This number, points to one of the entries in the color map. The RGB values in that entry determine the color of the physical device coordinate location controlled by that three-bit number.

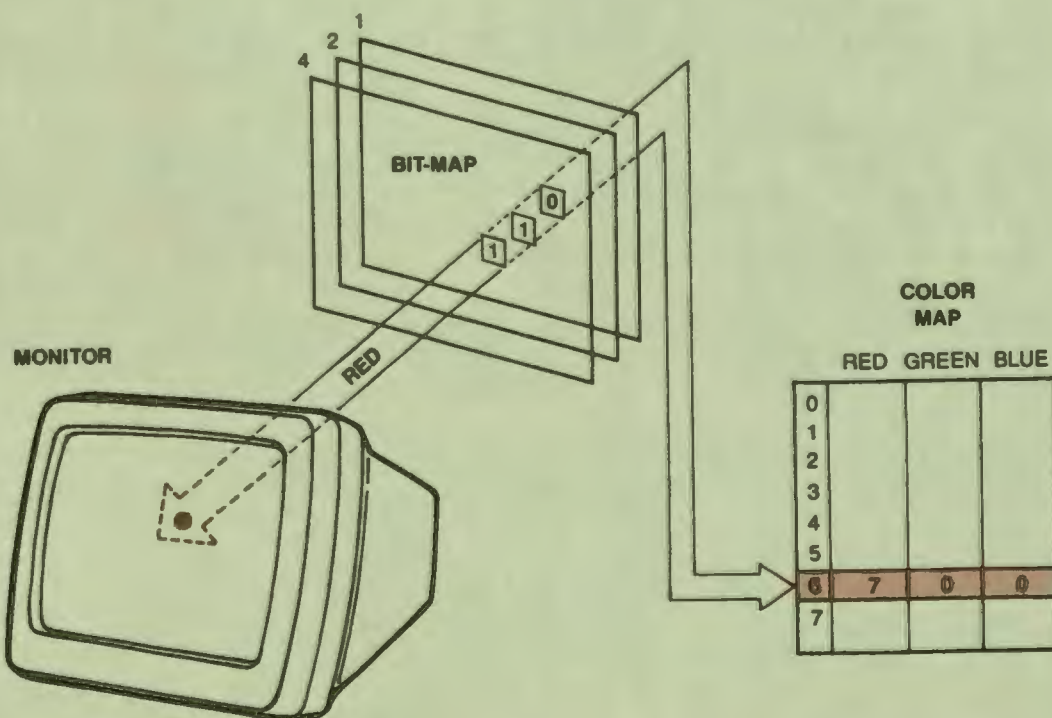


Figure 1-19: The Color Map/Bitmap Interface

ATTRIBUTES

Figure 1-19 shows the relationship between the bitmap and the color map. The three bitmap planes form the number 110 (binary) or 6 (decimal) which the terminal subsystem uses as an index into the color map. Color map entry 6 contains the color red, which appears on the screen at the appropriate location.

You can control to which of the three bitmap planes your program has access. In other words, you can make each bitmap plane "read/write" or "read only." The `SET WRITING PLANES` instruction specifies which planes your program can write into.

The ability to "write-protect" individual planes should be used only for advanced graphics techniques.

NOTE

If a write-protected plane contains image information, that information will affect any image written over it. For example, suppose that you write-protect plane 2, and write ones into planes 1 and 4. Wherever plane 2 contains zero (forming 101 binary) the bitmap will point to color map entry 5. Wherever plane 2 contains one (forming 111 binary) the bitmap will point to color map entry 7.

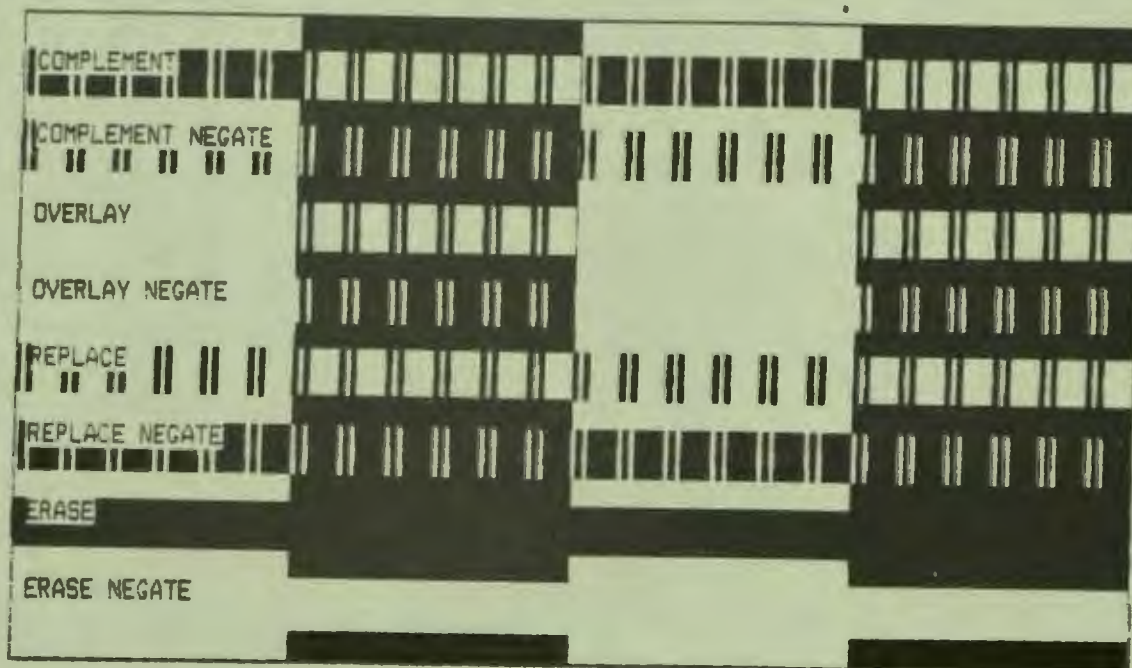


Figure 1-20: The Writing Modes (Shown with Line Style)

ATTRIBUTES

1.7.5 THE WRITING MODE

The writing mode is a powerful global attribute that is not included in the CORE Standard. The `SET_WRITING_MODE` instruction (described in Chapter 5) defines the exact manner in which output primitives are drawn on the view surfaces. The ten writing mode values are described below.

To describe how the writing mode works, the term "current pattern" is defined to be the bit pattern of whatever is being drawn. The set bits (1) are said to be "on" and the clear bits (0) are said to be "off."

- If CGL is drawing a line with fill disabled, the current pattern is the line style. For example, if the line style is `SOLID`, all bits are "on"; there are no "off" bits.
- Otherwise, with fill enabled, the current pattern is a character cell. The bits that represent the character are "on." The remainder of the cell is "off." This encompasses line primitives with fill enabled, text primitives, and marker primitives.

Figure 1-20 shows a screen image from an example program that demonstrates `SET_WRITING_MODE`. First, the program uses fill to create two large, vertical bars so that the screen has four equal areas of "on" and "off." Then it draws seven `DOT_DASHED` horizontal lines using each of the visible writing mode values. The ten writing mode values are:

0. TRANSPARENT

In transparent mode, CGL goes through the process of drawing output primitives and updates the current position without actually drawing anything. Thus, transparent mode is useful for determining what the new current position will be after an image is drawn, without actually drawing the image.

1. TRANSPARENT_NEGATE

Transparent negate mode is identical to transparent mode.

2. COMPLEMENT

The purpose of complement mode is to draw output primitives so that they stand out from existing images and thus have maximum visibility. Where the current pattern is "on," the image is affected. Wherever the current pattern is "off," the image is unaffected.

With the Extended Bitmap Option, CGL draws the "on" areas

ATTRIBUTES

using the "complement" of the existing image. Assuming all three planes are available:

$$\text{complement} = 7 - n$$

where n is the current (decimal) value in the bitmap

For example, a solid line drawn in complement mode over a bitmap value of 5 changes the value to 2. The same line drawn over a value of 1 changes the value to 6.

NOTE

For complement mode to work effectively, you must first set up the color map so that complementary entries contain complementary (or at least different) colors. The default values were chosen for VT125 compatibility and may not produce the desired result.

With no Extended Bitmap Option, CGL draws the "on" areas by negating (reversing) the existing image. For example, a solid line drawn over a dark value changes the value to light, while the same line over a light value changes the value to dark.

3. COMPLEMENT_NEGATE

Complement negate mode is identical to complement mode except the current pattern is negated. Wherever the current pattern is "off," the existing image is affected. Wherever the current pattern is "on," no drawing occurs.

4. OVERLAY

Overlay is the default writing mode. In overlay mode, CGL draws output primitives "on top of" existing images. Wherever the current pattern is "on," CGL draws with the writing index. Where the current pattern is "off," no drawing occurs.

5. OVERLAY_NEGATE

Overlay negate mode is identical to overlay mode except the current pattern is negated. Wherever the current pattern is "off," CGL draws in the writing index. Wherever the current pattern is "on," no drawing is done.

ATTRIBUTES

6. REPLACE

In replace mode, CGL draws output primitives while erasing any existing image. Wherever the current pattern is "on," CGL draws in the writing index. Wherever the current pattern is "off," CGL draws in the background index.

7. REPLACE_NEGATE

Replace negate mode is identical to replace mode except the current pattern is negated. Wherever the current pattern is "on," CGL draws in the background index. Wherever the current pattern is "off," CGL draw in the writing index.

8. ERASE

In erase mode, CGL draws output primitives by erasing existing images. CGL draws the entire current pattern in the background index.

9. ERASE_NEGATE

Erase negate mode is identical to erase mode except CGL draws the entire current pattern in the writing index.

1.7.6 THE GLOBAL ATTRIBUTE LIST

CGL maintains a list that contains the current values of all attributes, both global and output primitive-specific. This attribute list is named "global" for compatibility with future versions of the CORE Graphics Library. The SET GLOBAL ATTRIBUTES instruction sets the values of the entire global attributes list. Its use will improve performance when many attributes have to be set at once (for example on application startup), since the number of calls to CGL are minimized.

In general, reducing the number of calls to CGL will give improved performance because of the overhead associated with each call to the resident library (and especially when CGL is clustered with other libraries).

CHAPTER 2

PROGRAMMING WITH THE CORE GRAPHICS LIBRARY

2.1 CALLING CGL ROUTINES FROM HIGH-LEVEL LANGUAGES

To access the CORE Graphics Library, use the standard PDP-11 R5 calling sequence convention (sometimes called the FORTRAN Calling Sequence Convention). The library has one global entry point: CGL. The first parameter is an integer value that specifies an individual instruction. All parameters are passed by reference.

The data type and relative position of each parameter must match that expected by CGL. Although CGL cannot perform data type checking of parameters, it checks the number of parameters supplied. If a function doesn't work correctly, check the parameter data types. One of the most common bugs is a real parameter where an integer is required and vice-versa.

2.1.1 THE BASIC-PLUS-2 INTERFACE

To call CGL from a BASIC-PLUS-2 program or subprogram, use the CALL (BY REF) statement. External routine names do not have to be declared. Refer to your BASIC-PLUS-2 documentation for more information on the CALL statement.

Format

```
CALL CGL BY REF (inst_name, p1, p2, ..., pn)
```

BY REF specifies that the parameters are to be passed by reference (BASIC-PLUS-2 passes the parameter's address). Always use BY REF with CGL calls.

inst_name is an integer expression specifying the desired CGL instruction. CGL provides a file named "LB:[1,5]CGL.B2S" (listed in Appendix C) that declares a set of integer constants corresponding to the names of the CGL instructions.

THE BASIC-PLUS-2 INTERFACE

pl,p2,... are parameters as described in the individual instruction sections of this manual.

Example

```
10      %INCLUDE "LB:[1,5]CGL.B2S"
        CALL CGL BY REF (INITIALIZE_CORE)
        CALL CGL BY REF (NEW_FRAME)
        CALL CGL BY REF (SET_WINDOW, 0, 100, 0, 100)
        CALL CGL BY REF (MOVE_ABS 2, 0, 0)
        CALL CGL BY REF (RECTANGLE_ABS 2, 100, 100)
        CALL CGL BY REF (TERMINATE_CORE)
        END
```

Notes

- To pass an array to CGL, you must include the (empty) parentheses in the BASIC-PLUS-2 call, for example:

```
CALL CGL BY REF (POLYLINE_ABS_2, X(), Y(), 4%)
```

- BASIC-PLUS-2 does not allow you to pass array elements by reference. This line is invalid:

```
CALL CGL BY REF (INQUIRE_CURRENT_POSITION_2, &
                  CP(0%), CP(1%))
```

- You can pass a dynamic string variable to CGL. For example:

```
CALL CGL BY REF (TEXT, S$, LEN(S$))
```

2.1.2 THE COBOL-81 INTERFACE

To call CGL from a COBOL program, use the CALL statement. External routine names do not have to be declared. Refer to the Tool Kit COBOL-81 Documentation Supplement for detailed information on calling CGL routines from COBOL.

Format

```
CALL "CGL" USING inst_name pl p2 ... pn.
```

inst_name is a PIC S9(4) COMP item specifying the desired CGL instruction.

pl p2 ... are actual parameters as described in the individual instruction sections of this manual.

THE COBOL-81 INTERFACE

Example

IDENTIFICATION DIVISION.
PROGRAM-ID.

OUTLINE-WINDOW.

DATA DIVISION.

WORKING-STORAGE SECTION.

01	NEW-FRAME	PIC	S9(4) COMP VALUE 92.
01	SET-WINDOW	PIC	S9(4) COMP VALUE 80.
01	MOVE-ABS-2	PIC	S9(4) COMP VALUE 1.
01	RECTANGLE-ABS-2	PIC	S9(4) COMP VALUE 10.
01	INITIALIZE-CORE	PIC	S9(4) COMP VALUE 90.
01	TERMINATE-CORE	PIC	S9(4) COMP VALUE 91.
01	NUM-CONST-0	PIC	S9(4) COMP VALUE 0.
01	NUM-CONST-100	PIC	S9(4) COMP VALUE 100.
01	REAL1	PIC	X(4).
01	REAL2	PIC	X(4).
01	REAL3	PIC	X(4).
01	REAL4	PIC	X(4).

PROCEDURE DIVISION.

MAIN.

CALL "CGL" USING INITIALIZE-CORE.

CALL "CGL" USING NEW-FRAME.

CALL "CONIFL" USING

NUM-CONST-0 REAL1

NUM-CONST-100 REAL2

NUM-CONST-0 REAL3

NUM-CONST-100 REAL4.

CALL "CGL" USING SET-WINDOW REAL1 REAL2 REAL3 REAL4.

CALL "CGL" USING MOVE-ABS-2 REAL1 REAL3.

CALL "CGL" USING RECTANGLE-ABS-2 REAL2 REAL4.

CALL "CGL" USING TERMINATE-CORE.

GO TO DONE.

DONE.

EXIT.

Notes

- Text can be described as PIC X(n), where "n" is the maximum length of the text string.
- Tool Kit COBOL-81 provides a routine named CONIFL to convert integers to real numbers, as required by CGL (see example above).

2.1.3 THE DIBOL INTERFACE

To call CGL from a DIBOL program, use the XCALL statement. External routine names do not have to be declared. Refer to the

THE DIBOL INTERFACE

Tool Kit DIBOL User's Guide for detailed information on calling CGL routines from DIBOL.

Format

```
XCALL CGL (inst_name, p1, p2, ..., pn)
```

`inst_name` is the name of the CGL instruction. DIBOL provides a file named "LB:[1,5]CGL.DBL" (listed in Appendix C) that declares a set of integer constants corresponding to the names of the CGL instructions.

`p1,p2,...` are actual parameters as described in the individual instruction sections of this manual.

Example

```
.INCLUDE 'LB:[1,5]CGL.DBL'  
PROC  
    XCALL DCGL (GIC)  
    XCALL DCGL (GNF)  
    XCALL DCGL (GSW, 0, 100, 0, 100)  
    XCALL DCGL (GMA2, 0, 0)  
    XCALL DCGL (GRA2, 100, 100)  
    XCALL DCGL (GTC)  
END
```

2.1.4 THE FORTRAN INTERFACE

To call CGL from a FORTRAN program, use the CALL statement. External routine names do not have to be declared. Refer to the Tool Kit FORTRAN-77 Documentation Supplement for detailed information on calling P/OS routines from FORTRAN.

Format

```
CALL CGL (inst_name, p1, p2, ..., pn)
```

`inst_name` is an integer constant specifying the desired CGL instruction. CGL provides a file named "LB:[1,5]CGL.FTN" (listed in Appendix C) that defines a set of symbolic names corresponding to the CGL instruction numbers.

`p1,p2,...` are actual parameters as described in the individual instruction sections of this manual.

THE FORTRAN INTERFACE

Example

```
10      INCLUDE 'LB:[1,5]CGL.FTN'  
      CALL CGL (GIC)  
      CALL CGL (GNF)  
      CALL CGL (GSW, 0., 100., 0., 100.)  
      CALL CGL (GMA2, 0., 0.)  
      CALL CGL (GRA2, 100., 100.)  
      CALL CGL (GTC)  
      END
```

2.1.5 THE PASCAL INTERFACE

To call CGL from a PASCAL program, you must use the external (SEQ11) procedure names declared in "LB:[1,5]CGL.PAS," which is provided with the Tool Kit PASCAL distribution kit. Refer to the Tool Kit PASCAL User's Guide for detailed information on calling CGL routines from PASCAL.

To use CGL.PAS, include this line in your program:

```
      %INCLUDE 'LB:[1,5]CGLDEFS.PAS/NOLIST';
```

Remove the "/NOLIST" option if you prefer to see the declarations in your program listing.

Format

```
      instname (p1, p2, ..., pn);
```

inst_name is the name of a CGL instruction (a SEQ11 procedure declared in CGL.PAS).

p1,p2,... are actual parameters as described in the individual instruction sections of this manual. Check CGL.PAS for data types.

Example

```
PROGRAM OUTLINE WINDOW;  
%INCLUDE 'LB:[1,5]CGLDEFS.PAS/NOLIST'  
BEGIN  
  INITIALIZE_CORE;  
  NEW_FRAME;  
  SET_WINDOW (0.0, 100.0, 0.0, 100.0);  
  MOVE_ABS_2 (0.0, 0.0);  
  RECTANGLE_ABS_2 (100.0, 100.0);  
  TERMINATE_CORE;  
END.
```


THE PASCAL INTERFACE

Notes

- When calling a CGL routine, always pass the exact number of actual parameters specified in this manual. The declarations are set up so that each procedure has a formal parameter named \$\$\$, which has a default value corresponding to a CGL instruction number. For example:

```
[EXTERNAL($PCGL)]  
PROCEDURE MOVE_ABS_2 (VAR X, Y : [READONLY] REAL;  
                    $$$ : INTEGER := 1); SEQ11;
```

Do not pass an actual parameter for \$\$\$. You would invalidate the instruction number and cause an error.

- Some of the instructions have formal parameters with the READONLY attribute and can accept constants as actual parameters (as shown above).
- Some of the instructions have formal parameters with the UNSAFE attribute so that you can pass arrays of different lengths. They are:
 - POLYLINE_ABS_2
 - POLYLINE_REL_2
 - POLYGON_ABS_2
 - POLYGON_REL_2
 - TXT
 - POLYMARKER_ABS_2,
 - POLYMARKER_REL_2
 - CURVE_ABS_2
 - CURVE_REL_2

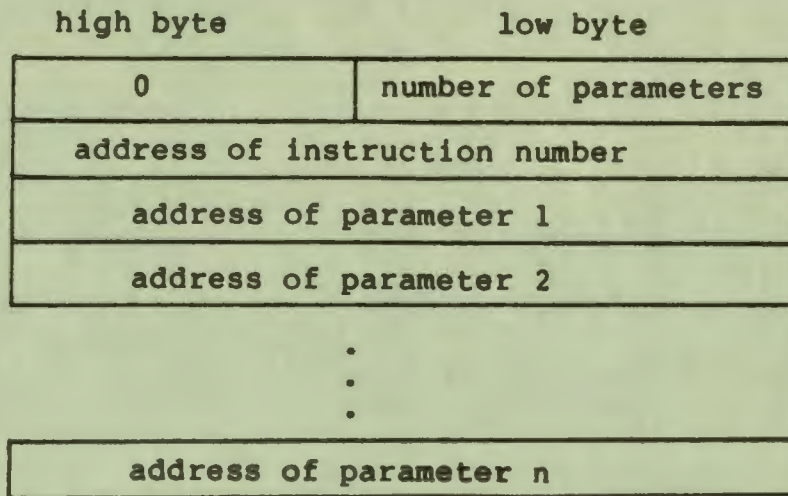
2.2 CALLING CGL ROUTINES FROM MACRO-11

To transfer control to a CGL routine:

```
JSR PC,CGL
```

General purpose register 5 (R5) contains the address of the parameter block which has the following format:

CALLING CGL ROUTINES FROM MACRO-11



The second word contains the address of a word containing the CGL instruction number. When the CGL routine returns, the contents of registers R0 through R5 and floating accumulators 0 through 3 are undefined. The stack pointer (SP) is restored to its state before the call.

You can check for errors by testing the carry (C) bit. If a function caused an error and an application does not call `REPORT_MOST_RECENT_ERROR`, then the carry (C) bit is returned set from CGL.

2.3 TERMINAL INPUT/OUTPUT

Do not use your programming language's output facilities to write to the terminal while CGL is operational. The results of mixed text mode and graphics mode operations are unpredictable. Because CGL has no input instructions, it is recommended that you use the `GETKEY` routine (described in the Developer's Tool Kit User's Guide) for keyboard input.

If you use your language's keyboard input facilities while CGL is operational, it is recommended that you turn off echo. Some languages can do this for you. For example, in `BASIC-PLUS-2`, you can say:

```
Z% = NOECHO(0%)
```

2.4 USING CGL WITH THE P/OS USER INTERFACE LIBRARY

You can use P/OS User Interface Library (POSRES) routines while

USING CGL WITH THE P/OS USER INTERFACE LIBRARY

CGL is operational. It is recommended that you follow this procedure to switch screen context from CGL to POSRES and back.

1. Use `INQUIRE_COLOR_MAP` to save the contents of the color map.
2. Use `NEW_FRAME` to clear the screen.
3. Use `SET_COLOR_MAP_ENTRY` to set entry zero to black.
4. Use `SET_COLOR_MAP_ENTRY` to set entry four to white.
5. Call the `POSRES` routine.
6. Use `SET_WRITING_PLANES (7)` to restore access to all planes.
7. Use `SET_COLOR_MAP` to restore the color map.
8. Redraw whatever was present before the `POSRES` call.

2.5 BUILDING YOUR CGL PROGRAM

Your language documentation describes how to create an Application Builder command (.CMD) file and an Overlay Descriptor Language (.ODL) file for your program. A typical .CMD file (for a PASCAL program named TEST) looks something like:

```
TEST/CP/FP,TEST/MA/-SP=TEST/MP
CLSTR=PASRES,CGLFPU,POSRES,RMSRES:RO
TASK   = TEST
STACK  = 30
UNITS  = 46
GBLDEF = TT$EFN:7
GBLDEF = WC$LUN:45
GBLDEF = MS$LUN:44
GBLDEF = HL$LUN:43
GBLDEF = MN$LUN:42
GBLDEF = TT$LUN:41
GBLDEF = G$LUN:41
ASG    = TT1:33
ASG    = SY:36
ASG    = LB:34:35:37
;EXTSCT = MS$BUF:3100
;EXTSCT = MN$BUF:4540
;EXTSCT = DM$BUF:4540
;EXTSCT = MM$BUF:1000
;EXTSCT = HL$BUF:3400
//
```

Make the following edits:

BUILDING YOUR CGL PROGRAM

1. If the default is not /FP, ensure that you use the /FP switch.
2. Find the line that begins with "CLSTR" and insert "CGLFPU" as the second library in the line. (If it says "CGLEIS," change it to "CGLFPU".) It should look something like:

```
CLSTR=PASRES,CGLFPU,POSRES,RMSRES:RO
```

NOTE

The CGLEIS (Extended Instruction Set) library is supported only for Tool Kit V1.0/1.5 applications that were built against CGLEIS and used the /-FP option. Use CGLFPU for Tool Kit V1.7 and later applications.

3. If there is already a line that defines the symbol "G\$LUN," after it insert the line:

```
GBLDEF = G$EFN:n
```

which defines the event flag number to be used by CGL. If there is no line defining G\$LUN, find the line that assigns a LUN to the terminal. It should look something like:

```
ASG = TT1:n
```

where "n" is a decimal number. Convert the number from decimal to octal and insert the following line:

```
GBLDEF = G$LUN:n  
GBLDEF = G$EFN:n
```

where "n" is an octal number. For example:

```
GBLDEF = G$LUN:41  
GBLDEF = G$EFN:1  
ASG = TT1:33
```

4. The .CMD file should now look something like:

BUILDING YOUR CGL PROGRAM

```
TEST/CP/FP,TEST/MA/--SP=TEST/MP
CLSTR=PASRES,CGLFPU,POSRES,RMSRES:RO
TASK   = TEST
STACK  = 30
UNITS  = 46
GBLDEF = TT$EFN:7
GBLDEF = WC$LUN:45
GBLDEF = MS$LUN:44
GBLDEF = HL$LUN:43
GBLDEF = MN$LUN:42
GBLDEF = TT$LUN:41
GBLDEF = G$EFN:1
GBLDEF = G$LUN:41
ASG    = TT1:33
ASG    = SY:36
ASG    = LB:34:35:37
;EXTSCT = MS$BUF:3100
;EXTSCT = MN$BUF:4540
;EXTSCT = DM$BUF:4540
;EXTSCT = MM$BUF:1000
;EXTSCT = HL$BUF:3400
//
```

2.6 INSTALLING YOUR CGL PROGRAM

Application programs that use CGL must specify the CGL cluster library in their installation command (.INS) file. (Refer to the Tool Kit User's Guide for detailed information on .INS files.)
Insert:

```
INSTALL [ZZSYS]CGLFPU.TSK/LIBRARY
```

If you use the PLAYBACK_FILE instruction in your application, you must install the file read task used by CGL. Insert:

```
INSTALL [ZZSYS]CGLGRT.TSK/TASK
```

If you use the LOAD_FONT instruction, you must install the font files as regions in memory by either:

1. Creating the region dynamically within your application (using the directive), or
2. Installing them here using:

```
INSTALL filespec/COMMON
```

CHAPTER 3

CONTROL INSTRUCTIONS

This chapter describes the instructions that control the overall operation of the CORE Graphics Library.

3.1 INITIALIZE_CORE - PREPARE GRAPHICS SYSTEM FOR USE

The INITIALIZE_CORE instruction guarantees that the graphics system is in a standard state with default parameters established. All programs that use CGL must execute the INITIALIZE_CORE instruction before any other CGL instruction (or any subprogram that uses CGL).

CORE Standard

INITIALIZE_CORE (outlevel, inlevel, dimension, hidden_surface)

CORE Graphics Library

INITIALIZE_CORE

Notes

- INITIALIZE_CORE is instruction number 90.
- You cannot execute INITIALIZE_CORE more than once before executing a TERMINATE_CORE instruction.
- If you do not execute this instruction before any other CGL calls, CGL does it for you and returns error number 743 for that call.
- The video monitor is implicitly initialized and selected by the INITIALIZE_CORE instruction.

INITIALIZE_CORE

Errors

701. The CORE system is already initialized.

3.2 TERMINATE_CORE - GRAPHICS SYSTEM USAGE FINISHED

The TERMINATE_CORE instruction performs an implicit END_BATCH, deselected and terminates all view surfaces, and releases all resources used by the CGL system.

CORE Standard

TERMINATE_CORE ()

CORE Graphics Library

TERMINATE_CORE

Notes

- TERMINATE_CORE is instruction number 91.
- Failing to terminate may cause your program to retain resources that are no longer needed.

Errors

743. The CORE system has not been initialized.

3.3 NEW_FRAME - REFRESH VIEW SURFACE

The NEW_FRAME instruction clears all currently selected view surfaces. Clearing a view surface is equivalent to filling the entire surface with the background index. All images are lost.

CORE Standard

NEW_FRAME ()

CORE Graphics Library

NEW_FRAME

NEW_FRAME

Notes

- NEW_FRAME is instruction number 92.
- NEW_FRAME affects only currently selected writing planes.
- NEW_FRAME has no effect on plotter view surfaces.

3.4 INITIALIZE_VIEW_SURFACE - ENABLE ACCESS TO DEVICE

The INITIALIZE_VIEW_SURFACE instruction prepares a specific output device for graphics output. It does not add that device to the list of currently selected devices; to do so you must also select the view surface.

CORE Standard

INITIALIZE_VIEW_SURFACE (surface_name, type)

CORE Graphics Library

INITIALIZE_VIEW_SURFACE (name, length)

name is a string expression that specifies the view surface name.

length is an integer expression that specifies the number of characters in the string.

There are three view surfaces available:

1. video
2. HP plotter
3. file

To initialize the video view surface use the surface name "TI: "; for the HP plotter use "GH:". Both these surface names, therefore, have length 3.

The file "device" creates a file of GIDIS display commands that can be used as input to other programs. The file view surface name should be an RMS compatible file specification with appropriate length (the maximum is 60 characters).

INITIALIZE_VIEW_SURFACE

NOTE

Only one file view surface can be active at one time.

You can re-execute the file using the `PLAYBACK_FILE` instruction. The file can also be printed using P/OS print services. It is recommended that the file name have an extension ".GID", since this is the default assumed by print services.

Not all CGL commands generate GIDIS output to the file.

Notes

- This will normally be transparent to your application with one exception, the `WAIT` command is handled internally by CGL. If you want to playback a series of slides, for example, with pauses between each slide, you must re-execute the wait again between each slide.
- `INITIALIZE_VIEW_SURFACE` is instruction number 103.
- A device must be initialized before it is selected.
- The video monitor is implicitly initialized and selected when you execute the `INITIALIZE_CORE` instruction.

Errors

- 705. View surface already initialized.
- 706. Invalid view surface name.
- 906. Error on view surface device.
- 907. Invalid when in begin/end batch.
- 908. View surface not ready.
- 910. Invalid when in begin/end define character.

3.5 TERMINATE_VIEW_SURFACE - DISABLE ACCESS TO DEVICE

The `TERMINATE_VIEW_SURFACE` instruction terminates access to and releases a specific output device.

TERMINATE_VIEW_SURFACE

CORE Standard

TERMINATE_VIEW_SURFACE (surface_name)

CORE Graphics Library

TERMINATE_VIEW_SURFACE (name, length)

name is a string expression that specifies the view surface name.

length is an integer expression that specifies the number of characters in the string.

Notes

- TERMINATE_VIEW_SURFACE is instruction number 104.
- See INITIALIZE_VIEW_SURFACE for a list of valid surface names.
- All view surfaces are implicitly deselected and terminated by the TERMINATE_CORE instruction.

Errors

- 708. View surface not initialized.
- 906. Error on view surface device.
- 907. Invalid when in begin/end batch.
- 910. Invalid when in begin/end define character.

3.6 SELECT_VIEW_SURFACE - ENABLE GRAPHICS OUTPUT TO DEVICE

The SELECT_VIEW_SURFACE instruction adds the specified device to the set of view surfaces to which CGL performs output.

CORE Standard

SELECT_VIEW_SURFACE (surface_name)

CORE Graphics Library

SELECT_VIEW_SURFACE (name, length)

name is a string expression that specifies the view surface name.

SELECT_VIEW_SURFACE

length is an integer expression that specifies the number of characters in the string.

Notes

- SELECT_VIEW_SURFACE is instruction number 105.
- See INITIALIZE_VIEW_SURFACE for a list of valid surface names.
- A device must be initialized before it is selected.
- The video monitor is implicitly initialized and selected by the INITIALIZE_CORE instruction.
- SELECT_VIEW_SURFACE has no effect on the current attribute values, current position, and viewing transformation.
- CGL conveys current state information (except font descriptions) to each view surface when you select it. SET_FONT_SIZE passes a font size and subsequent character definitions to all currently selected view surfaces. Thus, in theory, you can have different fonts, with characters of different aspect ratios, simultaneously defined for different view surfaces.

Example

```
program example;
%include 'lb:[1,5]cgldefs.pas/nolist'
  procedure draw_picture; external;
begin
  initialize_core;
  draw_picture;
  deselect_view_surface ('TI:',3);
  initialize_view_surface ('GH:',3);
  select_view_surface ('GH:',3);
  draw_picture;
  deselect_view_surface ('GH:',3);
  select_view_surface ('TI:',3);
end { example };
```

Errors

708. View surface not initialized.

SELECT_VIEW_SURFACE

- 709. View surface already selected.
- 906. Error on view surface device.
- 907. Invalid when in begin/end batch.
- 910. Invalid when in begin/end define character.

3.7 DESELECT_VIEW_SURFACE - DISABLE GRAPHICS OUTPUT TO DEVICE

The DESELECT_VIEW_SURFACE instruction removes the specified device to the set of view surfaces to which CGL performs output.

CORE Standard

DESELECT_VIEW_SURFACE (surface_name)

CORE Graphics Library

DESELECT_VIEW_SURFACE (name, length)

name is a string expression that specifies the view surface name.

length is an integer expression that specifies the number of characters in the string.

Notes

- DESELECT_VIEW_SURFACE is instruction number 106.
- See INITIALIZE_VIEW_SURFACE for a list of valid surface names.
- DESELECT_VIEW_SURFACE has no effect on the current attribute values, current position, and viewing transformation.
- All view surfaces are implicitly deselected and terminated by the TERMINATE_CORE instruction.

Errors

- 711. View surface not selected.
- 906. Error on view surface device. (complete batch first).

DESELECT_VIEW_SURFACE

907. Invalid when in begin/end batch.

910. Invalid when in begin/end define character.

3.8 PLAYBACK_FILE - EXECUTE FILE OF GRAPHICS COMMANDS

The PLAYBACK FILE instruction opens and reads a file of GIDIS commands and re-executes them on all currently selected view surfaces. Your application could, for example, create a file by selecting the file view surface, draw some lines, text, and so forth, on the video then play the file back to the plotter.

NOTE

If you want to playback a file currently open as a view surface you must deselect and terminate that view surface (to close the file) before the file can be read for playback.

You can also play back one file while a file view surface is selected, thus appending GIDIS commands from one file to another file.

CORE Standard

Not included.

CORE Graphics Library

PLAYBACK_FILE (name, length)

name is a string expression that specifies the view surface name.

length is an integer expression that specifies the number of characters in the string.

Notes

- PLAYBACK_FILE is instruction number 111.
- The name should be an RMS compatible file specification. The name length can be a maximum of 60 characters.

PLAYBACK_FILE

- All attributes are saved and restored around the playback.
- Files created with other software can also be played back. They must have sequential organization, with records of no more than 512 (decimal bytes) in length.

Errors

911. Error on file playback (file not found, etc).

3.9 BEGIN_BATCH - BEGIN STORING VIEW SURFACE UPDATES

The BEGIN_BATCH instruction begins storing all subsequent view surface updates in a buffer and continues to do so until it executes an END_BATCH instruction. If the buffer becomes full, CGL empties it (performs all stored updates) and continues to store subsequent updates. Some instructions also cause the buffer to be emptied (but they do not end batching); the CGL_WAIT instruction is an example. Batching should be used wherever appropriate since it will give a significant performance improvement.

CORE Standard

BEGIN_BATCH_OF_UPDATES ()

CORE Graphics Library

BEGIN_BATCH

Notes

- BEGIN_BATCH is instruction number 96.
- Images are not affected by BEGIN_BATCH - END_BATCH instructions. Only the view surface is affected.

Errors

716. There has been no END_BATCH since the last BEGIN_BATCH.

3.10 END_BATCH - END BATCH OF UPDATES

The END_BATCH instruction performs all of the view surface updates that have been stored since the last BEGIN_BATCH instruction. CGL no longer buffers view surface updates after END_BATCH executes.

END_BATCH

CORE Standard

END_BATCH_OF_UPDATES ()

CORE Graphics Library

END_BATCH

Notes

- END_BATCH is instruction number 97.
- You must execute a BEGIN_BATCH instruction at some point before using END_BATCH.

Errors

717. There has been no corresponding BEGIN_BATCH.

3.11 CGL_WAIT - SUSPEND EXECUTION

The CGL_WAIT instruction causes CGL to suspend all changes to view surfaces for a specified period of real time.

CORE Standard

Not included.

CORE Graphics Library

CGL_WAIT (seconds)

seconds, is a real expression that specifies the number of seconds to wait.

Notes

- CGL_WAIT is instruction number 95.
- The instruction name "CGL_WAIT" was chosen because "WAIT" is a reserved word in BASIC-PLUS-2.

3.12 ERASE_VIEWPORT - ERASE IMAGES IN VIEWPORT

The ERASE_VIEWPORT instruction clears the viewport without affecting other portions of the screen. Clearing the viewport is equivalent to filling the viewport with the background index.

ERASE_VIEWPORT

CORE Standard

Not included.

CORE Graphics Library

ERASE_VIEWPORT

Notes

- ERASE_VIEWPORT is instruction number 88.
- The ERASE_VIEWPORT instruction affects only the currently selected writing planes.
- This instruction is useful for applications that simulate multiple windows by moving the viewport to one of several disjoint areas. For example, you can create two pseudo-windows by bisecting the screen.
- The SET_VIEWPORT_2 instruction is described in Chapter 4.

3.13 PRINT_SCREEN - SEND SCREEN IMAGE TO PRINTER

The PRINT_SCREEN instruction sends an image of the video monitor screen contents to a graphics (LA50 or LA100) printer.

CORE Standard

Not included.

CORE Graphics Library

```
PRINT_SCREEN (lower_x, upper_x,  
              lower_y, upper_y,  
              x_offset, y_offset)
```

The parameters are real expressions representing world coordinates.

lower_x specifies the lower limit of the X coordinate.

upper_x specifies the upper limit of the X coordinate.

lower_y specifies the lower limit of the Y coordinate.

upper_y specifies the upper limit of the Y coordinate.

x_offset specifies the horizontal margin.

PRINT_SCREEN

`y_offset` specifies the vertical margin.

Notes

- PRINT_SCREEN is instruction number 94.
- If a plotter is connected, PRINT_SCREEN is inoperative.

3.14 REPORT_MOST_RECENT_ERROR - IDENTIFY EXECUTION ERROR

The REPORT_MOST_RECENT_ERROR instruction reports the instruction number and error code associated with the most recent CGL execution error and returns the system to a non-error state.

CORE Standard

REPORT_MOST_RECENT_ERROR (error_report)

CORE Graphics Library

REPORT_MOST_RECENT_ERROR (inst_name, code)

`inst_name` is an integer variable that receives the name (number) of the instruction that caused the most recent execution error.

`code` is an integer variable that receives the error code.

Notes

- REPORT_MOST_RECENT_ERROR is instruction number 93.
- The error codes are listed in Appendix A.
- Use REPORT_MOST_RECENT_ERROR if it appears that a CGL instruction is not working correctly or not working at all. For example, if you attempted to execute:

```
TEXT ("fubar", -5)
```

CGL would not draw anything. REPORT_MOST_RECENT_ERROR would tell you that a number 16 instruction (TEXT) caused error number 2 (N is less than or equal to zero).

- If CGL is in a non-error state, REPORT_MOST_RECENT_ERROR returns `inst_name` and `code` values of zero.

CHAPTER 4

VIEWING TRANSFORMATION INSTRUCTIONS

This section explains the instructions that describe the graphical world and control the viewing transformation.

4.1 SET_WINDOW - SPECIFY WORLD COORDINATE SPACE

The SET_WINDOW instruction specifies the edges of the window and resets the current position and the fill entity coordinates to the origin of the window. The window is the visible portion of world coordinate space (the portion that is mapped onto the viewport).

CORE Standard

SET_WINDOW (xmin, xmax, ymin, ymax)

INQUIRE_WINDOW (xmin, xmax, ymin, ymax)

CORE Graphics Library

SET_WINDOW (xmin, xmax, ymin, ymax)

INQUIRE_WINDOW (xmin, xmax, ymin, ymax)

The parameters are real expressions/variables representing world coordinates.

xmin specifies the X (horizontal) lower limit of the window.

xmax specifies the X (horizontal) upper limit of the window.

ymin specifies the Y (vertical) lower limit of the window.

ymax specifies the Y (vertical) upper limit of the window.

SET_WINDOW

Notes

- SET_WINDOW is instruction number 80.
- INQUIRE_WINDOW is instruction number 81.
- The default window specification is (0, 959, 0, 599), which corresponds to the Professional's physical device coordinates.

Errors

- 501. Invalid coordinate values (minimum >= maximum).

4.2 SET_ORIGIN - SPECIFY ORIGIN OF WINDOW

The SET_ORIGIN instruction specifies which corner of the viewport corresponds to the origin of the window and resets the current position and fill entity coordinates to the new origin. The origin of the window is the point addressed by the smallest world coordinate pair. For example, a window defined as (1,2,1,2) has as its origin the point (1,1).

CORE Standard

Not included.

CORE Graphics Library

SET_ORIGIN (origin

INQUIRE_ORIGIN (origin)

origin is is an integer expression/variable that specifies one of the following corners:

- 0 = bottom left
- 1 = top left (default)
- 2 = top right
- 3 = bottom right

Notes

- SET_ORIGIN is instruction number 86.

SET_ORIGIN

- INQUIRE_ORIGIN is instruction number 87.
- When you execute a SET_WINDOW or SET_VIEWPORT_2 instruction, CGL resets the current position (and the fill entity coordinates) to the origin of the window.

4.3 SET_WINDOW_CLIPPING - ENABLE OR DISABLE WINDOW CLIPPING

The SET_WINDOW_CLIPPING instruction enables or disables the displaying of output primitives (or portions of output primitives) that fall outside of the window.

CORE Standard

SET_WINDOW_CLIPPING (on_off)

INQUIRE_WINDOW_CLIPPING (on_off)

CORE Graphics Library

SET_WINDOW_CLIPPING (on_off)

INQUIRE_WINDOW_CLIPPING (on_off)

on_off is an integer expression/variable that contains one of the following values:

0 = off anything else = on

Notes

- SET_WINDOW_CLIPPING is instruction number 84.
- INQUIRE_WINDOW_CLIPPING is instruction number 85.
- Window clipping is on by default.
- If you disable window clipping, output primitives are clipped at the view surface edges only.

4.4 SET_NDC_SPACE_2 - DEFINE SIZE OF NDC SPACE

The SET_NDC_SPACE_2 instruction defines the NDC address space of all view surfaces within which viewports will be specified.

SET_NDC_SPACE_2

CORE Standard

SET_NDC_SPACE_2 (width, height)

INQUIRE_NDC_SPACE_2 (width, height)

CORE Graphics Library

SET_NDC_SPACE_2 (width, height)

INQUIRE_NDC_SPACE_2 (width, height)

width specifies the width of NDC space. The parameters are real expressions greater than zero, less than or equal to one. At least one parameter must equal one.

height specifies the height of NDC space.

Notes

- SET_NDC_SPACE_2 is instruction number 107.
- INQUIRE_NDC_SPACE_2 is instruction number 108.
- The default NDC space is (1,1).
- The SET_NDC_SPACE_2 instruction sets the default viewport to (0, 0, ,).
- SET_NDC_SPACE_2 can be used at most once per initialization of CGL and that call must appear before any SET/INQUIRE_VIEWPORT_2 instruction.
- For the Professional video monitor, an NDC space of (1, 0.625) will produce a square aspect ratio. For example, in a window defined as (0, 100, 0, 100), a rectangle 10 X 10 will be square and the bottom of the screen (with origin at top) will correspond to the Y coordinate 62.5.

Errors

- 503. SET_NDC_SPACE_2 already invoked since initialization.
- 504. Default NDC space already established.
- 505. A parameter is not in the range 0 to 1.

SET_NDC_SPACE_2

506. Neither width nor height has a value of 1.

507. Neither WIDTH nor HEIGHT can be equal to zero.

4.5 SET_VIEWPORT_2 - SPECIFY USABLE AREA OF VIEW SURFACE

The SET_VIEWPORT_2 instruction specifies a portion of normalized device coordinate space to be the viewport and resets the current position and the fill entity coordinates to the origin of the window. If you do not execute a SET_VIEWPORT_2 instruction, CGL uses all of NDC space (the entire view surface) by default.

CORE Standard

SET_VIEWPORT_2 (xmin, xmax, ymin, ymax)

INQUIRE_VIEWPORT_2 (xmin, xmax, ymin, ymax)

CORE Graphics Library

SET_VIEWPORT_2 (xmin, xmax, ymin, ymax)

INQUIRE_VIEWPORT_2 (xmin, xmax, ymin, ymax)

The parameters are real expressions representing normalized device coordinates in the range 0 to the NDC upper limit.

xmin specifies the lower limit of the X coordinate.

xmax specifies the upper limit of the X coordinate.

ymin specifies the lower limit of the Y coordinate.

ymax specifies the upper limit of the Y coordinate.

Notes

- SET_VIEWPORT_2 is instruction number 82.
- INQUIRE_VIEWPORT_2 is instruction number 83.
- The default viewport specification is (0, NDC WIDTH, 0, NDC HEIGHT).
- The viewport's sides are vertical and its top and bottom are horizontal.

SET_VIEWPORT_2

- The viewport cannot exceed the bounds of NDC space.

Errors

- 501. Invalid coordinate values (minimum \geq maximum).
- 508. A value outside NDC space is not allowed.

4.6 SCROLL - MOVE SCREEN CONTENTS

The SCROLL instruction moves the contents of the entire screen by a specified amount of world coordinate space. It has no effect on the viewing transformation or current values.

CORE Standard

Not included.

CORE Graphics Library

SCROLL (δ_x , δ_y)

The parameters are real expressions representing world coordinates.

δ_x specifies the X (horizontal) movement.

δ_y specifies the Y (vertical) movement.

Notes

- SCROLL is instruction number 89.
- The direction of movement depends on the origin of the window. For example, with the default origin (top-left), positive δ_x values scroll toward the left side of the screen and positive δ_y values scroll toward the top of the screen. Figure 4-1 shows how the SCROLL instruction works with the default origin.
- Scrolling does not cause any image to be drawn. The area scrolled onto the screen is filled with the background index.
- Images that scroll off the screen are lost.

SCROLL

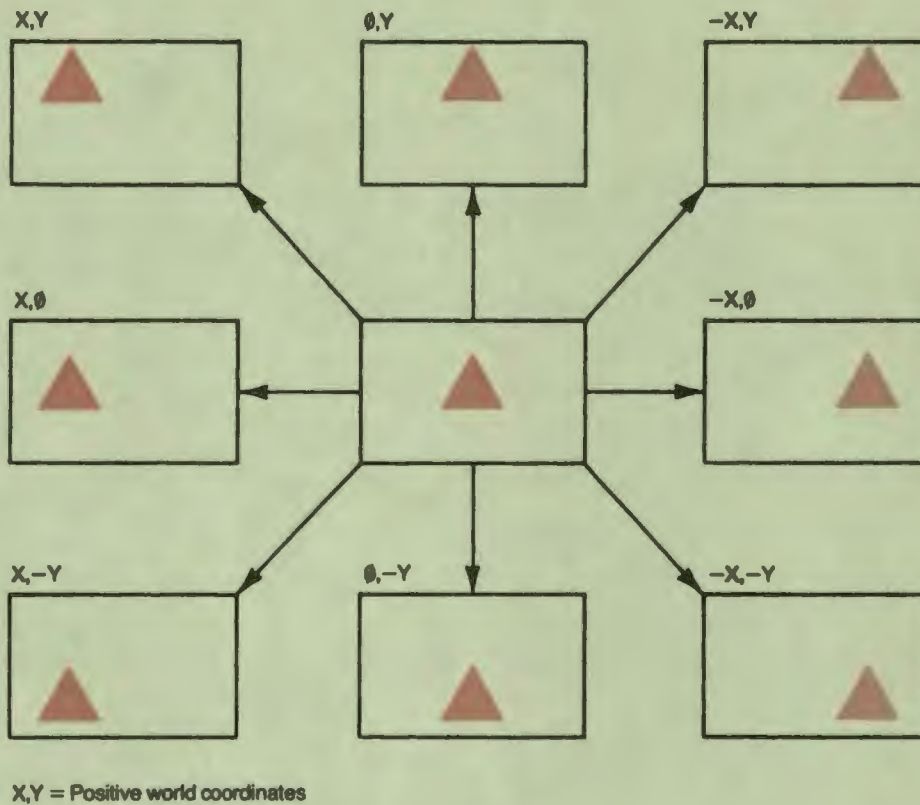


Figure 4-1: The SCROLL Instruction with Default Window Origin

4.7 SCROLL_VIEWPORT - MOVE VIEWPORT CONTENTS

The SCROLL_VIEWPORT instruction moves the contents of the viewport by a specified amount of world coordinate space without affecting images drawn outside the viewport. It has no effect on the viewing transformation or current values.

CORE Standard

Not included.

CORE Graphics Library

SCROLL_VIEWPORT (delta_x, delta_y)

The parameters are real expressions representing world coordinates.

SCROLL_VIEWPORT

`delta_x` specifies the X (horizontal) movement.

`delta_y` specifies the Y (vertical) movement.

Notes

- `SCROLL_VIEWPORT` is instruction number 102.
- The direction of movement depends on the origin of the window. For example, with the default origin (top-left), positive `delta_x` values scroll toward the left side of the screen and positive `delta_y` values scroll toward the top of the screen. Figure 4-1 shows how the `SCROLL_VIEWPORT` instruction works with the default origin.
- Scrolling does not cause any image to be drawn. The area scrolled into the viewport is filled with the background index.
- Images that scroll outside the viewport are lost.

CHAPTER 5
GLOBAL ATTRIBUTE INSTRUCTIONS

This chapter describes the instructions that set the values of global attributes.

NOTE

The following instructions require the Extended Bitmap Option. They do nothing if it is not present.

SET_COLOR_MAP_ENTRY

SET_COLOR_MAP

SET_WRITING_PLANES

5.1 SET_WRITING_INDEX - SELECT COLOR MAP INDEX FOR IMAGES

The SET_WRITING_INDEX instruction selects an index into the color map for images created by subsequent output primitive instructions. It does not change the appearance of any existing images.

CGL uses the writing index to draw images in OVERLAY, OVERLAY_NEGATE, and ERASE_NEGATE modes. For more detailed information, refer to the description of the writing mode in Chapter 1.

CORE Standard

DEFINE_COLOR_INDEX (surface_name, i, c1, c2, c3)

INQUIRE_COLOR_INDEX (surface_name, i, c1, c2, c3)

SET_WRITING_INDEX

CORE Graphics Library

SET_WRITING_INDEX (index)

INQUIRE_WRITING_INDEX (index)

index is an integer expression/variable that specifies one of the eight color map entries (0 to 7).

Notes

- SET_WRITING_INDEX is instruction number 60.
- INQUIRE_WRITING_INDEX is instruction number 61.
- The default writing index is color map entry 7 (which contains the color white by default).
- If the Extended Bitmap Option is not present, the following writing index values apply:
 - 0 = dark
 - other values = light (default)
- The SET_WRITING_INDEX instruction works differently when used with plotter view surfaces (see Appendix B).

Errors

- 401. One or more of the attribute values is invalid.

5.2 SET_BACKGROUND_INDEX - SET BACKGROUND COLOR MAP INDEX

The SET_BACKGROUND_INDEX instruction selects an index into the color map for the background. It does not change the appearance of the background until CGL executes a NEW_FRAME or ERASE_VIEWPORT instruction.

CGL uses the background index to erase the view surface (NEW_FRAME) and to draw images in REPLACE, REPLACE_NEGATE, and ERASE modes. For more detailed information, refer to the description of the writing mode in Chapter 1.

CORE Standard

SET_BACKGROUND_INDEX (index)

INQUIRE_BACKGROUND_INDEX (index)

SET_BACKGROUND_INDEX

CORE Graphics Library

SET_BACKGROUND_INDEX (index)

INQUIRE_BACKGROUND_INDEX (index)

index is an integer expression/variable that specifies one of the eight color map entries (0 to 7).

Notes

- SET_BACKGROUND_INDEX is instruction number 62.
- INQUIRE_BACKGROUND_INDEX is instruction number 63.
- The default background index is color map entry 0 (which contains the color black by default).
- If the Extended Bitmap Option is not present, the following background index values apply:
 - 0 = dark (default)
 - other values = light
- The SET_BACKGROUND_INDEX instruction works differently when used with plotter view surfaces (see Appendix B).

Errors

401. One or more of the attribute values is invalid.

5.3 SET_COLOR_MAP_ENTRY - SET COLOR MAP ENTRY RGB VALUES

The SET_COLOR_MAP_ENTRY instruction sets the RGB (red, green, blue) values of an individual color map entry and of any existing images drawn with that entry.

CORE Standard

Not included.

CORE Graphics Library

SET_COLOR_MAP_ENTRY (entry, color)

INQUIRE_COLOR_MAP_ENTRY (entry, color)

entry is an integer expression (range 0 to 7) that specifies which color map entry to set.

SET_COLOR_MAP_ENTRY

color is a three-element integer array (range 0 to 7) that specifies red, green, and blue values in that order.

Notes

- SET_COLOR_MAP_ENTRY is instruction number 66.
- INQUIRE_COLOR_MAP_ENTRY is instruction number 67.
- SET_COLOR_MAP_ENTRY has no effect on plotter view surfaces.
- You can declare a color map entry array with whatever subscripts your programming language allows. If the array is indexed from zero to two:

color(0) specifies a red value
color(1) specifies a green value
color(2) specifies a blue value

Example

This PASCAL code shows how to use an array constant to specify a color map entry. The type declarations are from the include file CGLDEFS.PAS and are shown only for illustrative purposes.

```
procedure example; { Set color map entry no. 4 to white }
type
  RGB_value = 0..7;
  color_map_entry = array [1..3] of RGB_value;

const
  white = color_map_entry (7, 7, 6);

begin
  set_color_map_entry (4, white);
end { example };
```

5.4 SET_COLOR_MAP - SET ALL COLOR MAP RGB VALUES

The SET_COLOR_MAP instruction sets the RGB (red, green, blue) values of the entire color map and of any existing images.

CORE Standard

Not included.

SET_COLOR_MAP

CORE Graphics Library

SET_COLOR_MAP (color_map)

INQUIRE_COLOR_MAP (color_map)

color_map is a 24-element integer array containing color values (range 0 to 7) that specify all eight color map entries in the order red, green, blue, red, green, blue, and so forth.

Notes

- SET_COLOR_MAP is instruction number 64.
- INQUIRE_COLOR_MAP is instruction number 65.
- SET_COLOR_MAP has no effect on plotter view surfaces.
- You can declare a color map array with whatever subscripts your programming language allows. If the array is indexed from zero to 23:

Array Element	0	1	2	3	4	...	19	20	21	22	23
Color Map Entry	0	0	0	1	1	...	6	6	7	7	7
RGB Value	R	G	B	R	G	...	G	B	R	G	B

Example

This PASCAL code shows how to use an array constant to set the color map. The type declarations are from the include file CGLDEFS.PAS and are shown only for illustrative purposes.

```
procedure example; { Set up the color map }
type
  RGB_value = 0..7;
  color_map = array [0..23] of RGB_value;

const
  default_map := color_map (7,0,0, 0,7,0, 0,0,6, 0,0,0,
                           7,7,6, 7,7,0, 7,0,6, 0,7,6);

begin
  set_color_map (default_map);
end {example};
```


SET_WRITING_PLANES - SELECT COMBINATION OF PLANES

5.5 SET_WRITING_PLANES - SELECT COMBINATION OF PLANES

The SET_WRITING_PLANES instruction selects which of the three bitmap planes can be written into by CGL. It does not affect the contents of any of the planes. For more information, refer to the description of the bitmap in Chapter 1.

CORE Standard

Not included.

CORE Graphics Library

SET_WRITING_PLANES (n)

INQUIRE_WRITING_PLANES (n)

n is an integer expression/variable that specifies that CGL can write into one of the following combinations of planes:

n	Plane 4	Plane 2	Plane 1	Writeable Color Map Entries*
0	-	-	-	none
1	-	-	X	0, 1
2	-	X	-	0, 2
3	-	X	X	0, 1, 2, 3
4	X	-	-	0, 4
5	X	-	X	0, 1, 4, 5
6	X	X	-	0, 2, 4, 6
7	X	X	X	0, 1, 2, 3, 4, 5, 6, 7

* assuming that all write-protected planes are empty

Notes

- SET_WRITING_PLANES is instruction number 68.
- INQUIRE_WRITING_PLANES is instruction number 69.

SET_WRITING_PLANES

- The default n value is 7 (all three planes selected).
- SET_WRITING_PLANES has no effect on plotter view surfaces.

5.6 SET_WRITING_MODE - SET WRITING CHARACTERISTICS

The SET_WRITING_MODE instruction defines the exact manner in which CGL draws output primitives. For detailed information, refer to the description of the writing mode in Chapter 1.

CORE Standard

Not included.

CORE Graphics Library

SET_WRITING_MODE (mode)

INQUIRE_WRITING_MODE (mode)

mode is an integer expression/variable that specifies one of the following values:

0 = TRANSPARENT	5 = OVERLAY_NEGATE
1 = TRANSPARENT_NEGATE	6 = REPLACE
2 = COMPLEMENT	7 = REPLACE_NEGATE
3 = COMPLEMENT_NEGATE	8 = ERASE
4 = OVERLAY (default)	9 = ERASE_NEGATE

Notes

- SET_WRITING_MODE is instruction number 70.
- INQUIRE_WRITING_MODE is instruction number 71.
- The constant declaration files included with the CGL software kit include symbols for all ten writing mode values.
- The SET_WRITING_MODE instruction works differently when used with plotter view surfaces (see Appendix B).

SET_GLOBAL_ATTRIBUTES - SET GLOBAL ATTRIBUTE LIST

5.7 SET_GLOBAL_ATTRIBUTES - SET GLOBAL ATTRIBUTE LIST

The SET_GLOBAL_ATTRIBUTES instruction sets the values of the entire global output primitive attribute list with a single call.

CORE Standard

SET_PRIMITIVE_ATTRIBUTES_2 (primitive_attribute_array_2)
 INQUIRE_PRIMITIVE_ATTRIBUTES_2 (primitive_attribute_array_2)

CORE Graphics Library

SET_GLOBAL_ATTRIBUTES (int_list, real_list)
 INQUIRE_GLOBAL_ATTRIBUTES (int_list, real_list)

int_list is a subscripted variable that specifies a 19-element integer array as shown below.

real_list is a subscripted variable that specifies an eight-element real array as shown below.

Table 5-1: Integer Attribute List

Element	Attribute Name	Default Value
0	writing index	7 (light/white)
1	background index	0 (dark/black)
2	writing mode	4 (OVERLAY)
3	line style style	1 (SOLID)
4	line style pattern	-1 *
5	line style mult	2
6	font	0 (DEC Multinational)
7	character path path	0 (horizontal)
8	character path mode	1 (string)
9	character justification x_just	1 (left)
10	character justification y_just	1 (top)
11	character italic	0 (vertical)
12	marker symbol symbol	1 (period)
13	marker symbol code	183 *
14	fill mode	0 (off)
15	fill character font	0 *
16	fill character char	0 *
17	fill character width_mult	1
18	fill character height_mult	1

* ignored by CGL while default values remain in effect

SET_GLOBAL_ATTRIBUTES

Table 5-2: Real Attribute List

Element	Attribute Name	Default Value
0	linewidth dx	0.0 *
1	linewidth dy	0.0 *
2	character size width	12.0
3	character size height	25.0
4	character spacing delta_x	12.0
5	character spacing delta_y	0.0
6	fill entity x	0.0
7	fill entity y	0.0

* one physical device coordinate unit

Notes

- o SET_GLOBAL_ATTRIBUTES is instruction number 72.
- o INQUIRE_GLOBAL_ATTRIBUTES is instruction number 73.
- o Some of the SET_GLOBAL_ATTRIBUTES parameters have no effect on, or work differently when used with plotter view surfaces (see Appendix B).

Example

```

procedure example; { set up global attribute list }
var
  integer_array = array [0..18] of integer;
  real_array = array [0..7] of real;

const
  integer_list = integer_array (7, 0, 4, 1, -1, 2, 0, 0, 0,
                                1, 1, 0, 1, 183, 0, 0, 0, 1, 1);

  real_list = real_array (0.0, 0.0, 12.0, 25.0,
                          12.0, 0.0, 0.0, 0.0);

begin
  set_global_attributes (integer_list, real_list);
end { example };

```




CHAPTER 6

CURRENT POSITION AND MARKER INSTRUCTIONS

This chapter describes the instructions that change and report on the current position, draw markers, and control marker attribute values.

6.1 CURRENT POSITION INSTRUCTIONS

Current position instructions change or report on the value of the current position; they do not affect the view surface.

6.1.1 MOVE_ABS_2 - Move to Absolute Position

The MOVE_ABS_2 instruction changes the current position to the specified world coordinate position.

CORE Standard

MOVE_ABS_2 (x, y)

CORE Graphics Library

MOVE_ABS_2 (x, y)

The parameters are real expressions representing world coordinates.

x specifies the new X (horizontal) value of the current position.

y specifies the new Y (vertical) value of the current position.

MOVE_ABS_2

Notes

- MOVE_ABS_2 is instruction number 1.

6.1.2 MOVE_REL_2 - Move Relative to Current Position

The MOVE_REL_2 instruction changes the current position according to the specified offsets (delta values).

CORE Standard

MOVE_REL_2 (dx, dy)

CORE Graphics Library

MOVE_REL_2 (delta_x, delta_y)

The parameters are real expressions representing world coordinates.

delta_x specifies a change in the X (horizontal) current position.

delta_y specifies a change in the Y (vertical) current position.

Notes

- o MOVE_REL_2 is instruction number 2.

6.1.3 INQUIRE_CURRENT_POSITION_2 - Get Current Position

The INQUIRE_CURRENT_POSITION_2 instruction returns the current world coordinate position.

CORE Standard

INQUIRE_CURRENT_POSITION_2 (x, y)

CORE Graphics Library

INQUIRE_CURRENT_POSITION_2 (x, y)

The parameters are real variables representing world coordinates.

x receives the value of the X (horizontal) current position.

INQUIRE_CURRENT_POSITION_2

y receives the value of the Y (vertical) current position.

Notes

- INQUIRE_CURRENT_POSITION_2 is instruction number 3.

6.1.4 SET_CURSOR - Specify Cursor Characteristics

The SET_CURSOR instruction controls the appearance of the cursor, the visual representation of the current position.

CORE Standard

Not included.

CORE Graphics Library

SET_CURSOR (font, char, width, height, dx, dy)

INQUIRE_CURSOR (font, char, width, height, dx, dy)

font is an integer expression/variable in the range 0 to 3 that specifies one of the four available fonts.

char is an integer expression/variable in the range 32 to 126 or 160 to 255 that specifies the decimal equivalent of the character.

width is an integer expression/variable that specifies a multiplier on the width of the character.

height is an integer expression/variable that specifies a multiplier on the height of the character.

dx is a real expression/variable in the range 0 to 1 that specifies the horizontal offset from the upper-left corner of the cursor character to the current position.

dy is a real expression/variable in the range 0 to 1 that specifies the vertical offset from the upper-left corner of the cursor character to the current position.

Notes

- SET_CURSOR is instruction number 100.

SET_CURSOR

- INQUIRE_CURSOR is instruction number 101.
- Font -1 is a special set of pre-defined cursors:

Char ----	Cursor -----
-1	none
0	default, crosshairs
1	crosshairs
2	full screen crosshairs
3	block

- Redefining the character currently being used as the cursor does not change the cursor. Only SET_CURSOR specifies a new cursor.
- SET_CURSOR has no effect on plotter view surfaces.

Errors

- 401. One or more of the attribute values is invalid.
- 910. Invalid when in begin/end define character.

6.2 MARKER PRIMITIVE INSTRUCTIONS

Marker instructions draw markers or series of markers.

6.2.1 MARKER_ABS_2 - Draw Marker at Absolute Position

The MARKER_ABS_2 instruction changes the current position to the specified world coordinate position and draws a marker at that position.

CORE Standard

MARKER_ABS_2 (x, y)

CORE Graphics Library

MARKER_ABS_2 (x, y)

The parameters are real expressions representing world

MARKER_ABS_2

coordinates.

x specifies the X (horizontal) position at which to draw a marker.

y specifies the Y (vertical) position at which to draw a marker.

Notes

- MARKER_ABS_2 is instruction number 33.

6.2.2 MARKER_REL_2 - Draw Marker Relative to Current Position

The MARKER_REL_2 instruction changes the current position according to the specified offsets (delta values) and draws a marker at the new current position.

CORE Standard

MARKER_REL_2 (dx, dy)

CORE Graphics Library

MARKER_REL_2 (delta_x, delta_y)

The parameters are real expressions representing world coordinates.

delta_x specifies the X (horizontal) offset at which to draw a marker.

delta_y specifies the Y (vertical) offset at which to draw a marker.

Notes

- MARKER_REL_2 is instruction number 34.

6.2.3 POLYMARKER_ABS_2 - Draw Markers at Absolute Positions

The POLYMARKER_ABS_2 instruction is an extension of the MARKER_ABS_2 instruction; it draws a series of markers. CGL changes the current position to each of a list of world coordinate positions and draws a marker at each position.

CORE Standard

POLYMARKER_ABS_2 (x_array, y_array, n)

CORE Graphics Library

POLYMARKER_ABS_2 (x_array, y_array, n)

x_array is a subscripted real variable that specifies a list of X world coordinate positions at which to draw a marker.

y_array is a subscripted real variable that specifies a list of Y world coordinate positions at which to draw a marker.

n is an integer expression that specifies the number of elements in each array.

Notes

- POLYMARKER_ABS_2 is instruction number 35.
- When the POLYMARKER_ABS_2 instruction has finished, the current position is the last specified position.

Errors

2. N is less than or equal to zero.

6.2.4 POLYMARKER_REL_2 - Draw Markers at Relative Positions

The POLYMARKER_REL_2 instruction is an extension of the MARKER_REL_2 instruction; it draws a series of markers. CGL changes the current position to each of a list of world coordinate offsets and draws a marker at each new position.

CORE Standard

POLYMARKER_REL_2 (dx_array, dy_array, n)

CORE Graphics Library

POLYMARKER_REL_2 (dx_array, dy_array, n)

dx_array is a subscripted real variable that specifies a list of X world coordinate offsets at which to draw a marker.

dy_array is a subscripted real variable that specifies a list of Y world coordinate offsets at which to draw a marker.

POLYMARKER_REL_2

n is an integer expression that specifies the number of elements in each array.

Notes

- POLYMARKER_REL_2 is instruction number 36.
- When the POLYMARKER_REL_2 instruction has finished, the current position is the last specified position.

Errors

2. N is less than or equal to zero.

6.3 MARKER ATTRIBUTE INSTRUCTIONS

This instruction allows you to specify the symbol to be used in subsequent marker instructions.

6.3.1 SET_MARKER_SYMBOL - Select New Marker Symbol

The SET_MARKER_SYMBOL instruction specifies one of five symbols defined by the CORE Standard or another character as the current marker symbol.

CORE Standard

SET_MARKER_SYMBOL (symbol)

INQUIRE_MARKER_SYMBOL (symbol)

CORE Graphics Library

SET_MARKER_SYMBOL (symbol, code)

INQUIRE_MARKER_SYMBOL (symbol, code)

symbol is an integer expression/variable that specifies one of the following five standard symbols (code is ignored) or another character (symbol > 5 or symbol < 1).

- | | | | |
|---|---|---|--------------------|
| 1 | = | . | (period) (default) |
| 2 | = | + | (plus sign) |
| 3 | = | * | (asterisk) |
| 4 | = | O | (upper case O) |
| 5 | = | X | (upper case X) |

SET_MARKER_SYMBOL

code is an integer expression/variable that specifies the decimal code of a character from the current font.

Notes

- SET_MARKER_SYMBOL is instruction number 37.
- INQUIRE_MARKER_SYMBOL is instruction number 38.
- A symbol value that is greater than five or less than one indicates that the "code" parameter specifies the desired character.
- The default symbol is the period (value = 1).
- SET_MARKER_SYMBOL works differently when used with plotter view surfaces (see Appendix B).

CHAPTER 7

LINE INSTRUCTIONS

7.1 STRAIGHT LINE PRIMITIVE INSTRUCTIONS

Line instructions draw straight lines or series of connected straight lines.

7.1.1 LINE_ABS_2 - Draw Line to Absolute Position

The LINE_ABS_2 instruction changes the current position to the specified world coordinate position and draws a line connecting the old current position and the new current position.

CORE Standard

LINE_ABS_2 (x, y)

CORE Graphics Library

LINE_ABS_2 (x, y)

The parameters are real expressions representing world coordinates.

x specifies an X (horizontal) position to which to draw a line.

y specifies a Y (vertical) position to which to draw a line.

Notes

- LINE_ABS_2 is instruction number 4.

7.1.2 LINE_REL_2 - Draw Line to Relative Position

The LINE_REL_2 instruction changes the current position according

LINE_REL_2

to the specified world coordinate offsets and draws a line connecting the old current position and the new current position.

CORE Standard

LINE_REL_2 (dx, dy)

CORE Graphics Library

LINE_REL_2 (delta_x, delta_y)

The parameters are real expressions representing world coordinates.

delta_x specifies an X (horizontal) offset to which to draw a line.

delta_y specifies a Y (vertical) offset to which to draw a line.

Notes

- LINE_REL_2 is instruction number 5.

7.1.3 POLYLINE_ABS_2 - Draw Lines to Absolute Positions

The POLYLINE_ABS_2 instruction is an iterated LINE_ABS_2 instruction. You supply a list of absolute positions and CGL draws a series of connected lines starting at the current position and ending at the last position in the list.

CORE Standard

POLYLINE_ABS_2 (x_array, y_array, n)

CORE Graphics Library

POLYLINE_ABS_2 (x_array, y_array, n)

x_array is a subscripted real variable that specifies a list of X world coordinate positions to which to draw a line.

y_array is a subscripted real variable that specifies a list of Y world coordinate positions to which to draw a line.

n is an integer expression that specifies the number of elements in each array.

POLYLINE_ABS_2

Notes

- POLYLINE_ABS_2 is instruction number 6.
- When the POLYLINE_ABS_2 instruction has finished, the current position is the end of the last line drawn: x_array(n), y_array(n).

Errors

2. N is less than or equal to zero.

7.1.4 POLYLINE_REL_2 - Draw Lines to Relative Positions

The POLYLINE_REL_2 instruction is an iterated LINE_REL_2 instruction. You supply a list of relative positions and CGL draws a series of connected lines starting at the current position and ending at the last position in the list.

CORE Standard

POLYLINE_REL_2 (dx_array, dy_array, n)

CORE Graphics Library

POLYLINE_REL_2 (dx_array, dy_array, n)

dx_array is a subscripted real variable that specifies a list of world coordinate offsets to which to draw a line.

dy_array is a subscripted real variable that specifies a list of world coordinate offsets to which to draw a line.

n is an integer expression that specifies the number of elements in each array.

Notes

- POLYLINE_REL_2 is instruction number 7.
- When the POLYLINE_REL_2 instruction has finished, the current position is the end of the last line drawn.

Errors

2. N is less than or equal to zero.

7.1.5 POLYGON_ABS_2 - Draw Polygon by Absolute Positions

The POLYGON_ABS_2 instruction is similar to the POLYLINE_ABS_2 instruction. You supply a list of absolute positions and CGL draws a series of connected lines. The differences are:

- CGL begins drawing at the first position in the specified list, rather than the current position.
- CGL draws a line from the last position in the list to the first position, closing the figure.

CORE Standard

```
POLYGON_ABS_2 (x_array, y_array, n)
```

CORE Graphics Library

```
POLYGON_ABS_2 (x_array, y_array, n)
```

x_array is a subscripted real variable that specifies a list of X world coordinate positions describing a polygon.

y_array is a subscripted real variable that specifies a list of Y world coordinate positions describing a polygon.

n is an integer expression that specifies the number of elements in each array.

Notes

- POLYGON_ABS_2 is instruction number 8.
- Assuming that arrays are numbered from 0 to n (as in BASIC-PLUS-2), the instruction (POLYGON_ABS_2, x_array, y_array, n+1) is equivalent to:

```
MOVE_ABS_2, x_array(0), y_array(0)
LINE_ABS_2, x_array(1), y_array(1)
LINE_ABS_2, x_array(2), y_array(2)
      .
      .
      .
LINE_ABS_2, x_array(n), y_array(n)
LINE_ABS_2, x_array(0), y_array(0)
```

POLYGON_ABS_2

When the POLYGON_ABS_2 instruction has finished, the current position has the value (x_array(0), y_array(0)).

Errors

3. N is less than or equal to two.
904. Too many points in closed, filled figure.

7.1.6 POLYGON_REL_2 - Draw Polygon by Relative Positions

The POLYGON_REL_2 instruction is similar to the POLYLINE_REL_2 instruction. You supply a list of relative positions and CGL draws a series of connected lines. The differences are:

- CGL begins drawing at the first position in the specified list, rather than the current position.
- CGL draws a line from the last position in the list to the first position, closing the figure.

CORE Standard

POLYGON_REL_2 (dx_array, dy_array, n)

CORE Graphics Library

POLYGON_REL_2 (dx_array, dy_array, n)

dx_array is a subscripted real variable that specifies a list of X world coordinate offsets describing a polygon.

dy_array is a subscripted real variable that specifies a list of Y world coordinate offsets describing a polygon.

n is an integer expression that specifies the number of elements in each array.

Notes

- POLYGON_REL_2 is instruction number 9.
- Assuming that arrays are numbered from 0 to n (as in BASIC-PLUS-2), the instruction (POLYGON_REL_2, dx_array, dy_array, n+1) is equivalent to:

```
MOVE_REL_2, dx_array(0), dy_array(0)
INQUIRE_CURRENT_POSITION_2, x1, y1
LINE_REL_2, dx_array(1), dy_array(1)
```


POLYGON_REL_2

```
LINE_REL_2, dx_array(2), dy_array(2)
      .
      .
      .
LINE_REL_2, dx_array(n), dy_array(n)
INQUIRE_CURRENT_POSITION_2, xn, yn
LINE_REL_2, x1 - xn, y1 - yn
```

When the POLYGON_REL_2 instruction has finished, the current position has the value that was obtained when CGL executed (MOVE_REL_2, dx_array(0), dy_array(0)).

Errors

3. N is less than or equal to two.
904. Too many points in closed, filled figure.

7.1.7 RECTANGLE_ABS_2 - Draw Rectangle by Absolute Position

The RECTANGLE_ABS_2 instruction draws a series of connected lines forming a four-sided, perpendicular, polygon with the current position at one corner and a point specified as an absolute position in world coordinate space at the opposing corner. It does not change the current position.

CORE Standard

Not included.

CORE Graphics Library

RECTANGLE_ABS_2 (x, y)

The parameters are real expressions representing world coordinates.

- x specifies an X (horizontal) position describing a rectangle.
- y specifies a Y (vertical) position describing a rectangle.

Notes

- RECTANGLE_ABS_2 is instruction number 10.

RECTANGLE_ABS_2

Example

Suppose that the current position is (0,0). The instruction (RECTANGLE_ABS, 2, 3) is equivalent to:

```
LINE_ABS_2, 2, 0
LINE_ABS_2, 2, 3
LINE_ABS_2, 0, 3
LINE_ABS_2, 0, 0
```

7.1.8 RECTANGLE_REL_2 - Draw Rectangle by Relative Position

The RECTANGLE_REL_2 instruction draws a series of connected lines forming a four-sided, perpendicular, polygon with the current position at one corner and a point specified as an offset in world coordinate space at the opposing corner. It does not change the current position.

CORE Standard

Not included.

CORE Graphics Library

RECTANGLE_REL_2 (dx, dy)

The parameters are real expressions representing world coordinates.

dx specifies an X (horizontal) offset describing a rectangle.

dy specifies a Y (vertical) offset describing a rectangle.

Notes

- RECTANGLE_REL_2 is instruction number 11.

Example

Suppose that the current position is (0,0). An instruction (RECTANGLE_REL, 2, 3) is equivalent to:

```
LINE_REL_2, 2, 0
LINE_REL_2, 0, 3
LINE_REL_2, -2, 0
LINE_REL_2, 0, -3
```


CURVED LINE PRIMITIVE INSTRUCTIONS

7.2 CURVED LINE PRIMITIVE INSTRUCTIONS

Arc and curve primitive instructions draw curved lines by interpolation.

7.2.1 ARC_ABS_2 - Draw Arc Based on Absolute Position

The ARC_ABS_2 instruction draws an arc of a circle whose center is at a specified world coordinate position. The arc begins at the current position and continues for a specified number of degrees. CGL updates the current position to the last point on the arc.

Unlike other output primitives, arcs do not change their shape in the viewing transformation. The circle described by the ARC_ABS_2 instruction is always a perfect circle, regardless of whether the window is the same shape as the viewport.

CORE Standard

Not included.

CORE Graphics Library

ARC_ABS_2 (x, y, angle)

x is a real expression that specifies the X world coordinate of the center of the circle.

y is a real expression that specifies the Y world coordinate of the center of the circle.

angle is an integer expression that specifies the angle (in degrees) of the arc.

Notes

- ARC_ABS_2 is instruction number 39.
- Positive angles cause the arc to be drawn counterclockwise. Negative angles cause the arc to be drawn clockwise.
- The angle can be any number up to machine infinity. CGL uses the specified angle value modulo 360.
- Because of rounding errors, a series of consecutive arcs does not necessarily describe a circle. For example, six consecutive 60-degree arcs do not join at the starting point. If you want an accurate circle, you must specify a 360-degree

ARC_ABS_2

arc.

- You can compute the radius of the circle with the Pythagorean Theorem. For example, in PASCAL:

```
arc_abs_2 (x1, y1, n);
inquire_current_position_2 (x2, y2);
radius := sqrt(sqr(abs(x1 - x2)) + sqr(abs(y1 - y2)));
```

Example

Figure 7-1 shows what the window would look like if you executed the following instructions:

```
SET_WINDOW (0.0, 9.0, 0.0, 14.0)
MOVE_ABS_2 (8.0, 9.0)
ARC_ABS_2 (4.0, 5.0, 90)
```

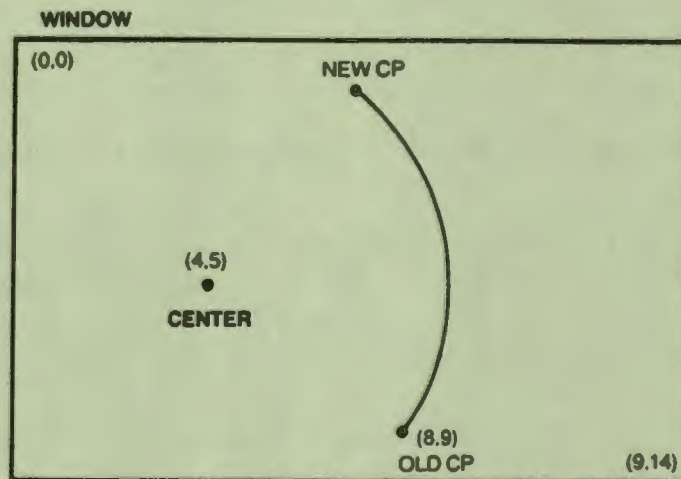


Figure 7-1: An Arc in World Coordinate Space

7.2.2 ARC_REL_2 - Draw Arc Based on Relative Position

The ARC_REL_2 instruction draws an arc of a circle whose center is a specified offset from the current position. The arc begins at the current position and continues for a specified number of degrees. CGL updates the current position to the last point on the arc.

Unlike other output primitives, arcs do not change their shape in the viewing transformation. The circle described by the ARC_REL_2 instruction is always a perfect circle, regardless of

ARC_REL_2

whether the window is the same shape as the viewport.

CORE Standard

Not included.

CORE Graphics Library

ARC_REL_2 (x, y, angle)

x is a real expression that specifies the center of the circle as an X offset from the current position.

y is a real expression that specifies the center of the circle as a Y offset from the current position.

angle is an integer expression that specifies the angle (in degrees) of the arc.

Notes

- ARC_REL_2 is instruction number 40.
- Positive angles cause the arc to be drawn counterclockwise. Negative angles cause the arc to be drawn clockwise.
- The angle can be any number up to machine infinity. CGL uses the specified angle modulo 360.
- Because of rounding errors, a series of consecutive arcs does not necessarily describe a circle. For example, six consecutive 60-degree arcs do not join at the starting point. If you want an accurate circle, you must specify a 360-degree arc.
- You can compute the radius of the circle with the Pythagorean Theorem. For example, in PASCAL:

```
arc_rel_2 (x1, y1, n);  
radius = sqrt(sqr(abs(x1)) + sqr(abs(y1)));
```

Example

Figure 7-1 shows what the window would look like if you executed the following instructions:

```
SET_WINDOW (0.0, 9.0, 0.0, 14.0)  
MOVE_ABS_2 (8.0, 9.0)  
ARC_REL_2 (-4.0, -3.0, 90)
```

7.2.3 CURVE_ABS_2 - Draw Curve by Absolute Positions

The CURVE_ABS_2 instruction draws a smooth curve connecting a list of world coordinate positions. You can specify an open or closed curve.

CGL begins Drawing at the first position in the specified list and continues to the last position. If you specify an open curve, CGL stops drawing there. If you specify a closed curve, CGL continues the curve back to the first position in the list. In either case, CGL updates the current position to the end of the curve.

CORE Standard

Not included.

CORE Graphics Library

CURVE_ABS_2 (x_array, y_array, n, type)

x_array is a subscripted real variable that specifies a list of X world coordinate positions.

y_array is a subscripted real variable that specifies a list of Y world coordinate positions.

n is an integer expression that specifies the number of elements in each array.

type is an integer expression that specifies one of the following values:

0 = open curve anything else = closed curve

Notes

- CURVE_ABS_2 is instruction number 41.
- If you are drawing a closed curve with polygon fill ON, the maximum number of points that can be on the curve is 28.

Errors

- 3. N is less than or equal to two.
- 904. Too many points in closed, filled figure.

7.2.4 CURVE_REL_2 - Draw Curve by Relative Positions

The CURVE_REL_2 instruction draws a smooth curve connecting a list of offsets in world coordinate space. You can specify an open or closed curve.

CGL begins drawing at the first offset in the supplied list and continues to the last offset. If you specify an open curve, CGL stops drawing there. If you specify a closed curve, CGL continues the curve back to the position described by the first offset in the list. In either case, CGL updates the current position to the end of the curve.

CORE Standard

Not included.

CORE Graphics Library

CURVE_REL_2 (x_array, y_array, n, type)

x_array is a subscripted real variable that specifies a set of X offsets in world coordinate space.

y_array is a subscripted real variable that specifies a set of Y offsets in world coordinate space.

n is an integer expression that specifies the number of elements in each array.

type is an integer expression that specifies one of the following values:

0 = open curve anything else = closed curve

Notes

- CURVE_REL_2 is instruction number 42.

Errors

3. N is less than or equal to two.
904. Too many points in closed, filled figure.

LINE ATTRIBUTE INSTRUCTIONS

7.3 LINE ATTRIBUTE INSTRUCTIONS

Line attribute instructions affect the appearance of the images produced by both straight and curved line primitive instructions.

7.3.1 SET_LINestyle - Set Line Drawing Style

The SET_LINestyle instruction sets the style of lines drawn by line drawing instructions. You specify one of the nine standard line styles or a user-defined line style. Figure 7-2 shows the nine standard line styles.

CORE Standard

SET_LINestyle (linestyle)

INQUIRE_LINestyle (linestyle)

CORE Graphics Library

SET_LINestyle (style, pattern, mult)

INQUIRE_LINestyle (style, pattern, mult)

The parameters are integer expressions/variables.

style specifies one of nine standard line styles (pattern and mult are ignored) or a user-defined style.

pattern specifies a 16-bit user-defined pattern where the set bits are "on" and the clear bits are "off".

mult specifies how many times to draw each bit in the pattern.

Notes

- SET_LINestyle is instruction number 12.
- INQUIRE_LINestyle is instruction number 13.
- Any style value less than one or greater than nine indicates a user-defined style.
- The default line style is 1 (SOLID).

SET_LINestyle

- The multiplier operates on individual bits. For example, suppose the pattern is:

10101010101010

A multiplier of two would produce:

11001100110011001100110011001100

A multiplier of three would produce:

111000111000111000111000111000111000111000111000

- SET_LINestyle works differently when used with plotter view surfaces (see Appendix B).

STYLE		BIT PATTERN															
NO.	NAME	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	SOLID																
2	DASHED																
3	DOT_DASHED																
4	DOTTED																
5	DOT_DOT_DASHED																
6	DOTTED_WIDE_SPACING																
7	DASHED_SHORT_LINES																
8	DASHED_LONG_LINES...																
9	DOT_DASHED_SHORT_LINES																

Figure 7-2: The Standard Line Styles

7.3.2 SET_LINEWIDTH - Set Line Drawing Width

The SET_LINEWIDTH instruction specifies the width of line primitives in world coordinate units. You can set the X and Y line widths independently.

SET_LINEWIDTH

CORE Standard

SET_LINEWIDTH (linewidth)

INQUIRE_LINEWIDTH (linewidth)

CORE Graphics Library

SET_LINEWIDTH (dx, dy)

INQUIRE_LINEWIDTH (dx, dy)

The parameters are real expressions/variables representing world coordinate units.

dx specifies the X (horizontal) width of lines created by line primitive instructions.

dy specifies the Y (vertical) width of lines created by line primitive instructions.

Notes

- SET_LINEWIDTH is instruction number 14.
- INQUIRE_LINEWIDTH is instruction number 15.
- The default line width is (dx = 0, dy = 0).
- A line width parameter less than or equal to zero sets the line width to one physical device coordinate unit.
- The drawing speed is noticeably slower if the line width is wider than one physical device coordinate unit.
- Complement mode does not work correctly if the line width is wider than one physical device coordinate unit.
- SET_LINEWIDTH works differently when used with plotter view surfaces (see Appendix B).

NOTE

The following feature is supported only for compatibility with earlier versions of CGL. It is recommended that you use SET_LINEWIDTH_ORIENTATION to control line positioning. Once executed, it always overrides the following.

SET_LINEWIDTH

- You can use the signs of the line width parameters to control a line's exact starting position relative to the drawing position and the origin of the window. CGL draws the end of a line as a rectangle with dimensions determined by the line width parameters. You can specify which of the four corners of the rectangle is to appear at the current position. Assuming the default origin (top-left):

dx	dy	corner
+	+	lower left
-	+	lower right
+	-	upper left
-	-	upper right

7.3.3 SET_LINEWIDTH_ORIENTATION - Set Line Endpoint Offset

The SET_LINEWIDTH_ORIENTATION instruction specifies the offset from the end of a line primitive to the actual drawing position specified in the line primitive instruction. CGL draws the end of a line as a rectangle with dimensions determined by the current vertical and horizontal line width. (This can be clearly seen in Figure 7-3.) You can specify X and Y offsets from the bottom-left corner of that rectangle to the actual drawing position.

CORE Standard

Not included.

CORE Graphics Library

SET_LINEWIDTH_ORIENTATION (dx, dy)

INQUIRE_LINEWIDTH_ORIENTATION (dx, dy)

The parameters are real expressions/variables in the range zero to one.

dx specifies the X (horizontal) offset from the upper-left corner of the end-point rectangle to the current position.

dy specifies the Y (vertical) offset from the upper-left corner of the end-point rectangle to the current position.

SET_LINEWIDTH_ORIENTATION

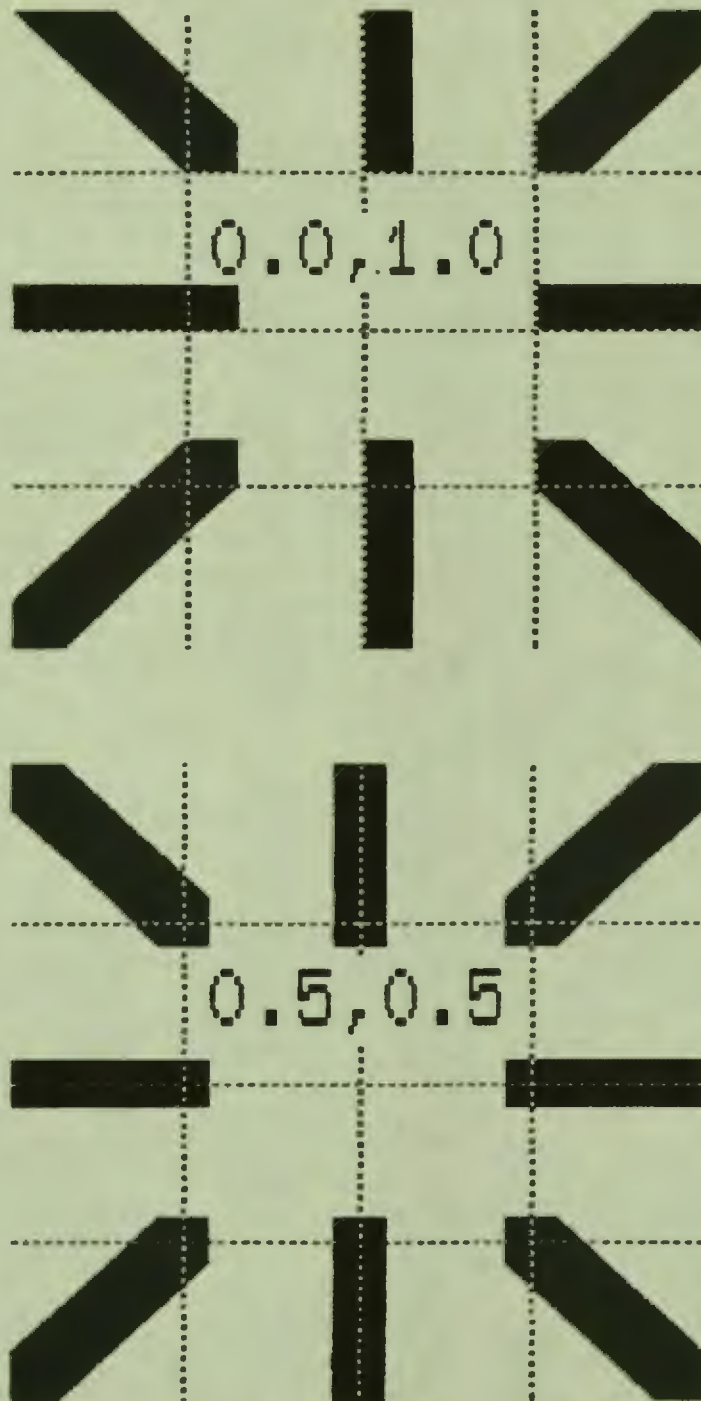


Figure 7-3: Line Width Orientations: Bottom-left and Centered

SET_LINEWIDTH_ORIENTATION

Notes

- SET_LINEWIDTH_ORIENTATION is instruction number 98.
- INQUIRE_LINEWIDTH_ORIENTATION is instruction number 99.
- The default line width orientation is the top-left corner (dx = 0, dy = 0).
- To center the end point over the drawing position, use (0.5, 0.5), as shown in Figure 7-3.

Errors

505. A parameter is not in the range 0 to 1.

7.3.4 SET_FILL_MODE - Enable or Disable Area Fill

The SET_FILL_MODE instruction selects the current area fill mode (or disables fill). When fill is enabled, CGL shades (fills in with a character) areas described by subsequent line primitives. You can use "polygon fill" mode with closed line primitives (a polygon, rectangle, closed arc, or closed curve) or fill to a specified line or point (see the SET_FILL_ENTITY instruction).

CORE Standard

Not included.

CORE Graphics Library

SET_FILL_MODE (mode)

INQUIRE_FILL_MODE (mode)

mode is an integer expression/variable that specifies one of the following values:

- 0 = fill off (default)
- 1 = fill to a vertical line
- 2 = fill to a horizontal line
- 3 = fill to a point
- 4 = fill polygons

SET_FILL_MODE

Notes

- SET_FILL_MODE is instruction number 74.
- INQUIRE_FILL_MODE is instruction number 75.
- This is a limit of 256 points in any filled polygon except for a closed curve, which has a limit of 28 points.

Errors

401. One or more of the attribute values is invalid.

7.3.5 SET_FILL_ENTITY - Specify Line or Point for Fill Reference

The SET_FILL_ENTITY instruction sets the line or point used as the reference for area fill (see SET_FILL_MODE).

CORE Standard

Not included.

CORE Graphics Library

SET_FILL_ENTITY (x, y)

INQUIRE_FILL_ENTITY (x, y)

The parameters are real expressions/variables representing absolute world coordinates.

x specifies the X value of the fill entity.

y specifies the Y value of the fill entity.

Notes

- SET_FILL_ENTITY is instruction number 76.
- INQUIRE_FILL_ENTITY is instruction number 77.
- The default fill entity coordinate pair is the origin of the window.
- If the fill mode is 1, the X value (a point on the horizontal axis) describes a vertical line.

SET_FILL_ENTITY

- If the fill mode is 2, the Y value (a point on the vertical axis) describes a horizontal line.
- If the fill mode is 3, the X and Y values describe a point.
- The fill entity does not have to be within the window.
- If polygon fill is enabled, the fill coordinates are not actually used when filling but are stored for a future change of fill mode.

7.3.6 SET_FILL_CHAR - Specify Character for Fill

The SET_FILL_CHAR specifies the character used for area fill.

CORE Standard

Not included.

CORE Graphics Library

SET_FILL_CHAR (font, char, width_mult, height_mult)

INQUIRE_FILL_CHAR (font, char, width_mult, height_mult)

The parameters are integer expressions/variables.

font specifies the number of the font containing the fill character.

char specifies the numeric code of the character.

width_mult specifies a multiplier on the standard character width.

height_mult specifies a multiplier on the standard character height.

Notes

- SET_FILL_CHAR is instruction number 78.
- INQUIRE_FILL_CHAR is instruction number 79.
- The default fill character is a special case; in fact it's not a character at all but a vertically-oriented version of the current line style.

SET_FILL_CHAR

- SET_FILL_CHAR works differently when used with plotter view surfaces (see Appendix B).

Errors

401. One or more of the attribute values is invalid.



CHAPTER 8

TEXT INSTRUCTIONS

This chapter describes text primitive and attribute instructions.

8.1 TEXT PRIMITIVE INSTRUCTIONS

8.1.1 TEXT - Draw Line of Text

The TEXT instruction draws a line of text. Unlike most other output primitives, text does not change the current position.

CORE Standard

TEXT (character_string)

CORE Graphics Library

TEXT (string, length)

string is a string expression.

length is an integer expression representing the number of characters in the string expression.

Notes

- TEXT is instruction number 16.
- In PASCAL, "TEXT" is a predeclared identifier, thus the name of this instruction is "TXT".

Errors

8. TEXT error, $N < 0$ or $\text{extent} > 32767$.

TEXT

208. The string contains one or more undefined characters.

8.1.2 INQUIRE_TEXT_EXTENT_2 - Report Position at End of String

The INQUIRE_TEXT_EXTENT_2 instruction reports the amount of world coordinate space that would be used to draw a string of the indicated length, unjustified, beginning at the current position. The current text attribute settings are used to compute the string extent vector. Nothing is drawn or changed.

CORE Standard

INQUIRE_TEXT_EXTENT_2 (character_string, surface_name, dx, dy)

CORE Graphics Library

INQUIRE_TEXT_EXTENT_2 (length, delta_x, delta_y)

length is an integer expression representing the number of characters in the string.

delta_x receives the X extent in world coordinate units.

delta_y receives the Y extent in world coordinate units.

Notes

- INQUIRE_TEXT_EXTENT_2 is instruction number 17.

8.1.3 LOAD_FONT - Load User-defined Font

The LOAD_FONT instruction loads characters into the currently selected font from a named region in memory. The format of this region is described in the PRO/GIDIS Manual (order no. AA-Y660A-TK).

CORE Standard

Not included.

CORE Graphics Library

LOAD_FONT (name, length)

name is a string expression.

LOAD_FONT

length is an integer value or expression that specifies the length of the name string.

Notes

- LOAD_FONT is instruction number 112.
- The name must correspond to the installed region name and should only contain characters A through Z (upper or lower case), and 0 through 9.
- The name length must be greater than zero and less than or equal to six.
- You must call SET_FONT_SIZE before using this instruction in order to pass the extent and size information to CGL (even though this information forms part of the file header).
- If a font could not be loaded, font 0 is loaded by default.
- Characters cannot be defined for the plotter using this instruction. Use begin/end define character.

Errors

- 9. Font 0 cannot be redefined.
- 401. One or more of the attribute values is invalid.
- 910. Invalid when in begin/end define character.
- 912. Font could not be loaded.

8.1.4 LOAD_CHARACTER - Load User-defined Character

The LOAD_CHARACTER instruction loads a character into the current (user-defined) font.

CORE Standard

Not included.

CORE Graphics Library

LOAD_CHARACTER (code, matrix)

code is an integer expression that specifies a DEC Multinational Character Set decimal code. The valid

LOAD_CHARACTER

codes range from 32 to 126 (GL less the delete character) and from 161 to 255 (GR). You cannot load characters that correspond to C0, the delete character, or C1.

matrix is an integer array variable that specifies the physical device coordinate unit pattern of the character.

Notes

- **LOAD_CHARACTER** is instruction number 32.
- You must execute the **SET_FONT_SIZE** instruction before using the **LOAD_CHARACTER** instruction.
- The character code value must be less than or equal to the extent specified in the **SET_FONT_SIZE** instruction.
- The number of elements in the matrix value must correspond to the **y_size** value specified in the **SET_FONT_SIZE** instruction.
- If the **x_size** specified in the **SET_FONT_SIZE** instruction is less than 16, CGL uses the high-order bits in each array element.
- If the **y_size** specified in the **SET_FONT_SIZE** instruction is less than the number of elements in the array, CGL uses the lower-numbered array elements.
- You cannot define characters on the plotter view surface using this instruction. Use the begin/end define character sequence.

Errors

- 9. Font 0 cannot be redefined.
- 401. One or more of the attribute values is invalid.
- 910. Invalid when in begin/end define character.

8.1.5 BEGIN_DEFINE_CHARACTER

This instruction provides an alternate way of loading a character into the current (user defined) font. Instructions between the **BEGIN_DEFINE_CHARACTER** and the **END_DEFINE_CHARACTER** instructions are used to describe the character. The world coordinates of

BEGIN_DEFINE_CHARACTER

output primitives and attribute sizes are mapped to the character dimensions specified in SET_FONT_SIZE.

CORE Standard

Not included.

CORE Graphics Library

BEGIN_DEFINE_CHARACTER (code)

code is an integer expression that specifies a DEC Multinational Character Set decimal code. The valid codes range from 32 to 126 (GL less the delete character) and from 160 to 255 (GR). You cannot load characters that correspond to C0, the delete character, or C1.

Notes

- BEGIN_DEFINE_CHARACTER is instruction number 109.
- You must execute the SET_FONT_SIZE instruction before using this instruction.
- The character code value must be less than or equal to the extent specified in the SET_FONT_SIZE instruction.
- Use this instruction to define characters to be used on the plotter view surface.
- Some instructions are invalid within BEGIN and END define character. Examples are LOAD_CHARACTER, SET_CURSOR and SELECT_VIEW_SURFACE.

Errors

- 9. Font 0 cannot be redefined.
- 401. One or more of the attribute values is invalid.
- 910. Invalid when in begin/end define character.

8.1.6 END_DEFINE_CHARACTER

This instruction terminates the definition of a character.

END_DEFINE_CHARACTER

CORE Standard

Not included.

CORE Graphics Library

END_DEFINE_CHARACTER

Notes

- END_DEFINE_CHARACTER is instruction number 110.

Errors

910. Invalid when in begin/end define character.

8.2 TEXT ATTRIBUTE INSTRUCTIONS

8.2.1 SET_CHARSIZE - Set Character Size

The SET_CHARSIZE instruction sets the size, in world coordinate units, of the characters drawn by subsequent TEXT instructions. You can set the X and Y sizes independently. (See the detailed discussion of character size in Chapter 1.)

CORE Standard

SET_CHARSIZE (charwidth, charheight)

INQUIRE_CHARSIZE (charwidth, charheight)

CORE Graphics Library

SET_CHARSIZE (width, height)

INQUIRE_CHARSIZE (width, height)

The parameters are real expressions/variables representing world coordinate units.

width specifies the X (horizontal) size of the character.

height specifies the Y (vertical) size of the character.

Notes

SET_CHARSIZE

- SET_CHARSIZE is instruction number 20.
- INQUIRE_CHARSIZE is instruction number 21.
- The default width (12) and height (25) values produce graphics characters that appear the same size as text mode characters using the default window (0, 959, 0, 599). CGL's default differ from the CORE Standard, which specifies a default of 100 lines of 100 characters.
- Negative width or height values cause CGL to invert the characters. In other words, a negative width value produces characters that are backwards and a negative height value produces characters that are upside-down.
- If the specified character size is smaller than the default character size, the terminal subsystem draws the characters "half size" by using every other physical device coordinate unit.

8.2.2 SET_CHARSPACE - Set Character Spacing

The SET_CHARSPACE instruction specifies the displacement between the starting points of adjacent letters. The displacement can be horizontal, or vertical, or both.

CORE Standard

SET_CHARSPACE (charspace)

INQUIRE_CHARSPACE (charspace)

CORE Graphics Library

SET_CHARSPACE (delta_x, delta_y)

INQUIRE_CHARSPACE (delta_x, delta_y)

The parameters are real expressions/variables representing world coordinate units.

delta_x specifies the X offset between characters.

delta_y specifies the Y offset between characters.

SET_CHARSPACE

Notes

- SET_CHARSPACE is instruction number 24.
- INQUIRE_CHARSPACE is instruction number 25.
- The default delta_x value is 12 (the same as the default character width). The default delta_y value is zero (no vertical offset).
- In string mode CGL adjusts the spacing to maintain the current character path (see SET_CHARPATH).

8.2.3 SET_CHARPATH - Set Text Writing Direction

The SET_CHARPATH instruction has two modes: character and string.

In character mode, SET_CHARPATH changes the angle (relative to horizontal) in which CGL draws individual characters. The character spacing is set to that last explicitly defined by a SET_CHARSPACE instruction.

In string mode, SET_CHARPATH changes the angle (relative to horizontal) in which CGL draws individual characters and adjusts the character spacing so that characters are drawn along the base line described by the character angle (see Figure 1-13).

CORE Standard

SET_CHARPATH (charpath)

INQUIRE_CHARPATH (charpath)

CORE Graphics Library

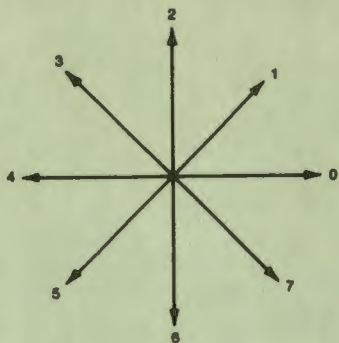
SET_CHARPATH (path, mode)

INQUIRE_CHARPATH (path, mode)

The parameters are integer expressions/variables.

path specifies one of the following values:

SET_CHARPATH



mode specifies one of the following values:

- 0 = character mode
- anything else = string mode (default)

Notes

- SET_CHARPATH is instruction number 22.
- INQUIRE_CHARPATH is instruction number 23.
- The default character path is zero.
- The angles (counterclockwise from horizontal) that correspond to the eight possible paths are:

<u>path</u>	<u>angle</u>
0	0
1	51
2	90
3	129
4	180
5	132
6	270
7	309

- Characters drawn with a diagonal path appear somewhat italic. If necessary, use SET_CHARITALIC to compensate.

SET_CHARPATH

8.2.4 SET_CHARJUST - Set Text Justification

The SET_CHARJUST instruction specifies the starting position of text primitives relative to the current position. It allows horizontal and vertical justification and centering.

CORE Standard

SET_CHARJUST (charjust)

INQUIRE_CHARJUST (charjust)

CORE Graphics Library

SET_CHARJUST (x_just, y_just)

INQUIRE_CHARJUST (x_just, y_just)

x_just is an integer expression/variable that specifies one of the following X (horizontal) text justification values:

- 1 = left (default)
- 2 = center
- 3 = right

y_just is an integer expression/variable that specifies one of the following Y (vertical) text justification values:

- 1 = top (default)
- 2 = center
- 3 = bottom

Notes

- SET_CHARJUST is instruction number 26.
- INQUIRE_CHARJUST is instruction number 27.

8.2.5 SET_CHARITALIC - Set Character Slant

The SET_CHARITALIC instruction changes the shape of the individual characters in a text string to resemble italic type. The characters can have a forward or backward slant.

CORE Standard

Not included.

SET_CHARITALIC

CORE Graphics Library

SET_CHARITALIC (angle)

INQUIRE_CHARITALIC (angle)

angle is an integer expression/variable that specifies an angle (in degrees) of slant.

Notes

- SET_CHARITALIC is instruction number 28.
- INQUIRE_CHARITALIC is instruction number 29.
- A negative angle specifies a forward (right) slant. A positive angle specifies backward (left) slant.
- You should confine the angle of slant to the range -40 to 40 for readability.
- The default angle is zero (vertical).

8.2.6 SET_FONT - Select Character Font

The SET_FONT instruction selects one of the four character fonts available to your program.

Font 0 contains the DEC Multinational Character Set (GL and GR) and cannot be redefined.

Fonts 1 through 3 are user-defined fonts in which you can load your own special characters. Refer to the SET_FONT_SIZE, LOAD_FONT, LOAD_CHARACTER, and BEGIN/END_DEFINE_CHARACTER instructions for more information about user-defined fonts.

CORE Standard

SET_FONT (font)

INQUIRE_FONT (font)

CORE Graphics Library

SET_FONT (font)

INQUIRE_FONT (font)

SET_FONT

font is an integer expression/variable that specifies a value in the range 0 to 3.

Notes

- SET_FONT is instruction number 18.
- INQUIRE_FONT is instruction number 19.
- Font 0 (DEC Multinational) is the default.
- SET_FONT works differently when used with plotter view surfaces (see Appendix B).

8.2.7 SET_FONT_SIZE - Define Size of Character Font

The SET_FONT_SIZE instruction initializes the current user-defined font. It establishes the size of the font by specifying the highest DEC Multinational Character Set decimal code (the lowest is always 32) and specifies the size of the characters in physical device coordinate units.

When you execute SET_FONT_SIZE, CGL passes the font size and subsequent character definitions to all currently selected view surfaces. If a view surface is not selected at the time the font is defined, it cannot access the font.

CORE Standard

Not included.

CORE Graphics Library

SET_FONT_SIZE (extent, x_size, y_size)

INQUIRE_FONT_SIZE (extent, x_size, y_size)

The parameters are integer expressions/variables.

extent specifies the highest decimal code in the font. The valid codes for a user-defined font range from 32 to 126 (GL less the delete character) and from 161 to 255 (GR). You cannot define characters that correspond to C0, the delete character, or C1.

x_size specifies the width of the font's characters in physical device coordinate units (range 1 to 16).

SET_FONT_SIZE

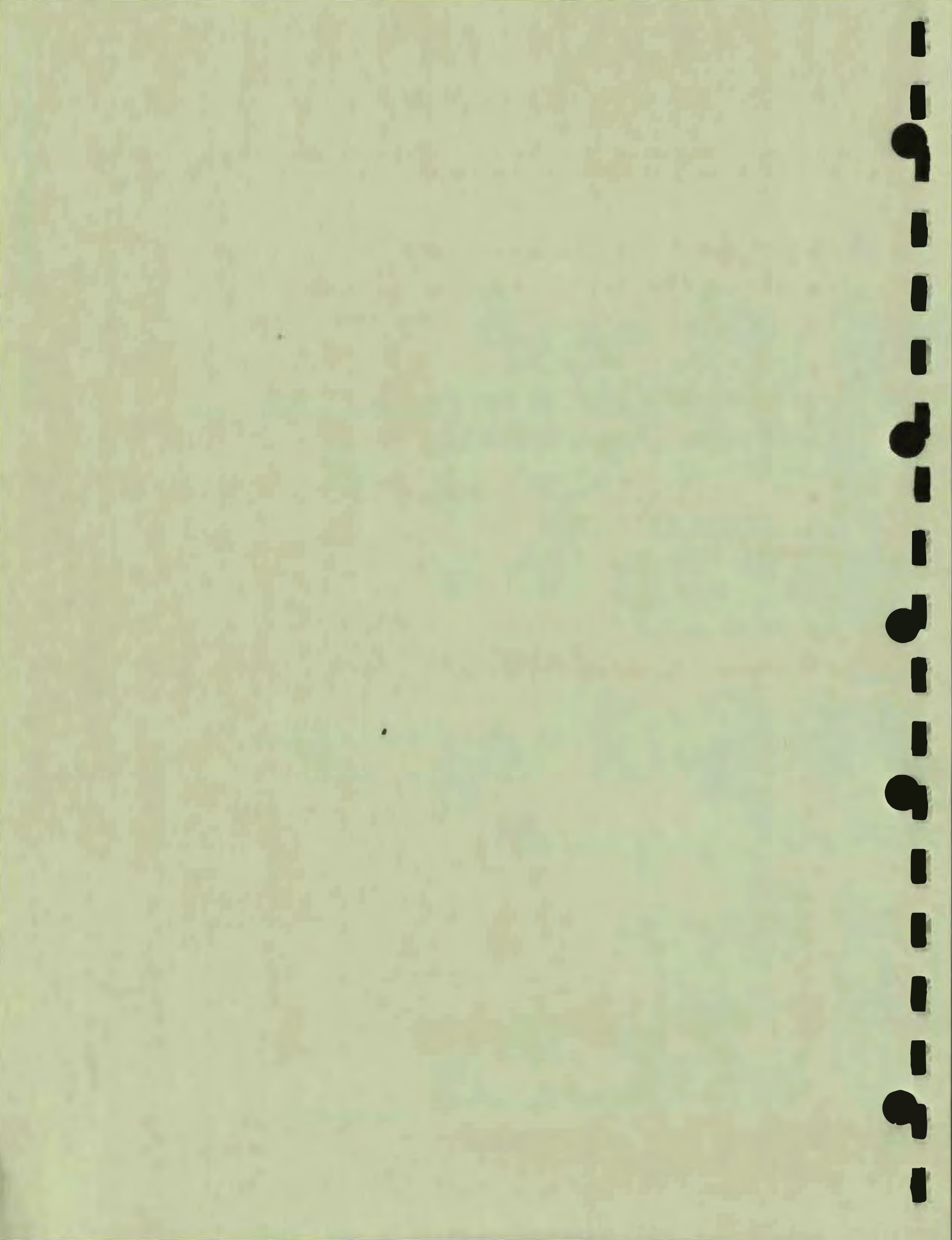
`y_size` specifies the height of the font's characters in physical device coordinate units (range 1 to 16).

Notes

- `SET_FONT_SIZE` is instruction number 30.
- `INQUIRE_FONT_SIZE` is instruction number 31.
- The extent of font 0 is 255.
- The `x_size` of font 0 is 12.
- The `y_size` of font 0 is 10.
- An extent value less than 32 clears the specified font (except font 0).
- `SET_FONT_SIZE` works differently when used with plotter view surfaces (see Appendix B).

Errors

- 9. Font 0 cannot be redefined.
- 401. One or more of the attribute values is invalid.
- 902. There is insufficient space for the font.



APPENDIX A
ERROR MESSAGES

- 0. There is no error.
- 2. N is less than or equal to zero.
- 3. N is less than or equal to two.
- 8. TEXT error, $N < 0$ or extent > 32767 .
- 9. Font 0 cannot be redefined.
- 208. The string contains one or more undefined characters.
- 401. One or more of the attribute values is invalid.
- 501. Invalid coordinate values (minimum \geq maximum).
- 503. SET_NDC_SPACE_2 already invoked since initialization.
- 504. Default NDC space already established.
- 505. A parameter is not in the range 0 to 1.
- 506. Neither width nor height has a value of 1.
- 507. Neither WIDTH nor HEIGHT can be equal to zero.
- 508. A value outside NDC space is not allowed.
- 701. The CORE system is already initialized.
- 705. View surface already initialized.
- 706. Invalid view surface name.

ERROR MESSAGES

- 708. View surface not initialized.
- 709. View surface already selected.
- 711. View surface not selected.
- 716. There has been no END_BATCH since the last BEGIN_BATCH.
- 717. There has been no corresponding BEGIN_BATCH.
- 743. The CORE system has not been initialized.
- 900. Function number out of range or wrong number of parameters.
- 902. There is insufficient space for the font.
- 903. I/O error (unassigned LUN, etc.).
- 904. Too many points in closed, filled figure.
- 906. Error on view surface device.
- 907. Invalid when in begin/end batch.
- 908. View surface not ready.
- 909. Function not implemented.
- 910. Invalid when in begin/end define character.
- 911. Error on file playback (file not found, etc).
- 912. Font could not be loaded.

APPENDIX B
OPTIONAL VIEW SURFACES

This appendix provides information specific to view surfaces other than the Professional 300 Series video monitor.

B.1 HEWLETT-PACKARD HP7470A AND HP7475A GRAPHICS PLOTTERS

The following sections describe all the differences between the way CGL works with a plotter and with the other view surfaces. If an instruction is not mentioned, it performs exactly as specified in Chapters 3 through 8.

The plotter pen normally operates at 38 cm/s. CGL can "feed" the plotter fast enough to keep it active.

In some cases, exact support for CGL instructions would require too much computation, reduce the quality or speed of output, or put unnecessary stress on the plotter. In those cases, suitable "fallbacks" have been devised; the instructions perform somewhat differently on the plotter than they do on other view surfaces. Other instructions simply have no effect on a plotter view surface at all.

B.1.1 Hardware Requirements

To connect a plotter to the printer port, you must have a standard DEC printer cable (BCC05). If you want to connect a plotter only, you must have a DEC Male-Male cable (BC22H). If you want to connect a printer and a plotter to the same system, you must have the "Eavesdrop" cable supplied by Hewlett-Packard (07470-60090).

A printer cable (BCC20) with a male connector at the printer (or plotter) end will become available in early 1984. It will eliminate the need for the Male-Male cable.

HEWLETT-PACKARD PLOTTERS

B.1.2 Setting Up the Plotter

This list supplements the Hewlett-Packard documentation in describing how to set the plotter's rocker switches:

- Switches B1 thru B4 control baud rate. Specify 4800 baud by setting B1 and B4 on (to B1 and B4).
- The next two switches on the 7475 and the next switch on the 7470 control paper size. See the HP operator's manual for details.
- The Y/D switch controls cabling. If you are using the Male-Male cable, set it to off (D). If you are using the "Eavesdrop" cable, set it to on (Y).
- The S1 and S2 switches control byte size and parity. Set both to off (eight-bit bytes and no parity checking).

B.1.3 Physical Device Coordinate Space

Physical device coordinate space for the HP7470A is 1000 x 720 (paper sizes A and A4). Physical device coordinate space for the HP7475A is either 1000 x 720 or 1520 x 1000 (paper sizes B and A3). These coordinate spaces have been set so that one unit is approximately the line width drawn by the .3 mm pen supplied by Hewlett-Packard.

B.1.4 Inoperative Instructions

The following instructions have no effect on plotter output.

- LOAD_FONT
- LOAD_CHARACTER
- NEW_FRAME
- SCROLL
- SCROLL_VIEWPORT
- SET_COLOR_MAP

HEWLETT-PACKARD PLOTTERS

- SET_COLOR_MAP_ENTRY
- SET_CURSOR
- SET_WRITING_PLANES

B.1.5 SET_WRITING_INDEX

Writing index to pen mapping was chosen to maximize potential compatibility with the video color map. SET_WRITING_INDEX specifies pens as follows:

7470A plotter (two pens)

- 1 = left pen
- 2 = right pen
- 3 = left pen at two-thirds speed
- 4 = right pen at two-thirds speed
- 5 = left pen
- 6 = right pen
- 7 = left pen

7475A plotter (six pens)

- 1 = pen 1
- 2 = pen 2
- 3 = pen 3
- 4 = pen 4
- 5 = pen 1 at two-thirds speed
- 6 = pen 5
- 7 = pen 6

A writing index value of zero for either plotter is described in the section on SET_BACKGROUND_INDEX.

Slowing the pen down to two-thirds full speed thickens and darkens a line slightly, particularly with a fresh pen.

If fill is enabled, CGL uses the writing index to determine the alignment of the hatch lines in order to maximize differentiation between hatch lines drawn in different colors. The difference in alignment between writing index n and writing index $n+2$ is one unit.

HEWLETT-PACKARD PLOTTERS

B.1.6 SET_BACKGROUND_INDEX

In the context of a plotter-only application, the color of the background is the color of the paper currently being used. Thus, there is no reason to change the background index. Other view surfaces, however, may require different background indexes. Thus, while the background index does not by itself specify a pen, changing it will in some circumstances temporarily remap pens to minimize the chance that adjacent areas will accidentally be the same color.

- If the writing index and the background index are both zero, CGL draws with the right-hand pen on the HP7470A and pen six on the HP7475A.
- If the writing index is zero and the background index is non-zero, CGL draws with the pen specified by the background index.
- If the writing index and the background index are the same non-zero number, CGL draws with the specified pen.

Setting the background index to a value of eight will slow the pen down. This is particularly suitable for plotting on transparency material.

B.1.7 SET_WRITING_MODE

All writing modes are mapped to OVERLAY or TRANSPARENT. TRANSPARENT, TRANSPARENT_NEGATE, ERASE, and ERASE_NEGATE are treated as TRANSPARENT. All other modes are treated as OVERLAY.

B.1.8 SET_MARKER_SYMBOL

If you set the current font to a font whose extent is less than or equal to 60, CGL uses a special font consisting of 20 markers:

0 = lower-case x	10 = square
1 = lower-case o	11 = diamond
2 = plus sign	12 = filled square
3 = star	13 = filled diamond
4 = double dagger	14 = pi
5 = asterisk	15 = up arrow
6 = sideways H	16 = down arrow
7 = triangle	17 = left arrow
8 = inverted triangle	18 = right arrow
9 = crosshatch	19 = check mark

HEWLETT-PACKARD PLOTTERS

If you specify a marker symbol character greater than 19, CGL uses the number modulo 20.

B.1.9 SET_LINestyle

The style parameter specifies one of the following, built-in plotter line styles. These resemble but do not match the video monitor line styles.

- 1 = SOLID
- 2 = DASHED_LONG_LINES
- 3 = DOT_DOT_DASHED
- 4 = DASHED_SHORT_LINES
- 5 = DOT_DOT_DASHED
- 6 = DOTTED_WIDE_SPACING
- 7 = DASHED
- 8 = DOT_DASHED
- 9 = DASHED

The size of the line style pattern is set to the value specified in the command, with a minimum of about .125 inches. The pattern is rotated rather than projected when a diagonal line is drawn.

B.1.10 SET_LINEWIDTH

Actual line width is only an approximation because of the nature of the hardware. A line width of one approximates the .3mm pens supplied by Hewlett-Packard.

B.1.11 SET_FILL_CHAR

Fill characters are mapped to a special set of hatch patterns. There are four specific cases:

- SET_FILL_CHAR (0, 0, ...)

This specifies horizontal hatch lines about .04 inches apart that are drawn using the current linestyle.

- SET_FILL_CHAR (n, 32, ...)

This specifies solid fill. Parameter "n" is a integer in the range one to three representing a user-defined font.

HEWLETT-PACKARD PLOTTERS

- SET_FILL_CHAR (0, c, ...)

This specifies one of the hatch patterns shown in Table B-1. Parameter "c" is a positive integer representing a character code. CGL uses the character code specified.

- SET_FILL_CHAR (n, c, ...)

This specifies one of the hatch patterns shown in Table B-1. Parameter "n" specifies a user-defined font and "c" specifies a character code. CGL maps the character code as described under the LOAD_CHARACTER instruction in Chapter 8. For example, SET_FILL_CHAR (1,33,...) specifies pattern 1 (plus sign) with line separation of six units and solid lines.

Table B-1: Hatch Patterns

Line Separation: 6 Units

Solid Lines	Dashes	Long Dashes	Long/Short Dashes
1 plus sign	13 plus sign	25 plus sign	37 plus sign
2 slash	14 slash	26 slash	38 slash
3 horiz. line	15 horiz. line	27 horiz. line	39 horiz. line
4 backslash	16 backslash	28 backslash	40 backslash
5 vert. line	17 vert. line	29 vert. line	41 vert. line
6 X	18 X	30 X	42 X

Line Separation: 11 Units

Solid Lines	Dashes	Long Dashes	Long/Short Dashes
7 plus sign	19 plus sign	31 plus sign	43 plus sign
8 slash	20 slash	32 slash	44 slash
9 horiz. line	21 horiz. line	33 horiz. line	45 horiz. line
10 backslash	22 backslash	34 backslash	46 backslash
11 vert. line	23 vert. line	35 vert. line	47 vert. line
12 X	24 X	36 X	48 X

Hatch pattern one is the same as pattern seven, and so forth. The difference is that that the lower-numbered patterns have hatch lines that are separated by six units and the higher-numbered six patterns have lines separated by 11 units. The entire hatch pattern set repeats with codes 49 through 96. Some of the patterns are shown in Figures B-1 and B-2.

HEWLETT-PACKARD PLOTTERS

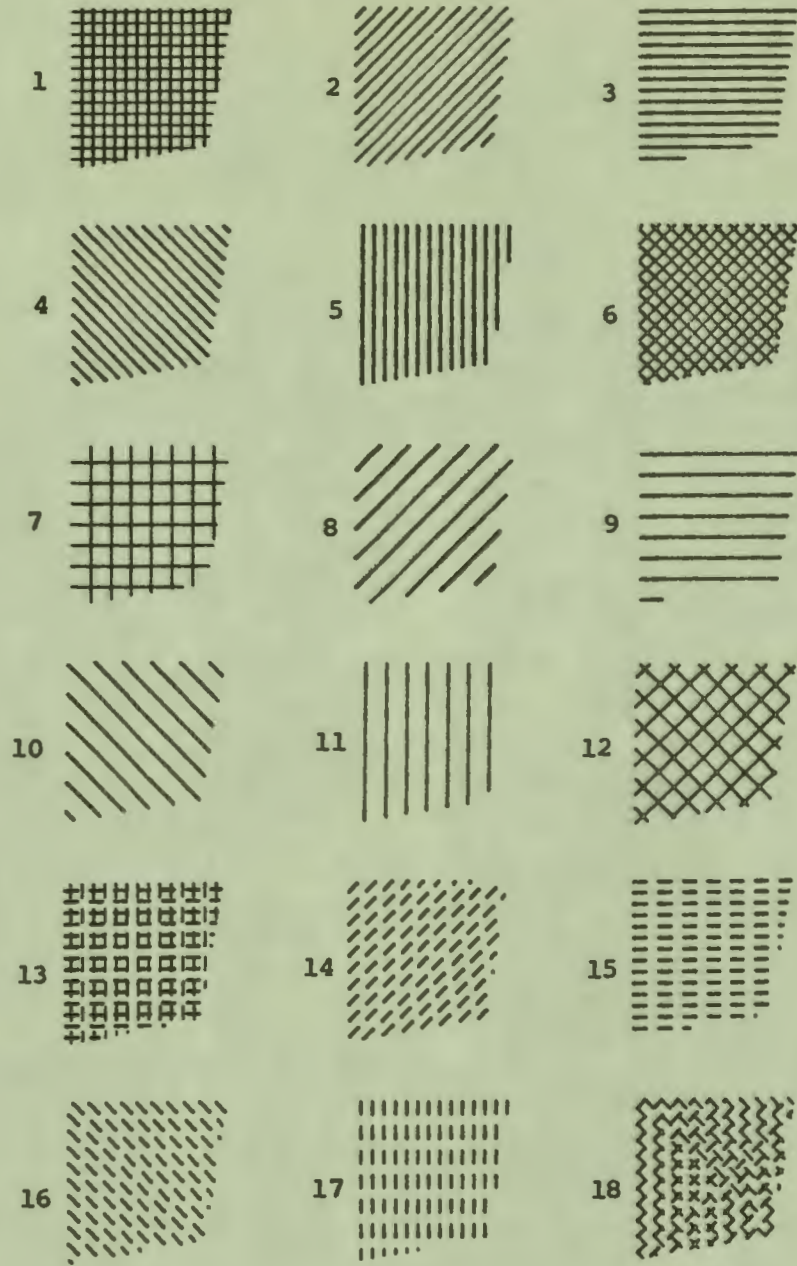


Figure B-1: Hatch Patterns 1 through 18

HEWLETT-PACKARD PLOTTERS

B.1.12 SET_FONT

All characters of Font 0 (DEC Multinational) are supported. The error character, for control characters and so forth, is the question mark.

B.1.13 SET_FONT_SIZE

SET_FONT_SIZE is ignored except that the font extent specifies either DEC Multinational or the marker alphabet as the current font. There are three cases:

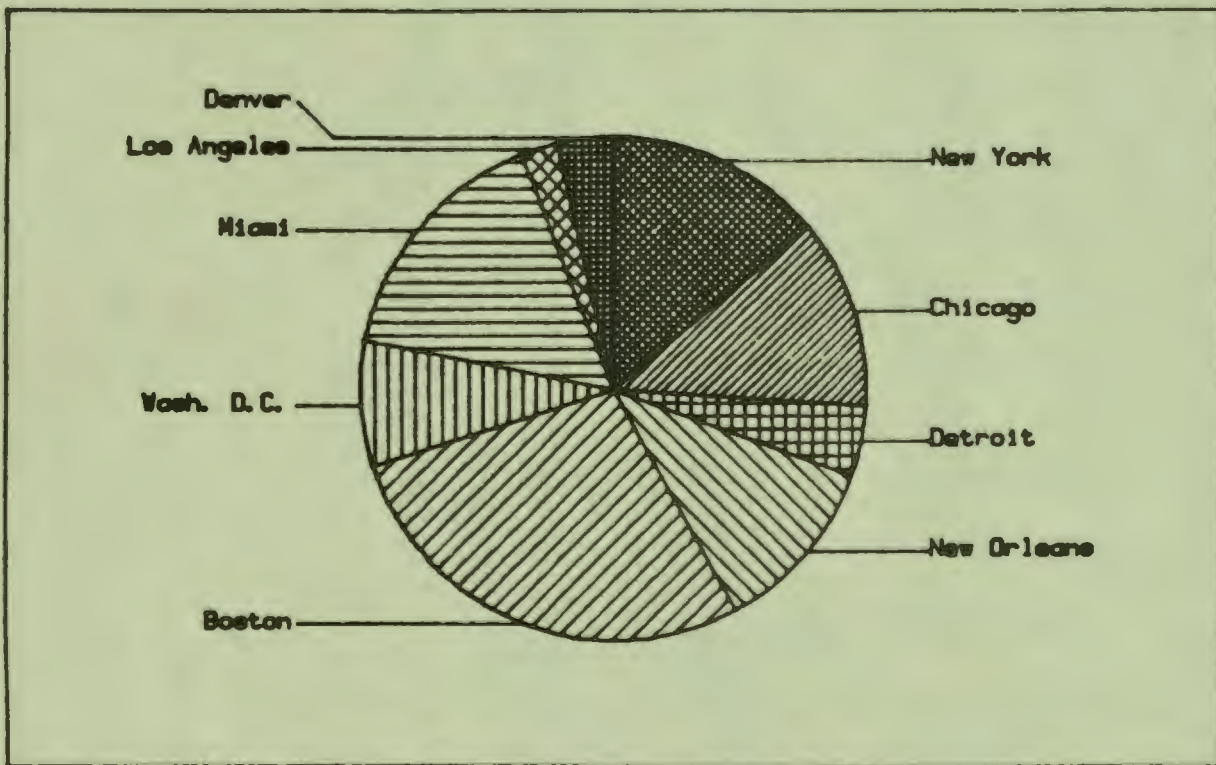


Figure B-2: Pie Chart (Drawn with HP7470 Plotter)

- An extent less than or equal to 60 specifies the marker alphabet.
- An extent of 128 or 256 specifies DEC Multinational with no character index mapping.

HEWLETT-PACKARD PLOTTERS

- Any other extent specifies DEC Multinational without C0 and C1.

B.1.14 Plotter Errors

When something does wrong with the plotter, CGL simply returns error 906: "Error on view surface device." This list documents the behavior of the plotter in unusual conditions so that you can find and correct the error.

NOTE

This list assumes that the plotter and printer are in series via the "Eavesdrop" cable provided by Hewlett-Packard.

- If either the plotter or printer is offline (not plugged in or either's power is off) when you start a plotting application, INITIALIZE_VIEW_SURFACE causes an error.
- If the printer port is already attached, INITIALIZE_VIEW_SURFACE causes an error.
- If you turn the plotter (or printer) off while the plotter is active, data will be lost.
- To resume using the plotter or printer after the plotter has been turned off, you must turn both devices off and on.
- If your application does not call TERMINATE_VIEW_SURFACE before exiting, CGL implicitly terminates the plotter.

B.1.15 HP-GL Features Not Accessible from CGL

- Automatic generation of tick marks on the axis of a graph.
- Direct control over the velocity of the pen. However for certain primary colors, pen velocity is set to 26 cm/sec.
- The various Report commands.
- Manual setting of P1 and P2. In other words, you cannot manually adjust the plotting area from the front panel of the plotter.

HEWLETT-PACKARD PLOTTERS

- Digitize Point mode.
- Rotating of the X and Y axes on the HP7475A.
- Arcs with other than one segment per 10 degrees of arc.
- Hatching with arbitrary line separation. Separation is either an eighth or a sixteenth of an inch, approximately.

APPENDIX C

INCLUDE FILES

The following include files can be found (if present) in LB:[1,5].

C.1 BASIC-PLUS-2

The following include file, CGL.B2S, is provided with the CORE Graphics Library.

```
!
! Professional 300 CORE Graphics Library V2.0
!
! declarations for Tool Kit BASIC-PLUS-2
!
! 01-Mar-1984
!
! This include file is provided for
! instructional purposes only. It
! is not supported software.
!
! DECLARE INTEGER CONSTANT
!
! Instruction names
!
! MOVE_ABS_2 = 1%, &
! MOVE_REL_2 = 2%, &
! INQUIRE_CURRENT_POSITION_2 = 3%, &
! LINE_ABS_2 = 4%, &
! LINE_REL_2 = 5%, &
! POLYLINE_ABS_2 = 6%, &
! POLYLINE_REL_2 = 7%, &
! POLYGON_ABS_2 = 8%, &
! POLYGON_REL_2 = 9%, &
! RECTANGLE_ABS_2 = 10%, &
```


CGL.B2S

RECTANGLE_REL_2	=	11%,	&
SET_LINESTYLE	=	12%,	&
INQUIRE_LINESTYLE	=	13%,	&
SET_LINEWIDTH	=	14%,	&
INQUIRE_LINEWIDTH	=	15%,	&
TEXT	=	16%,	&
INQUIRE_TEXT_EXTENT_2	=	17%,	&
SET_FONT	=	18%,	&
INQUIRE_FONT	=	19%,	&
SET_CHARSIZE	=	20%,	&
INQUIRE_CHARSIZE	=	21%,	&
SET_CHARPATH	=	22%,	&
INQUIRE_CHARPATH	=	23%,	&
SET_CHARSPACE	=	24%,	&
INQUIRE_CHARSPACE	=	25%,	&
SET_CHARJUST	=	26%,	&
INQUIRE_CHARJUST	=	27%,	&
SET_CHARITALIC	=	28%,	&
INQUIRE_CHARITALIC	=	29%,	&
SET_FONT_SIZE	=	30%,	&
INQUIRE_FONT_SIZE	=	31%,	&
LOAD_CHARACTER	=	32%,	&
MARKER_ABS_2	=	33%,	&
MARKER_REL_2	=	34%,	&
POLYMARKER_ABS_2	=	35%,	&
POLYMARKER_REL_2	=	36%,	&
SET_MARKER_SYMBOL	=	37%,	&
INQUIRE_MARKER_SYMBOL	=	38%,	&
ARC_ABS_2	=	39%,	&
ARC_REL_2	=	40%,	&
CURVE_ABS_2	=	41%,	&
CURVE_REL_2	=	42%,	&
SET_WRITING_INDEX	=	60%,	&
INQUIRE_WRITING_INDEX	=	61%,	&
SET_BACKGROUND_INDEX	=	62%,	&
INQUIRE_BACKGROUND_INDEX	=	63%,	&
SET_COLOR_MAP	=	64%,	&
INQUIRE_COLOR_MAP	=	65%,	&
SET_COLOR_MAP_ENTRY	=	66%,	&
INQUIRE_COLOR_MAP_ENTRY	=	67%,	&
SET_WRITING_PLANES	=	68%,	&
INQUIRE_WRITING_PLANES	=	69%,	&
SET_WRITING_MODE	=	70%,	&
INQUIRE_WRITING_MODE	=	71%,	&
SET_GLOBAL_ATTRIBUTES	=	72%,	&
INQUIRE_GLOBAL_ATTRIBUTES	=	73%,	&
SET_FILL_MODE	=	74%,	&
INQUIRE_FILL_MODE	=	75%,	&
SET_FILL_ENTITY	=	76%,	&
INQUIRE_FILL_ENTITY	=	77%,	&
SET_FILL_CHAR	=	78%,	&

CGL.B2S

INQUIRE_FILL_CHAR	=	79%, &
SET_WINDOW	=	80%, &
INQUIRE_WINDOW	=	81%, &
SET_VIEWPORT_2	=	82%, &
INQUIRE_VIEWPORT_2	=	83%, &
SET_WINDOW_CLIPPING	=	84%, &
INQUIRE_WINDOW_CLIPPING	=	85%, &
SET_ORIGIN	=	86%, &
INQUIRE_ORIGIN	=	87%, &
ERASE_VIEWPORT	=	88%, &
SCROLL	=	89%, &
INITIALIZE_CORE	=	90%, &
TERMINATE_CORE	=	91%, &
NEW_FRAME	=	92%, &
REPORT_MOST_RECENT_ERROR	=	93%, &
PRINT_SCREEN	=	94%, &
CGL_WAIT	=	95%, &
BEGIN_BATCH	=	96%, &
END_BATCH	=	97%, &
SET_LINEWIDTH_ORIENTATION	=	98%, &
INQUIRE_LINEWIDTH_ORIENTATION	=	99%, &
SET_CURSOR	=	100%, &
INQUIRE_CURSOR	=	101%, &
SCROLL_VIEWPORT	=	102%, &
INITIALIZE_VIEW_SURFACE	=	103%, &
TERMINATE_VIEW_SURFACE	=	104%, &
SELECT_VIEW_SURFACE	=	105%, &
DESELECT_VIEW_SURFACE	=	106%, &
SET_NDC_SPACE_2	=	107%, &
INQUIRE_NDC_SPACE_2	=	108%, &
BEGIN_DEFINE_CHARACTER	=	109%, &
END_DEFINE_CHARACTER	=	110%, &
PLAYBACK_FILE	=	111%, &
LOAD_FONT	=	112%, &
!		&
! Fill modes		&
!		&
OFF	=	0%, &
VERT_LINE	=	1%, &
HORIZ_LINE	=	2%, &
POINT	=	3%, &
POLYGON	=	4%, &
!		&
! Character justification modes		&
!		&
LEFT_JUST	=	1%, &
CENTER_JUST	=	2%, &
RIGHT_JUST	=	3%, &
TOP_JUST	=	1%, &
BOTTOM_JUST	=	3%, &
!		&

CGL.B2S

```

! Standard line styles
!
SOLID = 1%, &
DASHED = 2%, &
DOT_DASHED = 3%, &
DOTTED = 4%, &
DOT_DOT_DASHED = 5%, &
DOTTED_WIDE_SPACING = 6%, &
DASHED_SHORT_LINES = 7%, &
DASHED_LONG_LINES = 8%, &
DOT_DASHED_SHORT_LINES = 9%, &
!
! Writing modes
!
TRANSPARENT = 0%, &
TRANSPARENT_NEGATE = 1%, &
COMPLEMENT = 2%, &
COMPLEMENT_NEGATE = 3%, &
OVERLAY = 4%, &
OVERLAY_NEGATE = 5%, &
REPLACE = 6%, &
REPLACE_NEGATE = 7%, &
ERASE = 8%, &
ERASE_NEGATE = 9%, &

```

C.2 DIBOL

The following include file, CGL.DBL, is provided with Tool Kit DIBOL.

```

;
; Professional 300 CORE Graphics Library V2.0
; definitions for Tool Kit DIBOL
;
; 01-Mar-1984
;
; This include file is provided for instructional
; purposes only. It is not supported software.
;
;
RECORD
GMA2 ,D2 ,01 ; Move Absolute 2
GMR2 ,D2 ,02 ; Move Relative 2
GICP2 ,D2 ,03 ; Inquire Current Position 2
GLA2 ,D2 ,04 ; Line absolute 2
GLR2 ,D2 ,05 ; Line relative 2
GPLA2 ,D2 ,06 ; Polyline absolute 2
GPLR2 ,D2 ,07 ; Polyline relative 2

```

CGL.DBL

GPGA2	,D2	,08	; Polygon absolute 2
GPGR2	,D2	,09	; Polygon relative 2
GRA2	,D2	,10	; Rectangle absolute 2
GRR2	,D2	,11	; Rectangle relative 2
GSLS	,D2	,12	; Set line style
GILS	,D2	,13	; Inquire line style
GSLW	,D2	,14	; Set line width
GILW	,D2	,15	; Inquire line width
GT	,D2	,16	; Text
GITE2	,D2	,17	; Inquire text extent 2
GSF	,D2	,18	; Set font
GIF	,D2	,19	; Inquire font
GSCS	,D2	,20	; Set character size
GICS	,D2	,21	; Inquire character size
GSCP	,D2	,22	; Set character path
GICP	,D2	,23	; Inquire character path
GSCSP	,D2	,24	; Set character space
GICSP	,D2	,25	; Inquire character space
GSCJ	,D2	,26	; Set character justification
GICJ	,D2	,27	; Inquire character justification
GSCI	,D2	,28	; Set character italics
GICI	,D2	,29	; Inquire character italics
GSFS	,D2	,30	; Set font size
GIFS	,D2	,31	; Inquire font size
GLC	,D2	,32	; Load character
GMKA2	,D2	,33	; Marker absolute 2
GMKR2	,D2	,34	; Marker relative 2
GPMA2	,D2	,35	; Polymarker absolute 2
GPMR2	,D2	,36	; Polymarker relative 2
GSMKS	,D2	,37	; Set marker symbol
GIMKS	,D2	,38	; Inquire marker symbol
GAA2	,D2	,39	; Arc absolute 2
GAR2	,D2	,40	; Arc relative 2
GCA2	,D2	,41	; Curve absolute 2
GCR2	,D2	,42	; Curve relative 2
GSWI	,D2	,60	; Set writing index
GIWI	,D2	,61	; Inquire writing index
GSBI	,D2	,62	; Set background index
GIBI	,D2	,63	; Inquire background index
GSCM	,D2	,64	; Set color map
GICM	,D2	,65	; Inquire color map
GSCME	,D2	,66	; Set color map entry
GICME	,D2	,67	; Inquire color map entry
GSWP	,D2	,68	; Set writing planes
GIWP	,D2	,69	; Inquire writing planes
GSWM	,D2	,70	; Set writing mode
GIWM	,D2	,71	; Inquire writing mode
GSGA	,D2	,72	; Set global attributes
GIGA	,D2	,73	; Inquire global attributes
GSFM	,D2	,74	; Set fill mode
GIFM	,D2	,75	; Inquire fill mode

CGL.DBL

GSFE	,D2	,76	; Set fill entity
GIFE	,D2	,77	; Inquire fill entity
GSFC	,D2	,78	; Set fill character
GIFC	,D2	,79	; Inquire fill character
GSW	,D2	,80	; Set window
GIW	,D2	,81	; Inquire window
GSV2	,D2	,82	; Set viewport 2
GIV2	,D2	,83	; Inquire viewport 2
GSWC	,D2	,84	; Set window clipping
GIWC	,D2	,85	; Inquire window clipping
GSO	,D2	,86	; Set origin
GIO	,D2	,87	; Inquire origin
GEV	,D2	,88	; Erase viewport
GS	,D2	,89	; Scroll
GIC	,D2	,90	; Initialize cgl
GTC	,D2	,91	; Terminate cgl
GNF	,D2	,92	; New frame
GRMRE	,D2	,93	; Report most recent error
GPS	,D2	,94	; Print screen
GCW	,D2	,95	; Cgl wait
GBB	,D2	,96	; Begin Batch
GEB	,D2	,97	; End Batch
GSLO	,D2	,98	; Set Linewidth Orientation
GILO	,D2	,99	; Inquire Linewith Orientation
GSCU	,D2	,100	; Set CURsor
GICU	,D2	,101	; Inquire CURsor
GSV	,D2	,102	; Scroll Viewport
GIVS	,D2	,103	; Initialize View Surface
GTVS	,D2	,104	; Terminate View Surface
GSVS	,D2	,105	; Select View Surface
GDVS	,D2	,106	; Deselect View Surface
GSNS2	,D2	,107	; Set NDC Space 2
GINS2	,D2	,108	; Inquire NDC Space 2
GBDC	,D2	,109	; Begin Define Character
GEDC	,D2	,110	; End Define Character
GPF	,D2	,111	; Playback File
GLF	,D2	,112	; Load Font

C.3 FORTRAN-77

The following include file, CGL.FTN, is provided with the CORE Graphics Library.

C
C
C
C

Professional 300 CORE Graphics Library V2.0
definitions for Tool Kit FORTRAN-77

01-Mar-1984

C
C
C
C
C

This include file is provided for instructional purposes only. It is not supported software.

```

IMPLICIT INTEGER G
PARAMETER (GMA2 = 01) ! Move Absolute 2
PARAMETER (GMR2 = 02) ! Move Relative 2
PARAMETER (GICP2 = 03) ! Inquire Current Position 2
PARAMETER (GLA2 = 04) ! Line Absolute 2
PARAMETER (GLR2 = 05) ! Line Relative 2
PARAMETER (GPLA2 = 06) ! PolyLine Absolute 2
PARAMETER (GPLR2 = 07) ! PolyLine Relative 2
PARAMETER (GPGA2 = 08) ! PolyGon Absolute 2
PARAMETER (GPGR2 = 09) ! PolyGon Relative 2
PARAMETER (GRA2 = 10) ! Rectangle Absolute 2
PARAMETER (GRR2 = 11) ! Rectangle Relative 2
PARAMETER (GSLs = 12) ! Set Line Style
PARAMETER (GILs = 13) ! Inquire Line Style
PARAMETER (GSLW = 14) ! Set Line Width
PARAMETER (GILW = 15) ! Inquire Line Width
PARAMETER (GT = 16) ! Text
PARAMETER (GITE2 = 17) ! Inquire Text Extent 2
PARAMETER (GSF = 18) ! Set Font
PARAMETER (GIF = 19) ! Inquire Font
PARAMETER (GSCS = 20) ! Set Character Size
PARAMETER (GICS = 21) ! Inquire Character Size
PARAMETER (GSCP = 22) ! Set Character Path
PARAMETER (GICP = 23) ! Inquire Character Path
PARAMETER (GSCSP = 24) ! Set Character SPace
PARAMETER (GICSP = 25) ! Inquire Character SPace
PARAMETER (GSCJ = 26) ! Set Character Justification
PARAMETER (GICJ = 27) ! Inquire Character Justification
PARAMETER (GSCI = 28) ! Set Character Italics
PARAMETER (GICI = 29) ! Inquire Character Italics
PARAMETER (GSFS = 30) ! Set Font Size
PARAMETER (GIFS = 31) ! Inquire Font Size
PARAMETER (GLC = 32) ! Load Character
PARAMETER (GMKA2 = 33) ! MarKer Absolute 2
PARAMETER (GMKR2 = 34) ! MarKer Relative 2
PARAMETER (GPMA2 = 35) ! PolyMarker Absolute 2
PARAMETER (GPMR2 = 36) ! PolyMarker Relative 2
PARAMETER (GSMKS = 37) ! Set MarKer Symbol
PARAMETER (GIMKS = 38) ! Inquire MarKer Symbol
PARAMETER (GAA2 = 39) ! Arc Absolute 2
PARAMETER (GAR2 = 40) ! Arc Relative 2
PARAMETER (GCA2 = 41) ! Curve Absolute 2
PARAMETER (GCR2 = 42) ! Curve Relative 2
PARAMETER (GSWI = 60) ! Set Writing Index
PARAMETER (GIWI = 61) ! Inquire Writing Index
PARAMETER (GSBI = 62) ! Set Background Index

```


CGL.FTN

PARAMETER (GIBI = 63) ! Inquire Background Index
 PARAMETER (GSCM = 64) ! Set Color Map
 PARAMETER (GICM = 65) ! Inquire Color Map
 PARAMETER (GSCME = 66) ! Set Color Map Entry
 PARAMETER (GICME = 67) ! Inquire Color Map Entry
 PARAMETER (GSWP = 68) ! Set Writing Planes
 PARAMETER (GIWP = 69) ! Inquire Writing Planes
 PARAMETER (GSWM = 70) ! Set Writing Mode
 PARAMETER (GIWM = 71) ! Inquire Writing Mode
 PARAMETER (GSGA = 72) ! Set Global Attributes
 PARAMETER (GIGA = 73) ! Inquire Global Attributes
 PARAMETER (GSFM = 74) ! Set Fill Mode
 PARAMETER (GIFM = 75) ! Inquire Fill Mode
 PARAMETER (GSFE = 76) ! Set Fill Entity
 PARAMETER (GIFE = 77) ! Inquire Fill Entity
 PARAMETER (GSFC = 78) ! Set Fill Character
 PARAMETER (GIFC = 79) ! Inquire Fill Character
 PARAMETER (GSW = 80) ! Set Window
 PARAMETER (GIW = 81) ! Inquire Window
 PARAMETER (GSV2 = 82) ! Set Viewport 2
 PARAMETER (GIV2 = 83) ! Inquire Viewport 2
 PARAMETER (GSWC = 84) ! Set Window Clipping
 PARAMETER (GIWC = 85) ! Inquire Window Clipping
 PARAMETER (GSO = 86) ! Set Origin
 PARAMETER (GIO = 87) ! Inquire Origin
 PARAMETER (GEV = 88) ! Erase Viewport
 PARAMETER (GS = 89) ! Scroll
 PARAMETER (GIC = 90) ! Initialize Cgl
 PARAMETER (GTC = 91) ! Terminate Cgl
 PARAMETER (GNF = 92) ! New Frame
 PARAMETER (GRMRE = 93) ! Report Most Recent Error
 PARAMETER (GPS = 94) ! Print Screen
 PARAMETER (GCW = 95) ! Cgl Wait
 PARAMETER (GBB = 96) ! Begin Batch
 PARAMETER (GEB = 97) ! End Batch
 PARAMETER (GSLO = 98) ! Set Linewidth Orientation
 PARAMETER (GILO = 99) ! Inquire Linewith Orientation
 PARAMETER (GSCU = 100) ! Set Cursor
 PARAMETER (GICU = 101) ! Inquire CURsor
 PARAMETER (GSV = 102) ! Scroll Viewport
 PARAMETER (GIVS = 103) ! Initialize View Surface
 PARAMETER (GTVS = 104) ! Terminate View Surface
 PARAMETER (GSVS = 105) ! Select View Surface
 PARAMETER (GDVS = 106) ! Deselect View Surface
 PARAMETER (GSNS2 = 107) ! Set NDC Space 2
 PARAMETER (GINS2 = 108) ! Inquire NDC Space 2
 PARAMETER (GBDC = 109) ! Begin Define Character
 PARAMETER (GEDC = 110) ! End Define Character
 PARAMETER (GPF = 111) ! Playback File
 PARAMETER (GLF = 112) ! Load Font

PASCAL

C.4 PASCAL

The following include file, CGLDEFS.PAS, is provided with Tool Kit PASCAL.

(* PASCAL declarations for the CORE Graphics Library V1.7 *)

CONST

(* clipping *)

off = 0;
on = 1;

(* curve mode *)

open_curve = 0;
closed_curve = 1;

(* definitions for V1.0 compatibility *)

opened = open_curve;
closed = closed_curve;

(* writing mode *)

transparent = 0;
transparent_negate = 1;
complement = 2;
complement_negate = 3;
overlay = 4;
overlay_negate = 5;
replace = 6;
replace_negate = 7;
erase = 8;
erase_negate = 9;

(* fill entity *)

fill_off = 0;
vert_line = 1;
horiz_line = 2;
point = 3;
polygon = 4;

(* justification *)

left_just = 1;
center_just = 2;

CGLDEFS.PAS

```
right_just = 3;  
top_just = 1;  
bottom_just = 3;
```

```
(* character path *)
```

```
char_mode = 0;  
string_mode = 1;
```

```
(* marker symbol *)
```

```
period = 1;  
plus_sign = 2;  
asterisk = 3;  
circle = 4;  
cross = 5;
```

```
(* window origin *)
```

```
bottom_left = 0;  
top_left = 1;  
top_right = 2;  
bottom_right = 3;
```

```
(* line style *)
```

```
solid = 1;  
dashed = 2;  
dot_dashed = 3;  
dotted = 4;  
dot_dot_dashed = 5;  
dotted_wide_spacing = 6;  
dashed_short_lines = 7;  
dashed_long_lines = 8;  
dot_dashed_short_lines = 9;
```

TYPE

```
clipping = off..on;  
curve_mode = open_curve..closed_curve;  
writing_mode = transparent..erase_negate;  
fill_mode = fill off..point;  
justification = left_just..right_just;  
char_path_mode = char_mode..string_mode;  
origin_code = bottom_left..bottom_right;  
byte = 0..255;  
planes = 0..7;  
RGB_value = 0..7;  
color_map_index = 0..7;  
font_desig = 0..3;  
font_extent = 32..126;  
reals = ARRAY [1..10] OF real; (* arbitrary size *)
```

CGLDEFS.PAS

```
string = PACKED ARRAY [1..80] OF char; (* arbitrary size *)
char_matrix = ARRAY [0..15] OF unsigned;
color_map = ARRAY [0..23] OF RGB_value;
color_map_entry = ARRAY [1..3] OF RGB_value;
```

```
[EXTERNAL($PCGL)]
PROCEDURE move_abs_2(VAR x, y: [readonly] real;
                    $$$: integer := 1); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE move_rel_2(VAR dx, dy: [readonly] real;
                    $$$: integer := 2); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_current_position_2(VAR x, y: real;
                                     $$$: integer := 3); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE line_abs_2(VAR x, y: [readonly] real;
                    $$$: integer := 4); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE line_rel_2(VAR dx, dy: [readonly] real;
                    $$$: integer := 5); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE polyline_abs_2(VAR x, y: [readonly, unsafe] reals;
                        VAR n: [readonly] integer;
                        $$$: integer := 6); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE polyline_rel_2(VAR dx, dy: [readonly, unsafe] reals;
                        VAR n: [readonly] integer;
                        $$$: integer := 7); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE polygon_abs_2(VAR x, y: [readonly, unsafe] reals;
                       VAR n: [readonly] integer;
                       $$$: integer := 8); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE polygon_rel_2(VAR dx, dy: [readonly, unsafe] reals;
                       VAR n: [readonly] integer;
```


CGLDEFS.PAS

\$\$\$: integer := 9); SEQ11;

```
[EXTERNAL($PCGL)]
PROCEDURE rectangle_abs_2(VAR x, y: [readonly] real;
                          $$$: integer := 10); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE rectangle_rel_2(VAR dx, dy: [readonly] real;
                          $$$: integer := 11); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_linestyle(VAR style: [readonly] integer;
                       VAR pattern: [readonly] unsigned;
                       VAR mult: [readonly] integer;
                       $$$: integer := 12); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_linestyle(VAR style: integer;
                           VAR pattern: unsigned;
                           VAR mult: integer;
                           $$$: integer := 13); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_linewidth(VAR dx, dy: [readonly] real;
                       $$$: integer := 14); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_linewidth(VAR dx, dy: real;
                           $$$: integer := 15); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE txt(VAR s: [readonly, unsafe] string;
             VAR len: [readonly] integer;
             $$$: integer := 16); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_text_extent_2(VAR len: [readonly] integer;
                               VAR dx, dy: real;
                               $$$: integer := 17); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_font(VAR n: [readonly] font_desig;
```

CGLDEFS.PAS

\$\$\$: integer := 18); SEQ11;

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_font(VAR n: font_desig;
                      $$$ : integer := 19); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_charsize(VAR width, height: [readonly] real;
                      $$$ : integer := 20); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_charsize(VAR width, height: real;
                          $$$ : integer := 21); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_charpath(VAR a: [readonly] integer;
                      VAR m: [readonly] char_path_mode;
                      $$$ : integer := 22); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_charpath(VAR a: integer;
                          VAR m: char_path_mode;
                          $$$ : integer := 23); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_charspace(VAR dx, dy: [readonly] real;
                       $$$ : integer := 24); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_charspace(VAR dx, dy: real;
                            $$$ : integer := 25); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_charjust(VAR x_just, y_just: [readonly] justification;
                      $$$ : integer := 26); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_charjust(VAR x_just, y_just: justification;
                          $$$ : integer := 27); SEQ11;
```

```
[EXTERNAL($PCGL)]
```


CGLDEFS.PAS

```
PROCEDURE set_charitalic(VAR angle: [readonly] integer;  
                        $$$: integer := 28); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_charitalic(VAR angle: integer;  
                             $$$: integer := 29); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_font_size(VAR extent: [readonly] font_extent;  
                        VAR x_size, y_size: [readonly] integer;  
                        $$$: Integer := 30); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_font_size(VAR extent: font_extent;  
                             VAR x_size, y_size: integer;  
                             $$$: Integer := 31); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE load_character(VAR ch: [readonly] font_extent;  
                         VAR matrix: [readonly] char_matrix;  
                         $$$: integer := 32); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE marker_abs_2(VAR x, y: [readonly] real;  
                       $$$: integer := 33); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE marker_rel_2(VAR dx, dy: [readonly] real;  
                       $$$: integer := 34); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE polymarker_abs_2(VAR x, y: [readonly, unsafe] reals;  
                            VAR n: [readonly] integer;  
                            $$$: integer := 35); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE polymarker_rel_2(VAR dx, dy: [readonly, unsafe] reals;  
                            VAR n: [readonly] integer;  
                            $$$: integer := 36); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_marker_symbol(VAR n: [readonly] integer;
```

CGLDEFS.PAS

```
VAR c: [readonly] char;  
$$$: integer := 37); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_marker_symbol(VAR n: integer;  
                                VAR c: char;  
                                $$$: integer := 38); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE arc_abs_2(VAR x, y: [readonly] real;  
                   VAR a: [readonly] integer;  
                   $$$: integer := 39); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE arc_rel_2(VAR dx, dy: [readonly] real;  
                   VAR a: [readonly] integer;  
                   $$$: integer := 40); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE curve_abs_2(VAR x, y: [readonly, unsafe] reals;  
                     VAR n: [readonly] integer;  
                     VAR c: [readonly] curve_mode;  
                     $$$: integer := 41); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE curve_rel_2(VAR dx, dy: [readonly, unsafe] reals;  
                     VAR n: [readonly] integer;  
                     VAR c: [readonly] curve_mode;  
                     $$$: integer := 42); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_writing_index(VAR n: [readonly] color_map_index;  
                            $$$: integer := 60); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_writing_index(VAR n: color_map_index;  
                                $$$: integer := 61); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_background_index(VAR n: [readonly] color_map_index;  
                               $$$: integer := 62); SEQ11;
```


CGLDEFS.PAS

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_background_index(VAR n: color_map_index;  
                                   $$$: integer := 63); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_color_map(VAR c: [readonly] color_map;  
                        $$$: integer := 64); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_color_map(VAR c: color_map;  
                             $$$: integer := 65); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_color_map_entry(VAR entry: [readonly] color_map_index;  
                              VAR color: [readonly] color_map_entry;  
                              $$$: integer := 66); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_color_map_entry(VAR entry: color_map_index;  
                                  VAR c: color_map_entry;  
                                  $$$: integer := 67); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_writing_planes(VAR n: [readonly] planes;  
                             $$$: integer := 68); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_writing_planes(VAR n: planes;  
                                 $$$: integer := 69); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_writing_mode(VAR n: [readonly] writing_mode;  
                           $$$: integer := 70); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_writing_mode(VAR n: writing_mode;  
                               $$$: integer := 71); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_fill_mode(VAR n: [readonly] fill_mode;  
                       $$$: integer := 74); SEQ11;
```

CGLDEFS.PAS

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_fill_mode(VAR n: fill_mode;  
                             $$$: integer := 75); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_fill_entity(VAR x, y: [readonly] real;  
                           $$$: integer := 76); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_fill_entity(VAR x, y: real;  
                               $$$: integer := 77); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_fill_char(VAR font: [readonly] integer;  
                        VAR ch: [readonly] byte;  
                        VAR width_mult: [readonly] integer;  
                        VAR height_mult: [readonly] integer;  
                        $$$: integer := 78); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_fill_char(VAR font: integer;  
                            VAR ch: char;  
                            VAR width_mult: integer;  
                            VAR height_mult: integer;  
                            $$$: integer := 79); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_window(VAR xmin, xmax, ymin, ymax: [readonly] real;  
                     $$$: integer := 80); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_window(VAR xmin, xmax, ymin, ymax: real;  
                         $$$: integer := 81); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_viewport_2(VAR xmin, xmax,  
                         ymin, ymax: [readonly] real;  
                         $$$: integer := 82); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_viewport_2(VAR xmin, xmax, ymin, ymax: real;  
                              $$$: integer := 83); SEQ11;
```


CGLDEFS.PAS

```
[EXTERNAL($PCGL)]  
PROCEDURE set_window_clipping(VAR n: [readonly] clipping;  
                               $$$: integer := 84); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_window_clipping(VAR n: clipping;  
                                   $$$: integer := 85); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE set_origin(VAR n: [readonly] origin_code;  
                     $$$: integer := 86); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE inquire_origin(VAR n: origin_code;  
                          $$$: integer := 87); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE erase_viewport($$$: integer := 88); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE scroll(VAR dx, dy: [readonly] real;  
                $$$: integer := 89); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE initialize_core($$$: integer := 90); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE terminate_core($$$: integer := 91); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE new_frame($$$: integer := 92); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE report_most_recent_error(VAR f, e: integer;  
                                    $$$: integer := 93); SEQ11;
```

```
[EXTERNAL($PCGL)]  
PROCEDURE print_screen(VAR xmin, xmax,  
                       ymin, ymax, xoff, yoff: [readonly] real;  
                       $$$: integer := 94); SEQ11;
```

CGLDEFS.PAS

```
[EXTERNAL($PCGL)]
PROCEDURE cgl_wait(VAR s: [readonly] real;
                  $$$: integer := 95); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE begin_batch($$$: integer := 96); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE end_batch($$$: integer := 97); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_linewidth_orientation(VAR dx, dy: [readonly] real;
                                    $$$: integer := 98); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_linewidth_orientation(VAR dx, dy: real;
                                       $$$: integer := 99); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_cursor(VAR font: [readonly] font_desig;
                    VAR ch: [readonly] font_extent;
                    VAR width, height: [readonly] integer;
                    VAR dx, dy: [readonly] real;
                    $$$: integer := 100); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_cursor(VAR font: font_desig;
                        VAR ch: font_extent;
                        VAR width, height: integer;
                        VAR dx, dy: real;
                        $$$: integer := 101); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE scroll_viewport(VAR dx, dy: [readonly] real;
                         $$$: integer := 102); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE initialize_view_surface(VAR name: [readonly,unsafe] string
                                  VAR length: [readonly] integer;
                                  $$$: integer := 103); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE terminate_view_surface(VAR name: [readonly,unsafe] string;
                                  VAR length: [readonly] integer;
```


CGLDEFS.PAS

\$\$\$: integer := 104); SEQ11;

```
[EXTERNAL($PCGL)]
PROCEDURE select_view_surface(VAR name: [readonly, unsafe] string;
                              VAR length: [readonly] integer;
                              $$$ : integer := 105); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE deselect_view_surface(VAR name: [readonly, unsafe] string;
                                VAR length: [readonly] integer;
                                $$$ : integer := 106); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE set_ndc_space_2(VAR width, height: [readonly] real;
                          $$$ : integer := 107); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE inquire_ndc_space_2(VAR width, height : real;
                              $$$ : integer := 108); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE begin_define_character(VAR code: [readonly] integer;
                                  $$$ : integer := 109); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE end_define_character($$$ : integer := 110); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE playback_file(VAR s: [readonly, unsafe] string;
                        VAR len: [readonly] integer;
                        $$$ : integer := 111); SEQ11;
```

```
[EXTERNAL($PCGL)]
PROCEDURE load_font(VAR s: [readonly, unsafe] string;
                    VAR len: [readonly] integer;
                    $$$ : integer := 112); SEQ11;
```

APPENDIX D
EXAMPLE PROGRAMS

The following example programs are provided for educational purposes only. They are not supported software and are not included with the Tool Kit.

D.1 COLORMAP.PAS - COLOR MAP EDITOR

```
program color_map_editor;

{ This program is provided for instructional purposes only. }
{ It demonstrates some aspects of the following software tools: }

{ Tool Kit PASCAL V1.1 }
{ CORE Graphics Library V1.7 }
{ P/OS User Interface Library }

{ This application graphically demonstrates the function }
{ of the color map provided with the Professional 300 Series }
{ Extended Bitmap Option. Please refer to the associated }
{ help frame for more information. }

#include 'lb:[1,5]cgldefs.pas/nolist'

type
    status_block = array [0..1] of integer;

var
    x, y           : real;
    ttch, status  : status_block;
    current_RGB   : 1..3;
    current_map   : color_map;
    current_index : color_map_index;
    current_entry : color_map_entry;
    done         : boolean;
    RGB_char     : char;
```


COLORMAP.PAS - COLOR MAP EDITOR

```

const
  black = color_map_entry (0,0,0);
  white = color_map_entry (7,7,6);
  RGB_string = 'RGB';

procedure getkey (var s : status_block); seqll;

procedure help (var s : status_block); seqll;

procedure draw_color_map (var current_map : [readonly] color_map);
var
  i, j          : integer;
  x, y          : real;
  current_index : color_map_index;
  current_entry : color_map_entry;

begin
  new frame;
  begin_batch;
  set_writing_planes (7);
  set_color_map (current_map);
  set_background_index (0);
  set_writing_index(7);
  set_writing_mode (overlay);
  set_fill_mode (polygon);
  for i := 0 to 7 do
    begin
      { Draw the color bars }
      set_writing_index (i); y := i;
      move_abs_2 (0.0, y);
      rectangle_rel_2 (3.0, 1.0);
    end { for };
  set_fill_mode (fill_off);
  set_writing_index (7);
  for i := 0 to 8 do
    begin
      { Outline the color bars }
      y := i; move_abs_2 (0.0, y); line_abs_2 (3.0, y);
    end { for };
  for i := 0 to 3 do
    begin
      x := i; move_abs_2 (x, 0.0); line_abs_2 (x, 8.0);
    end { for };
  set_charsize (0.25, 0.5);
  set_charjust (center_just, center_just);
  for i := 1 to 3 do
    begin
      { Label the map }
      move_abs_2 (i - 0.5, -0.5); txt (RGB_string[i], 1);
    end { for };
  for i := 0 to 7 do
    begin
      move_abs_2 (-0.5, i + 0.5); txt (chr(i + 48), 1);
    end { for };

```

COLORMAP.PAS - COLOR MAP EDITOR

```

set_charspace (0.25,0.0);
move_abs_2 (1.5,8.5);
txt ('Press HELP for help.', 20);
set_writing_mode (complement);
for i := 0 to 7 do
    { Draw the RGB values }
    begin
        current_index := i;
        inquire_color_map_entry (current_index,current_entry);
        for j := 1 to 3 do
            begin
                move_abs_2 (j - 0.5, i + 0.5);
                txt (chr(current_entry[j] + 48), 1);
            end { for };
        end { for };
    end_batch;
end { draw_color_map };

begin
initialize_core; new_frame;
set_window (-1.0, 4.0, -1.0, 9.0);
set_viewport_2 (0.1875, 0.8125, 0.0, 1.0);
current_map := color_map
(0,0,0, 7,0,0, 0,7,0, 0,0,6, 7,7,0, 7,0,6, 0,7,6, 7,7,6);
{ black, red, green, blue, yellow, magenta, cyan, black }
draw_color_map (current_map);
current_index := 0; current_RGB := 1; done := false;
while not done do
    begin
        x := current_RGB - 0.5; y := current_index + 0.5;
        move_abs_2 (x,y); { position cursor on RGB number }
        getkey (ttch);
        case ttch[0] of
            1 : { Main keyboard key }
                if (ttch[1] - 48) in [0, 1, 2, 3, 4, 5, 6, 7]
                    then { new RGB value in ttch[1] }
                        begin
                            begin_batch;
                            inquire_color_map_entry (current_index, current_entry)
                            current_entry[current_RGB] := ttch[1] - 48;
                            set_color_map_entry (current_index, current_entry);
                            RGB_char := chr(current_entry[current_RGB] + 48);
                            { the old RGB number was drawn in complement mode }
                            { so to get rid of it, we draw it's negative image }
                            { using the same writing index as the color bar }
                            set_writing_index (current_index);
                            set_writing_mode (erase_negate); txt (RGB_char, 1);
                            { draw the new RGB number }
                            set_writing_mode (complement); txt (RGB_char, 1);
                            end_batch;
                        end { if };
            2 : { Function key }

```


COLORMAP.PAS - COLOR MAP EDITOR

```
case ttch[1] of
  7, 8, 9, 10 : { RESUME, CANCEL, MAIN SCREEN, EXIT }
    begin
      new_frame; terminate_core; done := true;
    end;
  15 : { HELP }
    begin { set up for text mode }
      inquire_color_map (current_map);
      new_frame;
      set_color_map_entry (0,black);
      set_color_map_entry (4,white);
      help (status);
      draw_color_map (current_map);
    end;
  27 : { Up arrow }
    current_index := (current_index - 1) mod 8;
  28 : { Left arrow }
    current_RGB := ((current_RGB + 1) mod 3) + 1;
  29 : { Down arrow }
    current_index := (current_index + 1) mod 8;
  30 : { Right arrow }
    current_RGB := (current_RGB mod 3) + 1;
  otherwise { ignore it };
end { case };
otherwise { ignore it };
end { case };
end { while };
end.
```

D.1.1 COLORMAP.HLP - HELP FRAME

Color Map Editor

What you see is a graphic representation of the Professional 300 series color map (EBO required). It contains the primary colors (red, green, and blue), the complementary colors (yellow, magenta, and cyan), black, and white. The background index is 0 and the writing index is 7.

You can manipulate the red, green, and blue values in each color map entry with the following keys:

Arrow keys move the cursor around the color map.

Numeric keys (range 0 to 7) set new RGB values.

EXIT and MAIN SCREEN return to the Main Menu.

Press RESUME to continue.

GEDIT.B2S - GRAPHICS SKETCHPAD

D.2 GEDIT.B2S - GRAPHICS SKETCHPAD

10

```
!
! Program GEDIT - Graphics Scratchpad Program
!
!       DEC ESD&P SCD
!       ZK01-2/E16
!       110 Spit Brook Road
!       Nashua, NH 03061
!
! Instructions:
!
! The status line displays the current
! function, home position, and mode.
!
!           *** Editing Keys ***
!
! +-----+-----+-----+      MOVE:  Select move mode
! | MOVE | WRIT | ERAS |          WRIT:  Select write mode
! +-----+-----+-----+      ERAS:  Select erase mode
! |  NH  | HOME |  CS  |          NH:    Set new home position
! +-----+-----+-----+      HOME:  Return to home position
! |      | ^   |      |          CS:    Clear the screen
! +-----+-----+-----+
! | <-  | V   | -> |          The arrow keys move the cursor.
! +-----+-----+-----+
!
!           *** Function Keys ***
!
! F17  F18  F19  F20          VECT:  Begin vector
! +-----+-----+-----+      RECT:  Begin rectangle
! | VECT | RECT | CIRC |          CIRC:  Begin circle
! +-----+-----+-----+
!
! To draw a vector, press <VECT>, move the cursor to the
! other end of the vector and press <DO>.
!
! To draw a rectangle, press <RECT>, move the cursor to
! the opposing corner of the rectangle and press <DO>.
!
! To draw a circle, press <CIRC>, move the cursor from the
! center of the circle to any point on the circumference
! and press <DO>.
!
!           *** Other Keys ***
!
! Exit:          Exit graphics sketchpad.
! Cancel:       Cancel function (vector, circle, rectangle)
! F11:         Enable/disable keyboard bell
!
! Environmental Definitions
```


GEDIT.B2S - GRAPHICS SKETCHPAD

```

!
%include "LB:[1,5]CGL.B2S" ! CGL symbols
!
declare real constant                                     &
    XINC          = .003, ! X movement index           &
    YINC          = -.002 ! Y movement index           &
!
declare integer constant                                 &
    CIRCUMF       = 360   ! Degrees in a circle       &
    , DO_KEY      = 29    ! Do key parameter val.    &
    , CANCEL_KEY  = 19    ! Cancel Key                                           &
    , FIND_KEY    = 1     ! Find Key                                           &
    , INSERT_KEY  = 2     ! Insert key                                          &
    , REMOVE_KEY  = 3     ! Remove key                                          &
!
declare real                                             &
    CX            ! Current X position                 &
    , CY          ! Current Y position                 &
    , PX          ! Stored X position                  &
    , PY          ! Stored Y position                  &
    , XX          ! Vector end-point X                 &
    , YY          ! Vector end-point Y                 &
    , HOMEX       ! Home X position                    &
    , HOMEY       ! Home Y position                    &
!
declare integer                                         &
    V             ! Locator Mode Flag                 &
    , V1          ! Locator Action Flag                 &
    , WK          ! Function key param.                 &
    , CURRENT_MODE ! Writing mode                                       &
    , LOUD        ! Bell on error flag                 &
!
declare string                                         &
    MODE_NAME     ! Current Writing mode                 &
    , OPTION_NAME ! Current Action mode                 &
!
100 !
!
START: ! Program Initialization
!
! Sets the default writing mode, home position and
! positions the cursor at home.
!
call CGL by ref ( INITIALIZE_CORE )
call CGL by ref ( NEW_FRAME )
call CGL by ref ( SET_WINDOW, 0.0, 1.0, 0.0, 0.625) ! square
CURRENT_MODE = OVERLAY ! Default writing mode
MODE_NAME = "Write" ! Mode name
OPTION_NAME = "Plot" ! Default option
LOUD = -1% ! Default to beep for err.
V = 0% ! Start non-vector mode

```

GEDIT.B2S - GRAPHICS SKETCHPAD

```

CX, CY, HOMEX, HOMEY = 0.5      ! Default home position
gosub CLEAR_SCREEN             ! Erase the screen
gosub STATUS_LINE              ! Display status
call CGL by ref ( SET_WRITING_MODE, CURRENT_MODE )
call CGL by ref ( SET_LINestyle, SOLID, 0%, 0%)
call CGL by ref ( MOVE_ABS_2, CX, CY ) ! cursor home
call CGL by ref ( LINE_ABS_2, CX, CY ) ! plot a point
!
!
110 !
    MAIN:
! Main plotting loop. This loop calls the movement
! routine and plots a point.
!
gosub COMMANDS                  ! get movement
call CGL by ref ( LINE_ABS_2, CX, CY ) ! plot a point
goto MAIN                       ! loop
!
!
120 !
    COMMANDS:
!
! The movement routine
!
dim TTCH%(1%)
call GETKEY by ref (TTCH%())
return unless TTCH%(0%) = 2% ! Ignore non-function keys
select TTCH%(1%)
    case 8 ! CANCEL
        V1 = -1% if V = 1%
    case 10 ! EXIT
        gosub ENDIT
    case 11 ! F11
        gosub NOISE
    case 15 ! HELP
        gosub HELP_IT
    case 16 ! DO
        V1 = 1% IF V = 1%
    case 17 ! F17
        gosub VECTOR_PLOT unless V = 1%
    case 18 ! F18
        gosub RECTANGLE unless V = 1%
    case 19 ! F19
        gosub CIRCLE unless V = 1%
    case 21 ! FIND
        gosub TRACE_MODE
    case 22 ! INSERT HERE
        gosub WRITE_MODE
    case 23 ! REMOVE
        gosub ERASE_MODE
    case 24 ! SELECT
        gosub SET_HOME unless V = 1%
    case 25 ! PREV SCREEN

```


GEDIT.B2S - GRAPHICS SKETCHPAD

```

                                gosub HOME_CURSOR unless V = 1%
case 26 ! NEXT SCREEN
                                gosub CLEAR_SCREEN unless V = 1%
case 27 ! up arrow
                                CY = CY + YINC
case 28 ! left arrow
                                CX = CX - XINC
case 29 ! down arrow
                                CY = CY - YINC
case 30 ! right arrow
                                CX = CX + XINC
                                case else
end select
return
!
240 !
    HOME_CURSOR:
    !
    ! Move current position to Home position
    !
    CX = HOMEX \ CY = HOMEY
    call CGL by ref ( MOVE_ABS_2, CX, CY )
    return
250 !
    CLEAR_SCREEN:
    !
    call CGL by ref ( NEW_FRAME )
    call CGL by ref ( SET_WRITING_MODE, OVERLAY)
    call CGL by ref ( MOVE_ABS_2, 0.0, 0.0)
    call CGL by ref ( RECTANGLE_ABS_2, 1.0, 0.625)
    call CGL by ref ( SET_WRITING_MODE, CURRENT_MODE)
    gosub STATUS_LINE ! Redisplay status
    return
    !
260 !
    ERASE_MODE:
    !
    ! Set writing mode to erase mode and update status line.
    ! If in Locator mode, do not change actual writing mode.
    !
    MODE_NAME = "Erase"
    CURRENT_MODE = ERASE ! Mode switch
    gosub STATUS_LINE ! Display Status Line
    return if V ! Locator mode return
    call CGL by ref ( SET_WRITING_MODE, CURRENT_MODE)
    return
270 !
    WRITE_MODE:
    !
    ! Set mode to replace mode and update status line. In
    ! Locator mode, don't update real writing mode.

```

GEDIT.B2S - GRAPHICS SKETCHPAD

```

!
MODE_NAME = "Write"
CURRENT_MODE = OVERLAY           ! Mode switch
gosub STATUS_LINE                ! Display Status Line
return if V                       ! Locator mode return
call CGL by ref ( SET_WRITING_MODE, CURRENT_MODE )
return
280 !
TRACE_MODE:
!
! Trace mode - no writing, just movement. Do not
! alter actual writing mode in locator mode.
!
MODE_NAME = "Move"
CURRENT_MODE = TRANSPARENT       ! set mode switch
gosub STATUS_LINE                ! Display Status Line
return if V                       ! Locator mode return
call CGL by ref ( SET_WRITING_MODE, CURRENT_MODE )
return
290 !
NOISE:
! Toggle beep on bad input
!
LOUD = not LOUD                  ! Toggle beep flag
return                            ! go back.
340 !
SET_HOME:
!
! Home is where the cursor is. Update status line.
!
HOMEX = CX \ HOMEY = CY
gosub STATUS_LINE                ! Display Status Line
return
350 !
HELP_IT:
!
! Help the user. No help yet.
!
return
!
700 ! Locator action routines
!
VECTOR_PLOT:
!
! Plot a vector from here to located point.
! leave cursor at end of vector.
!
OPTION_NAME = "Vector"           ! What routine we are
gosub STATUS_LINE                ! Display Status Line
PX = CX \ PY = CY                ! Save current position
gosub VECTOR_FIND                ! Find the end point

```


GEDIT.B2S - GRAPHICS SKETCHPAD

```

goto RESTORE_CURSOR if V1 < 0% ! Restore stuff if canceled
XX = CX \ YY = CY ! Prepare vector
gosub DRAW_VECTOR ! draw vector
OPTION_NAME = "Plot" ! Default option
gosub STATUS_LINE ! Display Status Line
return
720 !
RECTANGLE:
!
! Draw a rectangle by finding the oposite corner.
!
OPTION_NAME = "Rectangle"
gosub STATUS_LINE ! Display Status Line
PX = CX \ PY = CY
gosub VECTOR_FIND
goto RESTORE_CURSOR if V1 < 0
call CGL by ref ( MOVE_ABS_2, PX, PY ) ! Put corner back
call CGL by ref ( RECTANGLE_ABS_2, CX, CY )
goto RESTORE_CURSOR ! Put stuff back
730 !
CIRCLE:
!
! Draw a circle with center here and radius located
! Operation can be canceled.
!
OPTION_NAME = "Circle" ! Circle option
gosub STATUS_LINE ! Display Status Line
PX = CX ! Hang on to beginning
PY = CY ! point for use as center
gosub VECTOR_FIND ! Locate a radius
goto RESTORE_CURSOR if V1 < 0 ! Quit if canceled
call CGL by ref ( MOVE_ABS_2, CX, CY ) ! Back to center
call CGL by ref ( ARC_ABS_2, PX, PY, CIRCUMF )
735 !
RESTORE_CURSOR:
!
! Restore cursor after locator find
!
CX = PX \ CY = PY ! Reset current position
call CGL by ref ( MOVE_ABS_2, CX, CY )
OPTION_NAME = "Plot" ! Default option
gosub STATUS_LINE ! Display Status Line
return ! Return
10000 !
! Service routines...
!
VECTOR_FIND:
!
! Locator - Find endpoints of vector with one end here.
!
```

GEDIT.B2S - GRAPHICS SKETCHPAD

```

call CGL by ref ( SET_WRITING_MODE, COMPLEMENT)
V = 1 \ V1 = 0 \ XX = CX \ YY = CY
gosub DRAW_VECTOR
while V1 = 0%
    gosub DRAW_VECTOR
    XX = CX \ YY = CY
    gosub DRAW_VECTOR
    gosub COMMANDS
next
gosub DRAW_VECTOR
call CGL by ref ( SET_WRITING_MODE, CURRENT_MODE )
V = 0
return
10100 |
    DRAW_VECTOR:
    |
    | Draw Vector
    |
    | This routine plots a vector from point (PX,PY)
    | to point (XX,YY) in whatever writing mode.
    |
    call CGL by ref ( MOVE_ABS_2, PX, PY ) ! Beginning
    call CGL by ref ( LINE_ABS_2, XX, YY ) ! Plot to end
    return ! Return
10200 |
    STATUS_LINE:
    |
    | Display a status line at the bottom of the screen and
    | put the cursor back at the top of the screen
    |
    call CGL by ref (SET_WRITING_MODE, REPLACE)
    call CGL by ref (MOVE_ABS_2, 0.01, 0.58)
    STAT$ = FORMAT$(HOMEX, "Home: (###.###)" ) + &
            FORMAT$(HOMEY, "###.###" ) + &
            FORMAT$(OPTION_NAME, "Action Mode: 'LLLLLLLLL " ) + &
            FORMAT$(MODE_NAME, "Plot Mode: 'LLLL")
    call CGL by ref (TEXT, STAT$, LEN(STAT$))
    call CGL by ref (MOVE_ABS_2, CX, CY)
    if v then call CGL by ref (SET_WRITING_MODE, COMPLEMENT)
        else call CGL by ref (SET_WRITING_MODE, CURRENT_MODE)
    end if
    return
    |
32767 |
    ENDIT:
    |
    | end the program
    |
    call CGL by ref ( NEW_FRAME )
    call CGL by ref ( TERMINATE_CORE )
end

```


FONT.B2S - DISPLAY A FONT

D.3 FONT.B2S - DISPLAY A FONT

```

10      !
      ! This program is provided for instructional purposes.
      ! only. It demonstrates some aspects of the following
      ! software tools:
      !
      ! Tool Kit BASIC-PLUS-2 V2.1
      ! CORE Graphics Library V1.7
      !
      ! This application displays fonts. If you specify font 0,
      ! it displays the entire DEC Multinational set (including
      ! C0 and C1), as shown in Chapter 1.
      !
      ! If you specify a user-defined font, it assumes that data
      ! for one or more 16 x 16 characters exists as a terminal-
      ! format file on the target system. The first line of the
      ! file contains the number of characters. The next 16 lines
      ! specify the contents of a character definition matrix,
      ! followed by a single delimiter line, followed by another
      ! character matrix, and so forth.
      !
      %INCLUDE 'LB:[1,5]CGL.B2S'
      !
      CALL CGL BY REF (INITIALIZE_CORE)
      CALL CGL BY REF (NEW_FRAME)
      INPUT 'Font number'; FONT%
      IF FONT% = 0% THEN EXTENT% = 255% \ GOTO 20 \ END IF
      !
      ! User-defined font
      !
      CALL CGL BY REF (SET_FONT, FONT%)
      LINPUT 'File name'; FILE_NAME$
      OPEN FILE_NAME$ FOR INPUT AS FILE #1, ACCESS READ
      INPUT #1, EXTENT% \ EXTENT% = EXTENT% + 31%
      CALL CGL BY REF (SET_FONT_SIZE, 0%, 16%, 16%) ! Clear font
      CALL CGL BY REF (SET_FONT_SIZE, EXTENT%, 16%, 16%)
      !
      ! Convert terminal-format data to binary.
      !
      DIM CHAR_MATRIX%(15%)
      FOR CH% = 32% TO EXTENT% ! For each character
        FOR I% = 0% TO 15% ! For each definition line
          N% = 0% \ LINPUT #1, S$
          !
          ! Scan the line and set the appropriate bit for
          ! each non-space character.
          !
          N% = N% OR (2% ^ J%) &
          IF MID$(S$, J% + 1%, 1%) <> ' ' &
            FOR J% = 15% TO 0% STEP -1%

```

FONT.B2S - DISPLAY A FONT

```

        CHAR_MATRIX%(I%) = N%
    NEXT I%
    CALL CGL BY REF (LOAD_CHARACTER, CH%, CHAR_MATRIX%())
    LINPUT #1, S$ ! discard delimiter line
NEXT CH% \ CLOSE #1
!
! Display a font
!
20 CALL CGL BY REF (SET_WINDOW, -1.0, 16.0, -1.0, 16.0)
CALL CGL BY REF (NEW_FRAME)
IF FONT% = 0% THEN CH% = 0%
ELSE   CH% = 32%
      CALL CGL BY REF (SET_CHARSIZE,0.5,0.8)
END IF
CALL CGL BY REF (BEGIN_BATCH)
LOOP: FOR X = 0.0 TO 15.0
      FOR Y = 0.0 TO 15.0
        CALL CGL BY REF (MOVE_ABS_2, X, Y)
        CALL CGL BY REF (TEXT, CHR$(CH%), 1%)
        CH% = CH% + 1%
        EXIT LOOP IF CH% > EXTENT%
      NEXT Y
    NEXT X
CALL CGL BY REF (END_BATCH)
!
! Outline the window
!
CALL CGL BY REF (MOVE_ABS_2, -1.0, -1.0)
CALL CGL BY REF (RECTANGLE_ABS_2, 16.0, 16.0)
!
! Finish up
!
DIM FOO%(1%)
CALL GETKEY BY REF (FOO%()) ! Wait for input
CALL CGL BY REF (NEW_FRAME)
CALL CGL BY REF (TERMINATE_CORE)
END

```


FONT.B2S - DISPLAY A FONT

D.3.1 SUITS.FNT - SAMPLE USER-DEFINED FONT

For convenience, this file is shown in two columns.

4

```
X      !
XXX    !
XXXXX  !
XXXXXXX !
XXXXXXXX !
XXXXXXXXXX !
XXXXXXXXXXXX !
XXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXXXX !
XXXXX X XXXX !
XXX  X  XXX !
      X      !
      XXX    !
      XXXXX  !
      !
```

5432109876543210

```
XXX    XXX !
XXXXX  XXXX !
XXXXX  XXXX !
XXXXXXX XXXXXX !
XXXXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXXXX !
XXXXXXXXXXXX !
XXXXXXXXXX !
XXXXXXX !
XXXXXX !
XXX    !
X      !
      !
```

5432109876543210

```
X      !
XXX    !
XXXXX  !
XXXXXXX !
XXXXXXXX !
XXXXXXXXXX !
XXXXXXXXXXXX !
XXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXXXX !
XXXXX X XXXX !
XXX  X  XXX !
      X      !
      XXX    !
      XXXXX  !
      !
```

5432109876543210

```
XXX    !
XXXXX  !
XXXXXXX !
XXXXXXX !
XXXXXX !
      XXX !
      XXX X XXX !
      XXXXX X XXXX !
XXXXXXXXXXXXXXXXXXXX !
XXXXXXXXXXXXXXXXXXXX !
XXXXXX X XXXX !
XXX  X  XXX !
      X      !
      XXX    !
      XXXXX  !
      !
```

5432109876543210

MODE.B2S - DEMONSTRATE WRITING MODES

D.4 MODE.B2S - DEMONSTRATE WRITING MODES

```

10      |
      | This program is provided for instructional purposes.
      | only. It demonstrates some aspects of the following
      | software tools:
      |
      | Tool Kit BASIC-PLUS-2 V2.1
      | CORE Graphics Library V1.7
      |
      | This application was used to generate one of the figures
      | in Chapter One: The Writing Modes Shown with Line Style.
      | It draws two large filled areas, each covering one fourth
      | of the window, then seven horizontal lines, each in a
      | different writing mode.
      |
      |%INCLUDE 'LB:[1,5]CGL.B2S'
      |
      |CALL CGL BY REF (INITIALIZE_CORE)
      |CALL CGL BY REF (NEW_FRAME)
      |
      | Set up the color map for black and white
      |
      |DIM BLACK%(2%), WHITE%(2%)
      |DATA 0,0,0, 7,7,7
20      |READ BLACK%(I%) FOR I% = 0% TO 2%
      |READ WHITE%(I%) FOR I% = 0% TO 2%
      |CALL CGL BY REF (SET_COLOR_MAP_ENTRY, 7%, WHITE%())
      |CALL CGL BY REF (SET_COLOR_MAP_ENTRY, 0%, BLACK%())
      |
      | A line printer image is the negative of what appears on
      | the screen. So, make the screen image reversible.
      |
      |LINPUT 'Reverse image'; R$ \ R$ = EDIT$(R$, 32%)
      |LINPUT 'Print screen'; P$ \ P$ = EDIT$(P$, 32%)
      |IF R$ = 'Y' THEN
      |    CALL CGL BY REF (SET_WRITING_INDEX, 0%)
      |    CALL CGL BY REF (SET_BACKGROUND_INDEX, 7%)
      |ELSE
      |    CALL CGL BY REF (SET_WRITING_INDEX, 7%)
      |    CALL CGL BY REF (SET_BACKGROUND_INDEX, 0%)
      |END IF
      |CALL CGL BY REF (NEW_FRAME)
      |
      | Set up the window
      |
      |DECLARE REAL CONSTANT LOWER_X = 0, UPPER_X = 4, &
      |                        LOWER_Y = 0, UPPER_Y = 17
      |
      |CALL CGL BY REF (SET_WINDOW, LOWER_X, UPPER_X, &
      |                LOWER_Y, UPPER_Y)

```


MODE.B2S - DEMONSTRATE WRITING MODES

```

!
! Draw vertical fill for background
!
CALL CGL BY REF (SET_FILL_MODE, POLYGON)
CALL CGL BY REF (MOVE_ABS_2, 0, LOWER_Y)
CALL CGL BY REF (RECTANGLE_ABS_2, 1, UPPER_Y)
CALL CGL BY REF (MOVE_ABS_2, 2, LOWER_Y)
CALL CGL BY REF (RECTANGLE_ABS_2, 3, UPPER_Y)
CALL CGL BY REF (SET_FILL_MODE, OFF)
!
! Storage for visible mode names
!
DATA      "COMPLEMENT",      "COMPLEMENT NEGATE",      &
          "OVERLAY",         "OVERLAY NEGATE",         &
          "REPLACE",         "REPLACE NEGATE",        &
          "ERASE",           "ERASE NEGATE"
!
30 ! Draw a horizontal line for each mode
!
WRITING_MODE% = COMPLEMENT
CALL CGL BY REF (SET_LINEWIDTH, 0.0, 1.3)
CALL CGL BY REF (SET_LINEWIDTH_ORIENTATION, 0.0, 0.0)
CALL CGL BY REF (SET_LINestyle, DOT_DASHED, 0%, 0%)
FOR Y = 1 TO 16 STEP 2
    CALL CGL BY REF (BEGIN_BATCH)
    CALL CGL BY REF (SET_WRITING_MODE, WRITING_MODE%)
    READ WM$ \ WRITING_MODE% = WRITING_MODE% + 1%
    CALL CGL BY REF (MOVE_ABS_2, LOWER_X, Y)
    CALL CGL BY REF (LINE_ABS_2, UPPER_X, Y)
    CALL CGL BY REF (MOVE_ABS_2, LOWER_X + 0.05, Y)
    CALL CGL BY REF (SET_WRITING_MODE, REPLACE_NEGATE)
    CALL CGL BY REF (TEXT, WM$, LEN(WM$))
    CALL CGL BY REF (END_BATCH)
NEXT Y
!
! Outline the window
!
CALL CGL BY REF (SET_LINEWIDTH, 0.0, 0.0)
CALL CGL BY REF (SET_LINestyle, SOLID, 0%, 0%)
CALL CGL BY REF (SET_WRITING_MODE, OVERLAY)
IF R$ = 'Y'
    THEN CALL CGL BY REF (SET_WRITING_INDEX, 7%)
    ELSE CALL CGL BY REF (SET_WRITING_INDEX, 0%)
END IF
CALL CGL BY REF (MOVE_ABS_2, LOWER_X, LOWER_Y)
CALL CGL BY REF (RECTANGLE_ABS_2, UPPER_X, UPPER_Y)
!
! Finish up
!
40 CALL CGL BY REF (PRINT_SCREEN, LOWER_X, UPPER_X, &
                  LOWER_Y, UPPER_Y, 0, 0) IF P$ = 'Y'

```

MODE.B2S - DEMONSTRATE WRITING MODES

```
DIM FOO%(1%)  
CALL GETKEY BY REF (FOO%()) ! Wait for input  
CALL CGL BY REF (NEW_FRAME)  
CALL CGL BY REF (TERMINATE_CORE)  
END
```




APPENDIX E

SUMMARY OF INSTRUCTIONS

In BASIC-PLUS-2, the data types of constants and (implicit-created) variables can be observed at a glance. Thus, in this appendix, the following BASIC-PLUS-2 conventions are used to indicate the data types of CGL instruction parameters.

- No suffix indicates a real (two-word, floating point) type.
- A percent sign (%) indicates an integer (16-bit, signed) type.
- A dollar sign (\$) indicates a string (array of character) type.
- A subscript indicates an array.

NOTE

This summary does not include symmetric INQUIRE instructions. All SET instructions have a corresponding INQUIRE instruction with the same parameter list.

ARC_ABS_2 (x, y, angle%)

Draws an arc of a circle whose center is at a specified position, beginning at the current position, and continuing for a specified number of degrees.

ARC_REL_2 (x, y, angle%)

Draws an arc of a circle whose center is at a specified offset, beginning at the current position, and continuing for a specified number of degrees.

SUMMARY OF INSTRUCTIONS

BEGIN_BATCH

Begins storing all subsequent view surface updates in a buffer and continues to do so until END_BATCH or until the buffer is full.

BEGIN_DEFINE_CHARACTER (code%)

Begins the definition of a character.

CGL_WAIT (seconds)

Suspends all changes to the video monitor screen for a specified period of real time.

CURVE_ABS_2 (x_array, y_array, n%, type%)

Draws a smooth curve connecting a list of positions.

CURVE_REL_2 (x_array, y_array, n%, type%)

Draws a smooth curve connecting a list of offsets.

DESELECT_VIEW_SURFACE (name\$, length%)

The DESELECT_VIEW_SURFACE instruction removes a specific device from the set of devices to which CGL performs output.

END_BATCH

Performs all view surface updates stored since the last BEGIN_BATCH.

END_DEFINE_CHARACTER

Ends the definition of a character.

ERASE_VIEWPORT

Clears the viewport.

INITIALIZE_CORE

Guarantees that the graphics system is in a standard state with default parameters established.

INITIALIZE_VIEW_SURFACE (name\$, length%)

Prepares (does not implicitly select) a specific output device for operation.

SUMMARY OF INSTRUCTIONS

INQUIRE_CURRENT_POSITION_2 (x, y)

Returns the current world coordinate position.

INQUIRE_TEXT_EXTENT_2 (length%, delta_x, delta_y)

Reports the amount of world coordinate space that would be used to draw a string of the indicated length, unjustified, beginning at the current position.

LINE_ABS_2 (x, y)

Changes the current position to the specified position and draws a connecting line.

LINE_REL_2 (delta_x, delta_y)

Changes the current position to the specified offset and draws a connecting line.

LOAD_CHARACTER (code%, matrix%())

Loads a character into the current user-defined font.

LOAD_FONT (name\$, length%)

Loads characters into the current user-defined font from a named region.

MARKER_ABS_2 (x, y)

Changes the current position to the specified position and draws a marker.

MARKER_REL_2 (delta_x, delta_y)

Changes the current position to the specified offset and draws a marker.

MOVE_ABS_2 (x, y)

Changes the current position to the specified position.

MOVE_REL_2 (delta_x, delta_y)

Changes the current position to the specified offset.

NEW_FRAME

Clears currently selected view surfaces.

SUMMARY OF INSTRUCTIONS

PLAYBACK_FILE (name\$, length%)

Executes a file of GIDIS commands.

POLYGON_ABS_2 (x_array, y_array, n%)

Draws a series of connected lines starting and ending at the first position in the specified list.

POLYGON_REL_2 (dx_array, dy_array, n%)

Draws a series of connected lines starting and ending at the first offset in the specified list.

POLYLINE_ABS_2 (x_array, y_array, n%)

Draws a series of connected lines starting at the current position and ending at the last position in the specified list.

POLYLINE_REL_2 (dx_array, dy_array, n%)

Draws a series of connected lines starting at the current position and ending at the last offset in the specified list.

POLYMARKER_ABS_2 (x_array, y_array, n%)

Changes the current position to each of a list of positions and draws a marker at each position.

POLYMARKER_REL_2 (dx_array, dy_array, n%)

Changes the current position to each of a list of offsets and draws a marker at each offset.

PRINT_SCREEN (lower_x, upper_x, lower_y, upper_y, x_offset, y_offset)

Sends an image of the video monitor screen to a graphics printer (LA50 or LA100).

RECTANGLE_ABS_2 (x, y)

Draws a series of connected lines forming a four-sided, perpendicular, polygon with the current position at one corner and the specified point at the opposing corner.

SUMMARY OF INSTRUCTIONS

RECTANGLE_REL_2 (dx, dy)

Draws a series of connected lines forming a four-sided, perpendicular, polygon with the current position at one corner and the specified offset at the opposing corner.

REPORT_MOST_RECENT_ERROR (inst_name%, code%)

Reports the instruction number and error code associated with the most recent execution error and returns the system to a nonerror state.

SCROLL (delta_x, delta_y)

Moves the contents of the entire video monitor screen by a specified amount of world coordinate space.

SCROLL_VIEWPORT (delta_x, delta_y)

Moves the contents of the viewport by a specified amount of world coordinate space.

SELECT_VIEW_SURFACE (name\$, length%)

Adds a specific device to the set of view surfaces to which CGL performs output.

SET_BACKGROUND_INDEX (index%)

Specifies an index into the color map for the background.

SET_CHARITALIC (angle%)

Specifies the slant of the individual characters in a text string.

SET_CHARJUST (x_just%, y_just%)

Specifies the starting position of text primitives relative to the current position.

SET_CHARPATH (path%, mode%)

In character mode, specifies the path (relative to horizontal) of individual characters. In string mode, specifies the path (relative to horizontal) of entire strings.

SUMMARY OF INSTRUCTIONS

SET_CHARSIZE (width, height)

Specifies the X and Y size, in world coordinate units, of text primitives.

SET_CHARSPACE (delta_x, delta_y)

Specifies the horizontal and vertical displacement between the starting points of adjacent letters.

SET_COLOR_MAP (color_map%)

Specifies the RGB values of the entire color map and of any existing images.

SET_COLOR_MAP_ENTRY (entry%, color%)

Specifies the RGB values of an individual color map entry and of any existing images drawn with that entry.

SET_CURSOR (font%, char%, width%, height%, dx, dy)

Controls the appearance of the cursor, the visual representation of the current position.

SET_FILL_CHAR (font%, char%, width_mult%, height_mult%)

Specifies the character used for area fill.

SET_FILL_ENTITY (x, y)

Specifies the line or point used as the reference for area fill.

SET_FILL_MODE (mode%)

Specifies the current area fill mode: (off, vertical line, horizontal line, point).

SET_FONT (font%)

Specifies one of the four available character fonts.

SET_FONT_SIZE (extent%, x_size%, y_size%)

Initializes the current user-defined font: specifies the highest-numbered character and the size of the characters in physical device coordinate units.

SUMMARY OF INSTRUCTIONS

SET_GLOBAL_ATTRIBUTES (int_list%, real_list%)

Specifies the values of the entire global output primitive attribute list.

SET_LINESTYLE (style%, pattern%, mult%)

Specifies the current pattern for line primitives.

SET_LINEWIDTH (dx, dy)

Specifies the X and Y width of line primitives in world coordinate units.

SET_LINEWIDTH_ORIENTATION (dx, dy)

Specifies the offset from the end of a line primitive to the actual drawing position specified in the line primitive instruction.

SET_MARKER_SYMBOL (symbol%, code%)

Specifies one of five standard symbols or a user-defined symbol as the current marker symbol.

SET_NDC_SPACE_2 (width, height)

Defines normalized device coordinate space.

SET_ORIGIN (origin%)

Specifies which corner of the viewport corresponds to the origin of the window.

SET_VIEWPORT_2 (xmin, xmax, ymin, ymax)

Specifies a portion of normalized device coordinate space to be the viewport and resets the current position to the origin of the window.

SET_WINDOW (xmin, xmax, ymin, ymax)

Specifies the edges of the window and resets the current position to the origin of the window.

SET_WINDOW_CLIPPING (on_off%)

Controls the display of output primitives (or portions of output primitives) that fall outside of the window.

SUMMARY OF INSTRUCTIONS

SET_WRITING_INDEX (index%)

Specifies an index into the color map for images created by subsequent output primitive instructions.

SET_WRITING_MODE (mode%)

Specifies the exact manner in which CGL draws output primitives.

SET_WRITING_PLANES (n%)

Specifies which of the three bitmap planes can be written into by CGL.

TERMINATE_CORE

Releases all resources used by the CGL system.

TERMINATE_VIEW_SURFACE (name\$, length%)

Terminates access to and releases a specific output device.

TEXT (string\$, length%)

Draws a line of text.

APPENDIX F

GLOSSARY

The words in this glossary are used throughout this manual. These definitions are not absolute and may differ somewhat in other contexts. Where possible, the CORE Standard usage is the basis of the definition.

ATTRIBUTE

One of a CGL-maintained list of values that determine the characteristics of appearance of output primitives.

See also OUTPUT PRIMITIVE.

BASIC-PLUS-2

The Professional Developer's Tool Kit implementation of BASIC-PLUS-2, a highly extended compiler for BASIC (Beginner's All-purpose Symbolic Instruction Code), a widely-used programming language.

CLIPPING

The state in which output primitives occupying world coordinate positions outside the window do not appear on the view surface.

See also OUTPUT PRIMITIVE, WINDOW, WORLD COORDINATES.

CURRENT POSITION

The world coordinate position that defines the current drawing location.

See also WORLD COORDINATES.

GLOSSARY

CURSOR

In text mode, the cursor is the visual representation of where the next character will appear. It is indicated by a blinking character at that position or by a blinking underline.

While CGL is operational, the cursor is the visual representation of the current position. It is indicated by blinking cross-hairs or a character that you specify.

On a printing terminal, the cursor is considered to be the current location of the print head.

See also GRAPHICS MODE, TEXT MODE.

GRAPHICS MODE

A mode of operation in which physical device coordinate positions can be addressed and written to. Graphics mode and text mode are mutually exclusive. CGL is an interface between the Professional graphics mode and application programs.

See also PHYSICAL DEVICE COORDINATES, TEXT MODE.

IMAGE

A view of one or more graphical objects.

See also VIEWING TRANSFORMATION.

NORMALIZED DEVICE COORDINATES (NDC)

Device-independent Cartesian coordinates in the range 0 to 1 for specifying the viewport.

See also PHYSICAL DEVICE COORDINATES, VIEWPORT, WORLD COORDINATES.

OUTPUT PRIMITIVE

A part of a picture, such as a geometric object or a text string, that has a specific appearance. Values of attributes determine some aspects of the appearance.

PHYSICAL DEVICE COORDINATES

Device-dependent Cartesian coordinates for specifying locations on the view surface of an output device. The Professional's physical device coordinate space is 960 (horizontal) by 600 (vertical) units. Some special CGL instructions accept binary values where each bit corresponds to a physical device coordinate

GLOSSARY

unit.

SCREEN

A two-dimensional, physical view surface upon which images are drawn; specifically the Professional 300 Series video monitor.

TEXT MODE

A terminal subsystem mode in which the video display is divided into discrete rectangular cells, each consisting of 12 X 25 physical device coordinate units, that are treated as the smallest unit of display resolution.

See also PHYSICAL DEVICE COORDINATES, GRAPHICS MODE.

VIEWING TRANSFORMATION

A transformation that maps world coordinates to normalized device coordinates (which can include clipping).

See also CLIPPING, NORMALIZED DEVICE COORDINATES, WORLD COORDINATES.

VIEWPORT

The currently used portion of normalized device coordinate space.

See also NORMALIZED DEVICE COORDINATES.

VIEW SURFACE

The visual display component of a physical output device.

WORLD COORDINATES

Device-independent Cartesian coordinates defined by the application program to describe data to CGL.

See also CURRENT POSITION, NORMALIZED DEVICE COORDINATES, PHYSICAL DEVICE COORDINATES, VIEWING TRANSFORMATION.



INDEX

-A-

Absolute position
definition, 1-12

ACM
and CORE Standard, 1-1, 1-3

Application Builder
see PAB

Arc
appearance, 1-14
drawing, 7-8 to 7-9

ARC_ABS_2
general description, 1-19
reference description, 7-8
summary description, E-1

ARC_REL_2
general description, 1-19
reference description, 7-9
summary description, E-1

Aspect ratio
viewport, 1-14

Association
see ACM

Attribute
default values, 1-5
definition, 1-21
glossary definition, F-1
instruction, 1-5

-B-

Background
definition of, 1-21

Background index
default value, 5-3
definition, 1-33
in ERASE VIEWPORT, 3-10
in replace mode, 1-38
in replace negate mode, 1-38
in scrolling, 4-6, 4-8
on plotter, B-4
setting, 5-2, 5-9

BASIC-PLUS-2
array numbering, 7-4 to 7-5
array parameter, 2-2
CALL statement, 2-1
data types, E-1

glossary definition, F-1
include file, 2-1
listing, C-1
NOECHO function, 2-7
string parameter, 2-2
WAIT statement, 3-10

BEGIN_BATCH
general description, 1-6
reference description, 3-9
summary description, E-2

BEGIN_DEFINE_CHARACTER
reference description, 8-4
summary description, E-2

Bitmap
definition of, 1-30
interface to color map, 1-34

-C-

Calling Sequence
FORTRAN, 2-1
PDP-11 R5, 2-1

Cartesian coordinate
definition, 1-7

CGL
definition of, xi

CGL_WAIT
general description, 1-6
reference description, 3-10
summary description, E-2

Character
as current pattern, 1-36
half-size, 8-7
inversion, 8-7
user-defined, 1-20, 1-26, 8-3

Character italic
default value of, 8-11
definition, 1-29
setting, 5-9, 8-10

Character justification
default value of, 8-10
definition, 1-28
setting, 5-9, 8-10

Character path
character mode, 8-8
default value of, 8-9
definition, 1-27

INDEX

- effect on character spacing,
 - 8-8
 - setting, 5-9, 8-8
 - string mode, 8-8
 - Character size
 - computing, 1-25
 - default value of, 8-7
 - definition, 1-25
 - setting, 5-9, 8-6
 - standard, 1-26
 - Character spacing
 - default value of, 8-8
 - definition, 1-27
 - setting, 5-9, 8-7
 - Circle
 - drawing, 7-8, 7-10
 - Clipping
 - controlling, 4-3
 - definition, 1-14
 - glossary definition, F-1
 - COBOL-81
 - CALL statement, 2-2
 - integer to real conversion, 2-3
 - text declaration, 2-3
 - Color
 - complementary, 1-31
 - formation by addition, 1-31
 - Color map
 - background index, 5-2
 - default values, 1-5, 1-33
 - description, 1-32
 - example program, D-1
 - instruction, 1-32
 - interface to bitmap, 1-34
 - setting, 5-3 to 5-4
 - setting for complement mode,
 - 1-37
 - writeable entries, 5-6
 - writing index, 5-1
 - Complement mode
 - and cursor, 1-11
 - description, 1-36
 - setting, 5-7
 - when not to use, 7-15
 - Complement negate mode
 - description, 1-37
 - setting, 5-7
 - Complementary color
 - definition, 1-31
 - Control instruction
 - using
 - description, 1-4
 - Coordinate space
 - definition, 1-7
 - CORE Standard
 - character size, 8-7
 - compatibility, ix, 1-1
 - definition of, xi
 - description, 1-2
 - function name, x
 - function names, 1-16
 - Current pattern
 - definition, 1-36
 - Current position
 - description, 1-11
 - glossary definition, F-1
 - instruction, 1-17, 6-1
 - obtaining, 6-2
 - resetting, 4-2, 4-5
 - setting, 6-1 to 6-2, 6-4 to 6-6,
 - 7-1 to 7-3, 7-5 to 7-6, 7-8
 - to 7-9, 7-11 to 7-12
 - Cursor
 - glossary definition, F-2
 - purpose, 1-11
 - setting, 6-3
 - Curve
 - appearance, 1-15
 - drawing, 7-11 to 7-12
 - CURVE_ABS_2
 - general description, 1-19
 - reference description, 7-11
 - summary description, E-2
 - CURVE_REL_2
 - general description, 1-19
 - reference description, 7-12
 - summary description, E-2
- D-
- DEC Multinational Character Set
 - contents, 1-29
 - decimal code, 8-4 to 8-5, 8-12
 - in Font 0, 8-11
 - DESELECT VIEW SURFACE
 - general description, 1-6
 - reference description, 3-7
 - summary description, E-2
 - DIBOL
 - include file, 2-4
 - listing, C-4
 - XCALL statement, 2-3

INDEX

-E-

EBO
 and color map, 1-32
 description, 1-31
 effect on complement mode, 1-37
 with monochrome device, 1-31

Echo
 disabling, 2-7

END_BATCH
 general description, 1-6
 implicit, 3-2
 reference description, 3-9
 summary description, E-2

END_DEFINE_CHARACTER
 reference description, 8-5
 summary description, E-2

Erase mode
 description, 1-38
 on plotter, B-4
 setting, 5-7
 use of background index, 5-2

Erase negate mode
 description, 1-38
 on plotter, B-4
 setting, 5-7
 use of writing index, 5-1

ERASE_VIEWPORT
 effect on background index, 5-2
 general description, 1-7
 reference description, 3-10
 summary description, E-2

Error message
 behavior of plotter, B-9
 complete list, A-1

Error reporting
 see REPORT_MOST_RECENT_ERROR

Extended Bitmap Option
 see EBO

-F-

Fill
 definition, 1-23

Fill character
 default value of, 7-20
 definition, 1-24
 on plotter, B-5
 setting, 5-9, 7-20

Fill entity
 default value, 7-19

definition, 1-23
 purpose, 1-23
 resetting, 4-1 to 4-2, 4-5
 setting, 5-9, 7-19

Fill mode
 setting, 5-9, 7-18

Font
 definition, 1-29
 on plotter, B-8
 setting, 5-9, 8-11
 user-defined, 3-6, 8-2 to 8-3,
 8-11
 example, D-14

Font editor
 definition, 1-20
 example program, D-12

Font size
 definition, 1-30
 on plotter, B-8
 setting, 8-12

FORTRAN
 CALL statement, 2-4
 include file, 2-4
 listing, C-6

-G-

GETKEY
 for terminal I/O, 2-7

GIGI
 and ReGIS, 1-2

Global attribute
 default values, 5-9
 definition, 1-16

Global attribute list
 description, 1-38
 setting, 5-8

Graphics mode
 accessing, 1-1
 glossary definition, F-2

-I-

Image
 correcting distortion, 1-13
 glossary definition, F-2

INITIALIZE_CORE
 general description, 1-5
 reference description, 3-1
 summary description, E-2
 using, 1-5

INDEX

INITIALIZE VIEW SURFACE
 general description, 1-5
 reference description, 3-3
 summary description, E-2
INQUIRE instruction
 format conventions, x
 purpose, ix
 symmetric, E-1
INQUIRE CURRENT POSITION 2
 general description, 1-17
 reference description, 6-2
 summary description, E-3
INQUIRE TEXT EXTENT 2
 general description, 1-20
 reference description, 8-2
 summary description, E-3
Installation
 command file, 2-10
Instruction
 format conventions, x
Instruction name
 meaning of suffix, 1-16
 selection, ix
Interpolation
 definition, 1-19
Italic
 see Character italic

-L-

Line
 drawing, 7-1 to 7-7
Line attribute
 instruction, 1-21, 7-13
Line primitive
 instruction, 1-18, 7-1, 7-8
Line style
 as current pattern, 1-36
 as fill character, 1-24
 default value, 7-13
 definition, 1-21
 multiplier, 7-14
 on plotter, B-5
 setting, 5-9, 7-13
 vertical for fill, 7-20
Line width
 default value of, 7-15
 definition, 1-22
 effect of sign, 7-16
 effect on drawing speed, 7-15
 on plotter, B-5

 setting, 5-9, 7-14
Line width orientation
 default value, 7-18
 definition, 1-22
 setting, 7-16
LINE ABS 2
 effect on current position,
 1-11
 general description, 1-18
 reference description, 7-1
 summary description, E-3
LINE REL 2
 general description, 1-18
 reference description, 7-1
 summary description, E-3
LOAD CHARACTER
 general description, 1-20
 non-effect on plotter, B-2
 reference description, 8-3
 summary description, E-3
LOAD FONT
 general description, 1-20
 non-effect on plotter, B-2
 reference description, 8-2
 summary description, E-3
Logical unit number
 assigning, 2-9
LUN
 see logical unit number

-M-

MACRO-11
 calling CGL from, 2-6
Marker
 definition, 1-17
 drawing, 6-4 to 6-6
 symbol
 default value of, 6-8
Marker attribute
 instruction, 1-24, 6-7
Marker primitive
 instruction, 1-17, 6-4
Marker symbol
 definition, 1-24
 on plotter, B-4
 setting, 5-9, 6-7
MARKER ABS 2
 general description, 1-17
 reference description, 6-4
 summary description, E-3

INDEX

MARKER_REL_2
 general description, 1-17
 reference description, 6-5
 summary description, E-3

Monochromatic image
 description, 1-30

MOVE_ABS_2
 effect on current position,
 1-12
 general description, 1-17
 reference description, 6-1
 summary description, E-3

MOVE_REL_2
 effect on current position,
 1-12
 general description, 1-17
 reference description, 6-2
 summary description, E-3

-N-

NEW_FRAME
 effect on background index, 5-2
 general description, 1-5
 non-effect on plotter, B-2
 reference description, 3-2
 summary description, E-3

Normalized device coordinate
 definition, 1-8
 description, 1-12
 glossary definition, F-2
 space, 1-12

**Normalized device coordinate
 (NDC)**
 space, 1-13

-O-

Origin
 as default fill entity, 7-19
 definition, 1-10
 effect on scrolling, 4-6, 4-8
 resetting, 4-1 to 4-2

Output primitive
 description, 1-16
 factors affecting, 1-16
 glossary definition, F-2
 instruction, 1-4

Overlay mode
 description, 1-37
 on plotter, B-4

setting, 5-7
 use of writing index, 5-1

Overlay negate mode
 description, 1-37
 setting, 5-7
 use of writing index, 5-1

-P-

P/OS User Interface Library
 using with CGL, 2-8

PAB
 command file, 1-5, 2-8
 overlay descriptor file, 2-8

PASCAL
 include file, 2-5
 listing, C-9
 parameter passing, 2-6
 READONLY attribute, 2-6
 SEQ11 declaration, 2-5
 TEXT identifier, 8-1
 UNSAFE attribute, 2-6

Physical device coordinate
 definition, 1-8, 1-14
 glossary definition, F-2
 space, 1-14, B-2

Pixel
 general definition, 1-14

Plane
 definition of, 1-30
 selecting, 1-35, 5-6

PLAYBACK FILE
 general description, 1-6
 reference description, 3-8
 summary description, E-4

Plotter
 Hewlett-Packard, B-1, B-10

POLYGON_ABS_2
 general description, 1-18
 reference description, 7-4
 summary description, E-4

POLYGON_REL_2
 general description, 1-18
 reference description, 7-5
 summary description, E-4

POLYLINE_ABS_2
 general description, 1-18
 reference description, 7-2
 summary description, E-4

POLYLINE_REL_2
 general description, 1-18

INDEX

- reference description, 7-3
 - summary description, E-4
 - POLYMARKER_ABS_2**
 - general description, 1-17
 - reference description, 6-5
 - summary description, E-4
 - POLYMARKER_REL_2**
 - general description, 1-18
 - reference description, 6-6
 - summary description, E-4
 - Portability**
 - of programs, x, 1-3
 - POSRES**
 - using with CGL, 2-8
 - Primary color**
 - in light, 1-31
 - PRINT_SCREEN**
 - general description, 1-7
 - reference description, 3-11
 - summary description, E-4
 - PRO/GIDIS**
 - description, 1-1
 - Professional Application Builder**
 - see PAB
 - Pythagorean Theorem**
 - example, 7-9 to 7-10
- Q-
- QIO**
 - access to PRO/GIDIS, 1-2
- R-
- RECTANGLE_ABS_2**
 - general description, 1-19
 - reference description, 7-6
 - summary description, E-4
 - RECTANGLE_REL_2**
 - general description, 1-19
 - reference description, 7-7
 - summary description, E-5
 - ReGIS**
 - description, 1-2
 - implementation, 1-2
 - Relative position**
 - definition, 1-12
 - Replace mode**
 - description, 1-38
 - setting, 5-7
 - use of background index, 5-2
 - Replace negate mode**
 - description, 1-38
 - setting, 5-7
 - use of background index, 5-2
 - REPORT_MOST_RECENT_ERROR**
 - and carry bit, 2-7
 - general description, 1-7
 - reference description, 3-12
 - summary description, E-5
 - RGB value**
 - definition, 1-32
 - setting, 5-3 to 5-4
 - with monochrome device, 1-34
 - Rounding errors**
 - in consecutive arcs, 7-8, 7-10
- S-
- Screen**
 - glossary definition, F-3
 - printing
 - see PRINT_SCREEN
 - SCROLL**
 - non-effect on plotter, B-2
 - reference description, 4-6
 - summary description, E-5
 - SCROLL_VIEWPORT**
 - non-effect on plotter, B-2
 - reference description, 4-7
 - summary description, E-5
 - Scrolling**
 - of screen, 4-6
 - of viewport, 4-7
 - SELECT_VIEW_SURFACE**
 - general description, 1-6
 - reference description, 3-5
 - summary description, E-5
 - SET instruction**
 - format conventions, x
 - SET_BACKGROUND_INDEX**
 - effect on plotter, B-4
 - general description, 1-33
 - reference description, 5-2
 - summary description, E-5
 - SET_CHARITALIC**
 - effect on character size, 1-26
 - general description, 1-29
 - reference description, 8-10
 - summary description, E-5
 - use of with SET_CHARPATH, 8-9
 - SET_CHARJUST**

INDEX

- general description, 1-28
- reference description, 8-10
- summary description, E-5
- SET CHARPATH**
 - effect on character size, 1-26
 - general description, 1-27
 - modes, 1-27
 - reference description, 8-8
 - summary description, E-5
- SET CHARSIZE**
 - general description, 1-25
 - reference description, 8-6
 - summary description, E-6
- SET CHARSPACE**
 - general description, 1-27
 - reference description, 8-7
 - summary description, E-6
 - with SET CHARPATH, 1-28
- SET COLOR MAP**
 - EBO requirement, 5-1
 - general description, 1-32
 - non-effect on plotter, B-2
 - reference description, 5-4
 - summary description, E-6
- SET COLOR_MAP_ENTRY**
 - EBO requirement, 5-1
 - general description, 1-33
 - non-effect on plotter, B-3
 - reference description, 5-3
 - summary description, E-6
- SET CURSOR**
 - general description, 1-11, 1-17
 - non-effect on plotter, B-3
 - reference description, 6-3
 - summary description, E-6
- SET FILL CHAR**
 - effect on plotter, B-5
 - general description, 1-24
 - reference description, 7-20
 - summary description, E-6
- SET FILL ENTITY**
 - general description, 1-23
 - reference description, 7-19
 - summary description, E-6
- SET FILL MODE**
 - general description, 1-23
 - reference description, 7-18
 - summary description, E-6
- SET FONT**
 - effect on plotter, B-8
 - general description, 1-29
 - reference description, 8-11
 - summary description, E-6
- SET_FONT_SIZE**
 - effect on plotter, B-8
 - effect on view surfaces, 3-6
 - general description, 1-30
 - reference description, 8-12
 - relationship to LOAD_CHARACTER, 8-4 to 8-5
 - summary description, E-6
 - use of with LOAD_CHARACTER, 8-4 to 8-5
 - use of with LOAD_FONT, 8-3
- SET_GLOBAL_ATTRIBUTES**
 - general description, 1-38
 - reference description, 5-8
 - summary description, E-7
- SET LINESTYLE**
 - effect on plotter, B-5
 - general description, 1-21
 - reference description, 7-13
 - summary description, E-7
- SET LINEWIDTH**
 - effect on plotter, B-5
 - general description, 1-22
 - reference description, 7-14
 - summary description, E-7
- SET LINEWIDTH_ORIENTATION**
 - general description, 1-22
 - reference description, 7-16
 - summary description, E-7
- SET_MARKER_SYMBOL**
 - effect on plotter, B-4
 - general description, 1-24
 - reference description, 6-7
 - summary description, E-7
- SET_NDC_SPACE_2**
 - reference description, 4-3
 - summary description, E-7
- SET_ORIGIN**
 - general description, 1-11
 - reference description, 4-2
 - summary description, E-7
- SET_VIEWPORT_2**
 - general description, 1-13
 - reference description, 4-5
 - summary description, E-7
- SET_WINDOW**
 - general description, 1-10
 - reference description, 4-1
 - summary description, E-7

INDEX

SET_WINDOW_CLIPPING
 general description, 1-14
 reference description, 4-3
 summary description, E-7

SET_WRITING_INDEX
 effect on plotter, B-3
 general description, 1-33
 reference description, 5-1
 summary description, E-8

SET_WRITING_MODE
 effect on plotter, B-4
 general description, 1-36
 reference description, 5-7
 summary description, E-8

SET_WRITING_PLANES
 EBO requirement, 5-1
 general description, 1-35
 non-effect on plotter, B-3
 reference description, 5-6
 summary description, E-8

SIGGRAPH
 and CORE Standard, 1-1, 1-3

Slant
 see Character italic

Special Interest Group
 see SIGGRAPH

Suspending execution
 see CGL_WAIT

-T-

Terminal Subsystem
 modes, 1-1

TERMINATE_CORE
 general description, 1-5
 reference description, 3-2
 summary description, E-8

TERMINATE_VIEW_SURFACE
 general description, 1-6
 reference description, 3-4
 summary description, E-8

TEXT
 general description, 1-20
 reference description, 8-1
 summary description, E-8

Text
 appearance, 1-15
 drawing, 8-1

Text attribute
 instruction, 1-25, 8-6

Text extent

 definition, 8-2

Text mode
 accessing, 1-1
 and graphics, 1-19
 glossary definition, F-3

Text primitive
 instruction, 1-19, 8-1

Transparent mode
 description, 1-36
 on plotter, B-4
 setting, 5-7

Transparent negate mode
 description, 1-36
 on plotter, B-4
 setting, 5-7

-V-

Video monitor
 initialization and selection,
 3-1

View surface
 erasing
 see NEW_FRAME
 font definition, 8-12
 glossary definition, F-3
 in viewing transformation, 1-14
 optional, B-1

Viewing transformation
 default values, 1-5
 definition, 1-9, 1-14
 effect on arcs, 7-8 to 7-9
 effect on line width, 1-22
 effect on output primitives,
 1-16
 glossary definition, F-3
 instruction, 1-4

Viewport
 aspect ratio, 1-14
 default values of, 4-5
 definition, 1-13
 erasing
 see ERASE_VIEWPORT
 glossary definition, F-3
 in viewing transformation, 1-14
 setting, 4-5

VT125
 and ReGIS, 1-2
 color map compatibility, 1-33,
 1-37

INDEX

-W-

Window

- default value of, 4-2
- description, 1-10
- effect on character size, 8-7
- in viewing transformation, 1-14
- origin
 - see Origin
- resetting, 4-1
- simulating multiple, 3-11

World coordinate

- definition, 1-7
- description, 1-8
- glossary definition, F-3
- space, 1-9 to 1-10

Writing index

- default value, 5-2
- definition, 1-33
- in erase negate mode, 1-38
- in overlay mode, 1-37
- in overlay negate mode, 1-37
- in replace mode, 1-38
- in replace negate mode, 1-38
- on plotter, B-3
- setting, 5-1, 5-9

Writing mode

- definition, 1-36
- example program, D-15
- on plotter, B-4
- setting, 5-7, 5-9
- use of background index, 5-2
- use of writing index, 5-1



READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized?
Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

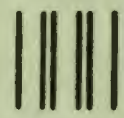
City _____ State _____ Zip Code _____

or
Country

Please cut along this line

----- Do Not Tear - Fold Here and Tape -----

digital

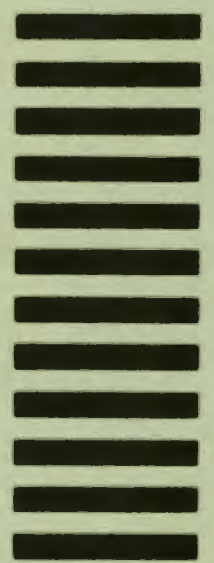


No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional 300 Series Publications
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754



----- Do Not Tear - Fold Here -----

Cut Along Dotted Line

