IRIS User's Guide
Volume II
Reference Guide

Version 4.0

Document Number 007-1101-040

IRIS User's Guide
Volume II
Reference Guide
Version 4.0
Document Number 007-1101-040

Silicon Graphics, Inc.
Mountain View, California

Contents

VOLUME I

GRAPHICS PROGRAMMING

1. Introduction 1-1
1.1 System Overview 1-1
1.2 The Graphics Library 1-3
1.3 Documentation Conventions 1-5
1.4 Related Publications 1-6

2. Global State Attributes 2-1
2.1 Initialization 2-1
2.2 Saving Global State Attributes 2-7

3. Drawing Routines 3-1
3.1 Current Drawing Positions 3-2
3.2 Clearing the Viewport 3-3
3.3 Points 3-3
3.4 Lines 3-5
3.4.1 Relative Drawing 3-7
3.5 Rectangles 3-8
3.6 Polygons 3-10
3.7 Circles and Arcs 3-13
3.7.1 Circles 3-13
3.7.2 Arcs 3-14
3.8 Text 3-16
3.9 Writing and Reading Pixels 3-19

4. Coordinate Transformations 4-1
4.1 Modeling Transformations 4-2
4.2 Viewing Transformations 4-7
4.3 Projection Transformations 4-10
4.4 Viewports 4-15
4.5 User-Defined Transformations 4-20

5. Linestyles, Patterns, and Fonts 5-1
5.1 Linestyles 5-1
5.1.1 Modifying the Linestyle Pattern 5-3
5.2 Patterns 5-6
5.3 Fonts 5-9

6. Display and Color Modes 6-1
6.1 Display Modes 6-1
6.1.1 RGB Mode 6-3
6.1.2 Single Buffer and Double Buffer Modes 6-3
6.2 Color Maps 6-8
6.3 Colors and Writemasks 6-12
6.3.1 Colors 6-12
6.3.2 Writemasks 6-16
6.4 Cursors 6-19

7. Input/Output Routines 7-1
7.1 Polling and Queueing 7-1
7.2 Initializing a Device 7-4
7.3 Polling a Device 7-6
7.4 The Event Queue 7-7
7.5 Controlling Peripheral Input/Output Devices 7-11
7.6 Special Devices 7-14
7.6.1 Keyboard Devices 7-14
7.6.2 Timer Devices 7-14
7.6.3 Cursor Devices 7-15
7.6.4 Ghost Devices 7-15
7.6.5 Window Manager Devices 7-15

8. Graphical Objects 8-1
8.1 Defining An Object 8-1
8.2 Using Objects 8-5
8.3 Object Editing 8-8
8.3.1 Identifying Display List Items with Tags 8-10

8.3.2 Inserting, Deleting, and Replacing within Objects 8-13
8.3.3 Example 8-15
8.3.4 Object Memory Management 8-16

9. Picking and Selecting 9-1
9.1 Mapping Screen Coordinates to World Coordinates 9-1
9.2 Picking 9-2
9.2.1 Using the Name Stack 9-6
9.2.2 Defining the Picking Region 9-9
9.2.3 Example 9-10
9.3 Selecting 9-14

10. Geometry Pipeline Feedback 10-1
10.1 The Geometry Pipeline 10-1
10.2 Feedback Mode 10-6

11. Curves and Surfaces 11-1
11.1 Curve Mathematics 11-2
11.1.1 Bezier Cubic Curve 11-3
11.1.2 Cardinal Spline Cubic Curve 11-4
11.1.3 B-Spline Cubic Curve 11-6
11.2 Drawing Curves 11-7
11.2.1 Rational Curves 11-21
11.3 Drawing Surfaces 11-23

12. Hidden Surfaces 12-1
12.1 Z-Buffer Mode 12-1
12.2 Backfacing Polygon Removal 12-6

13. Shading and Depth-Cueing 13-1
13.1 Shading 13-1
13.2 Depth-Cueing 13-6

14. Textports 14-1

USING MEX, THE IRIS WINDOW MANAGER

1. Getting Started with *mex* W-1
- 1.1 What Is a Window Manager? W-1
- 1.2 Window Manager Terminology W-2
- 1.3 What Does *mex* Do? W-3
- 1.4 Creating Windows W-3
- 1.5 Interacting with the Window Manager W-4
  - 1.5.1 Attaching to Windows W-5
  - 1.5.2 Selecting a Window W-5
  - 1.5.3 Moving a Window W-5
  - 1.5.4 Reshaping a Window W-6
  - 1.5.5 Pushing and Popping Windows W-6
  - 1.5.6 Removing a Window W-6
- 1.6 Selecting a Title Font W-7
- 1.7 Attaching to the Window Manager W-7
- 1.8 Executing Graphics Programs from *mex* W-8

2. Programming with *mex* W-9
- 2.1 Opening and Closing Windows W-9
- 2.2 Setting Window Constraints W-11
  - 2.2.1 Setting Constraints for Existing Windows W-16
- 2.3 Changing Windows Noninteractively (from within a Program) W-17
- 2.4 Other Window Routines W-19
- 2.5 Programming Hints W-24
  - 2.5.1 Graphics Initialization W-24
  - 2.5.2 Shared Facilities W-24
  - 2.5.3 Raster Fonts W-24
  - 2.5.4 The Event Queue W-25
- 2.6 Sample Program: Single Buffer Mode W-25
- 2.7 Sample Program: Double Buffer Mode W-27

3. Making Pop-up Menus W-31
- 3.1 Creating a Pop-up Menu W-32
- 3.2 Calling Up a Pop-up Menu W-34
- 3.3 Choosing Colors for Pop-up Menus W-36
- 3.4 Advanced Menu Formats W-38

3.4.1 Getting Back the Default Values for Menu Selections W-38
3.4.2 Changing the Return Values for Menu Selections W-39
3.4.3 Making a Title W-39
3.4.4 Binding a Function to a Whole Menu W-39
3.4.5 Binding a Function to a Menu Entry W-40
3.4.6 Making a Nested (Rollover) Menu W-40
3.5 An Example from *cedit*, a Color Editing Program W-41
3.6 Sample Program W-43

4. Customizing *mex* W-47
4.1 Interpreting Button Events W-48
4.2 Binding Colors to the Title Bar and Borders W-51
4.3 Binding Colors to Pop-up Menus and the Cursor W-53
4.4 Setting Color Map Entries W-53
4.5 A Sample *.mexrc* W-53
4.6 Arranging Your Desktop W-54

5. Controlling Multiple Windows from a Single Process W-57

SAMPLE CODE - EXAMPLES

1: Getting Started S-1

2: Common Drawing Commands S-5

3: Object Coordinate Systems S-15

4: Double Buffer Display Mode S-19

5: Input/Output Devices S-23

6: Interactive Drawing S-29

7: Pop-up Menus S-33

8: Modeling Transformations S-47

9: Writemasks and Color Maps S-61

10: A Color Editor S-67

SAMPLE CODE - WORKSHOPS

1: diamond1.c S-75

2: color2.c S-79

3: double3.c S-83

4: overlay4.c S-87

5: poll5.c S-91

6: aim6.c S-95

7: queue7.c S-99

8: menu8.c S-105

9: threed9.c S-111

10: coord10.c S-117

11: translate11.c S-123

12: composite12.c S-129

13: view13.c S-135

14: parabola14.c S-141

GLOSSARY GL-1

INDEX I-1

VOLUME II

REFERENCE MANUAL

Introduction
Permuted Index
Manual Pages

APPENDICES

A: Type Definitions for C and FORTRAN A-1
A.1 C Definitions A-1
A.2 FORTRAN Definitions A-19

B: Geometry Engine Computations B-1

C: Transformation Matrices C-1
C.1 Translation C-1
C.2 Scaling and Mirroring C-1
C.3 Rotation C-2
C.4 Viewing Transformations C-2
C.5 Perspective Transformations C-3
C.6 Orthographic Transformations C-4

D: Feedback Parser D-1

E: Window Manager Programs E-1
E.1 Color Tools E-1
E.2 Image Tools E-4
E.3 General Desktop Tools E-5
E.4 Utilities E-6
E.5 Device Usage Examples E-7
E.6 Fantasy Demonstrations E-8

F: IRIS Programming Tutorial Manual Pages F-1

G: Fast Immediate Mode and User-Defined Display Lists G-1
G.1 Introduction and Overview G-1
G.2 User-Defined Display Lists G-7
G.3 Example -- Defining Your Own Display List G-10
G.4 Counting Instructions G-15
G.5 Conclusions G-16

H: Using the Image Library H-1
H.1 Image Files H-1
H.2 Opening and Closing an Image File H-1
H.2.1 iopen - Open an Image File H-2
H.2.2 iclose - Close an Image File H-3
H.3 Reading from and Writing to Image Files H-3
H.3.1 putrow - Write a Row of Pixels from Buffer to Image File H-3
H.3.2 getrow - Read a Row of Pixels from Image File to Buffer H-4
H.4 Miscellaneous Functions H-4
H.4.1 isetname - Name an Image File H-4
H.4.2 isetcolormap - Interpret Pixel Values H-5
H.4.3 scrsave - Save Rectangular Region of Screen to Image File H-5
H.5 An Example H-6

NAME

introduction – description of routines in the Graphics Library

DESCRIPTION

The *Graphics Library Reference* contains descriptions of the routines in the Graphics Library.

Each description defines the number, order, and types of the arguments for each routine. C and FORTRAN are provided under the heading **SPECIFICATION**. A description of the routine, including function, effects, and potential errors is given in the section called **DESCRIPTION**; related routines and related material are listed under **SEE ALSO**.

Some of the routines have several versions, depending on the number and type of the arguments. Coordinate data can be 2D or 3D, and can be specified as floating point numbers, integers, or short (16-bit) integers. The default is 3-D floating point data. Integer data and 2-D points are specified with suffixes: *i* for integer, *s* for short, and *2* for 2D.

If the routine can be compiled into a display list, it is so stated in the **NOTE**.

Bugs are listed under **BUGS**.

TEXT STRINGS

C text strings are terminated with a null character (ASCII 0); FORTRAN has a **character** data type that includes the length of the string.

POINTERS

Many of the Graphics Library routines return several values to the caller. The arguments to these routines are *pointers*, or addresses of memory locations. In C, they are declared as pointer variables by prefixing the variable name with an asterisk in the declaration. FORTRAN passes all parameters by reference, so no special declaration is necessary.

BOOLEANS

Many of the routines have boolean arguments or return boolean values. These are declared as type *Boolean* in C, and as *logical* or *integer* in FORTRAN. We assume that FALSE is zero, and that TRUE is anything except FALSE.

TYPE DECLARATIONS

We have constructed type declarations for C wherever they add to the readability of the code. Here are the type definitions:

C

```
#define PATTERN_16 16
#define PATTERN_32 32
#define PATTERN_64 64
#define PATTERN_16_SIZE 16
#define PATTERN_32_SIZE 64
#define PATTERN_64_SIZE 256

typedef unsigned char Byte;
typedef long Boolean;
typedef short Angle;
typedef short Scoord;
typedef short Screencoord;
typedef long Icoord;
typedef float Coord;
typedef char *String;
typedef float Matrix[4][4];

typedef unsigned short Device;

typedef unsigned short Colorindex;
typedef unsigned char RGBvalue;

typedef unsigned short Linestyle;
typedef unsigned short Cursor[16];
typedef struct {
        unsigned short offset;      /* 2 bytes */
        Byte w,h;                   /* 2 bytes */
        char xoff,yoff;             /* 2 bytes */
        short width;                /* 2 bytes */
} Fontchar;
```

```
typedef long Object;
typedef long Tag;

typedef unsigned short Pattern16[PATTERN_16_SIZE];
typedef unsigned short Pattern32[PATTERN_32_SIZE];
typedef unsigned short Pattern64[PATTERN_64_SIZE];
```

LIST OF GRAPHICS LIBRARY ROUTINES

addtopup - adds items to an existing pop-up menu
arc - draws a circular arc
arcf - draws a filled circular arc
attachcursor - attaches the cursor to two valuators
backbuffer - enables updating in the back buffer
backface - turns backfacing polygon removal on and off
bbox2 - specifies bounding box and minimum pixel radius
blankscreen - turns screen refresh on and off
blanktime - sets the screen blanking timeout
blink - changes the color map entry at a selectable rate
blkqread - reads multiple entries from the queue
callfunc - calls a function from within an object
callobj - draws an instance of an object
charstr - draws a string of raster characters on the screen
chunksize - specifies minimum object size in memory
circ - outlines a circle
circf - draws a filled circle
clear - clears the viewport
clearhitcode - sets the system hitcode to zero
clkoff - turns off the keyboard click
clkon - turns on the keyboard click
closeobj - closes an object
cmov - updates the current character position
color - sets the color index in the current mode
compactify - compacts the memory storage of an object
crv - draws a curve

crvn - draws a series of curve segments

curorigin - sets the origin of a cursor

cursoff - turns off the cursor

curson - turns on the cursor

curvebasis - sets the basis matrix used to draw curves

curveit - draws a curve segment

curveprecision - sets the number of line segments that draw a curve segment

cyclemap - cycles through color maps at a specified rate

dbtext - sets the dial and button box text

defbasis - defines a basis matrix

defcursor - defines a cursor glyph

deflinestyle - defines a linestyle

defpattern - defines patterns

defpup - defines a menu

defrasterfont - defines a raster font

delobj - deletes an object

deltag - deletes tags from objects

depthcue - turns depth-cue mode on and off

devport - assigns a serial port to an external graphics device

dopup - displays the specified pop-up menu

doublebuffer - sets the display mode to double buffer mode

draw - draws a line

editobj - opens an object for editing

endfeedback - turns off feedback mode

endfullscrn - ends full-screen mode

endpick - turns off picking mode

endpupmode - ends pop-up mode

endselect - turns off selecting mode
feedback - turns on feedback mode
finish - blocks the host process until the Geometry Pipeline is empty
font - selects a raster font for drawing text strings
foreground - keeps a graphical process in the foreground
freepup - returns a menu and its data structures to the system
frontbuffer - enables updating in the front buffer
fudge - specifies fudge values that are added to a graphics window
fullscrn - gives a program the entire screen as a window
gbegin - initializes the system without altering the color map
gconfig - reconfigures the system
genobj - returns a unique integer for use as an object identifier
gentag - returns a unique integer for use as a tag
getbackface - returns whether backfacing polygons will appear
getbuffer - indicates which buffers are enabled for writing
getbutton - returns the state (up or down) of a button
getcmmode - returns the current color map mode
getcolor - returns the current color
getcpos - returns the current character position
getcursor - returns the cursor characteristics
getdcm - indicates whether depth-cue mode is on or off
getdepth - returns the parameters of setdepth
getdescender - returns the character characteristics
getdev - reads a list of valuators at one time
getdisplaymode - returns the current display mode
getfont - returns the current raster font number
getgpos - returns the current graphics position

getheight - returns the maximum character height in the current raster font

gethitcode - returns the current system hitcode

getlsbackup - returns the current value of the linestyle backup flag

getlsrepeat - returns the linestyle repeat count

getlstyle - returns the current linestyle

getlwidth - returns the current linewidth

getmap - returns the number of the current color map

getmatrix - returns the current transformation matrix

getmcolor - returns a color map entry

getmem - returns the amount of available memory

getmonitor - returns the current display monitor

getopenobj - returns the current open object

getorigin - returns the position of a graphics window

getothermonitor - returns the nondisplayed monitor type

getpattern - returns the index of the current pattern

getplanes - returns the number of available bitplanes

getport - creates a graphics window

getresetls - returns the current value of resetls

getscrmask - returns the current screenmask

getshade - returns the current shade

getsize - returns the size of a graphics window

gettp - returns the location of the current textport

getvaluator - returns the current state of a valuator

getviewport - returns the current viewport

getwritemask - returns the current writemask

getzbuffer - indicates whether z-buffering is on or off

gexit - terminates a program

gflush - forces all unsent commands down the network

ginit - initializes the system

greset - resets all global state attributes to their initial values

gRGBcolor - returns the current RGB value

gRGBcursor - returns the cursor characteristics in RGB mode

gRGBmask - returns the current RGB writemask

gselect - puts the system in selecting mode

gsync - waits for a vertical retrace period

imakebackground - registers the screen background process

initnames - initializes the name stack

ismex - returns TRUE if the window manager is running

isobj - indicates whether a given object number identifies an object

isqueued - indicates if the specified device is queued

istag - indicates if a given tag is used within the current open object

keepaspect - specifies the aspect ratio of a graphics window

lampoff - turns off the keyboard display lights

lampon - turns on the keyboard display lights

linewidth - specifies the linewidth

loadmatrix - loads a transformation matrix

loadname - loads the name on the top of the name stack

lookat - defines a viewing transformation

lsbackup - controls whether the last two pixels of a line are closed

lsrepeat - sets repeat factor for the current linestyle

makeobj - creates an object

maketag - numbers a routine in the display list

mapcolor - changes a color map entry

mapw - maps a point on the screen into a line in 3-D world coordinates

mapw2 - maps a point on the screen into 2-D world coordinates
maxsize - specifies a maximum size of a graphics window
minsize - specifies a minimum size of a graphics window
move - moves the current graphics position to a specified point
multimap - organizes the color map as 16 small maps
multmatrix - premultiplies the current transformation matrix
newpup - allocates and initializes a structure for a new menu
newtag - creates a new tag in an object
noise - filters valuator motion
noport - specifies that a program does not require a graphics window
objdelete - deletes routines from an object
objinsert - inserts routines in an object at a specified location
objreplace - overwrites existing display list routines with new ones
onemap - organizes the color map as one large map
ortho - defines an orthographic projection transformation
pagecolor - sets the color of the textport background
pagewritemask - sets the writemask for the textport background
passthrough - passes a single token through the Geometry Pipeline
patch - draws a surface patch
patchbasis - sets current patch basis matrices
patchcurves - sets the number of curves that represent a patch
patchprecision - sets the precision at which curves are drawn
pclos - polygon close
pdr - polygon draw
perspective - defines a perspective projection transformation
pick - puts the system in picking mode
picksize - sets the dimensions of the picking region

pmv - polygon move

pnt - draws a point

polarview - defines the viewer's position in polar coordinates

polf - draws a filled polygon

poly - outlines a polygon

popattributes - pops the attribute stack

popmatrix - pops the transformation matrix stack

popname - pops a name off the name stack

popviewport - restores the viewport, screenmask, and setdepth parameters

prefposition - specifies the preferred location and size of a graphics window

prefsize - specifies the preferred size of a graphics window

pupcolor - specifies the current pop-up drawing color

pupmode - provides access to the pop-up menu bitplanes

pushattributes - saves the global state attributes

pushmatrix - pushes down the transformation matrix stack

pushname - pushes a new name on the name stack

pushviewport - duplicates the current viewport

qdevice - queues a device (keyboard, button, or valuator)

qenter - creates an event queue entry

qread - reads the first entry in the event queue

qreset - empties the event queue

qtest - checks the contents of the event queue

rcrv - draws a curve

rcrvn - draws a series of curve segments

rdr - relative draw

readpixels - returns values of specific pixels

readRGB - returns values of specific pixels

rect - outlines a rectangular region

rectcopy - copies a rectangle of pixels on the screen

rectf - fills a rectangular area

resetls - controls the continuity of linestyles

reshapeviewport - sets the viewport to the current graphics window dimensions

RGBcolor - sets the current color in RGB mode

RGBcursor - sets the characteristics of the cursor in RGB mode

RGBmode - sets a display mode that bypasses the color map

RGBwritemask - grants write access to a subset of available bitplanes

ringbell - rings the keyboard bell

rmv - relative move

rot - rotates graphical primitives (floating point version)

rotate - rotates graphical primitives

rpatch - draws a rational surface patch

rpdr - relative polygon draw

rpmv - relative polygon move

scale - scales and mirrors objects

screenspace - interprets graphics positions as absolute screen coordinates

scrmask - defines a clipping mask for fine character clipping

setbell - sets the duration of the keyboard bell

setcursor - sets the cursor characteristics

setdblights - sets the lights on the dial and button box

setdepth - sets up a 3-D viewport

setfastcom - sends data in 8 bits per byte

setlinestyle - selects a linestyle pattern

setmap - selects one of the 16 small color maps

setmonitor - sets the monitor type

setpattern - selects a pattern for filling polygons, rectangles, and curves

setshade - sets the current polygon shade

setslowcom - sends data in 6 bits per byte

setvaluator - assigns an initial value to a valuator

shaderange - sets range of color indices used in depth-cueing

singlebuffer - writes and displays all the bitplanes

spclos - draws the current open shaded polygon

splf - draws a shaded filled polygon

stepunit - specifies that a graphics window change size in discrete steps

strwidth - returns the width of the specified text string

swapbuffers - exchanges the front and back buffers

swapinterval - defines a minimum time between buffer swaps

textcolor - sets the color of text drawn in the textport

textinit - initializes the console textport

textport - allocates an area of the screen for the textport

textwritemask - grants write permission for text drawn in the textport

tie - ties two valuators to a button

tpoff - turns off the textport

tpon - turns on the textport

translate - translates graphical primitives

unqdevice - disables the specified device from making entries in the event queue

viewport - allocates an area of the window for an image

winat - returns the identifier of the window beneath the cursor

winattach - attaches the input focus to the current graphics window and call process

winclose - closes the identified graphics window

winconstraints - changes the constraints of the current graphics window

window - defines a perspective projection transformation

winget - returns the identifier of the current graphics window

winmove - moves the lower-left corner of the current graphics window

winopen - creates a graphics window

winpop - places the current graphics window in front of all other windows

winposition - changes the size and position of the current graphics window

winpush - places the current window behind all other graphics windows

winset - sets the current graphics window

wintitle - adds a title bar to the current graphics window

writemask - grants write permission to available bitplanes

writepixels - paints a row of pixels on the screen

writeRGB - paints a row of pixels on the screen

xfpt - transforms points

zbuffer - starts or ends z-buffer mode

zclear - initializes the z-buffer

PERMUTED INDEX

setmap: selects one of the 16 small color maps. . . . . setmap
multimap: organizes the color map as 16 small maps. . . . . multimap
maps a point on the screen into 2-D world coordinates. mapw2: mapw2
setdepth: sets up a 3-D viewport. . . . . setdepth
a point on the screen into a line in 3-D world coordinates. mapw: maps . . . mapw
setslowcom: sends data in 6 bits per byte.. . . . . setslowcom
setfastcom: sends data in 8 bits per byte.. . . . . setfastcom
/interprets graphics positions as absolute screen coordinates. . . . . screenspace
RGBwritemask: grants write access to a subset of available/ . . . . . RGBwritemask
pupmode: provides access to the pop-up menu bitplanes. . . . pupmode
fudge: specifies fudge values added to a graphics window. . . . . fudge
graphics window. wintitle: adds a title bar to the current . . . . . wintitle
menu. addtopup: adds items to an existing pop-up . . . . addtopup
pop-up menu. addtopup: adds items to an existing . . . addtopup
the textport. textport: allocates an area of the screen for . . . . textport
an image. viewport: allocates an area of the window for . . . viewport
structure for a new menu. newpup: allocates and initializes a . . . . . newpup
initializes the system without altering the color map. gbegin: . . . . . gbegin
getmem: returns the amount of available memory. . . . . getmem
whether backfacing polygons will appear. getbackface: returns . . . . . getbackface
arc: draws a circular arc. . . . . arc
arcf: draws a filled circular arc. . . . . arcf
arc: draws a circular arc. . . . . arc
arcf: draws a filled circular arc. . . . . arcf
textport: allocates an area of the screen for the textport. . . . . textport
viewport: allocates an area of the window for an image. . . . . viewport
rectf: fills a rectangular area. . . . . rectf
multimap: organizes the color map as 16 small maps. . . . . multimap
returns a unique integer for use as a tag. gentag: . . . . . gentag
gives a program the entire screen as a window. fullscrn: . . . . . fullscrn
/interprets graphics positions as absolute screen coordinates. . . . . screenspace
returns a unique integer for use as an object identifier. genobj: . . . . . genobj
onemap: organizes the color map as one large map. . . . . onemap
keepaspect: specifies the aspect ratio of a graphics window. . . . . keepaspect
graphics device. devport: assigns a serial port to an external . . . . devport
valuator. setvaluator: assigns an initial value to a . . . . . setvaluator
blink: changes a color map entry at a selectable rate. . . . . blink
inserts routines in an object at a specified location. objinsert: . . . . objinsert
cyclemap: cycles through color maps at a specified rate. . . . . cyclemap
getdev: reads a list of valuators at one time. . . . . getdev
patchprecision: sets the precision at which curves are drawn. . . . . patchprecision
two valuators. attachcursor: attaches the cursor to . . . attachcursor
valuators. attachcursor: attaches the cursor to two . . . . . attachcursor
current graphics window/ winattach: attaches the input focus to the . . . . . winattach
popattributes: pops the attribute stack. . . . . popattributes
saves the global state attributes. pushattributes: . . . . . pushattributes
greset: resets all global state attributes to their initial values. . . . . greset
getplanes: returns the number of available bitplanes. . . . . getplanes
grants write access to a subset of available bitplanes. RGBwritemask: . . . RGBwritemask

Permuted Index

grants write permission to available bitplanes. writemask: . . . . . writemask
getmem: returns the amount of available memory. . . . . getmem
back buffer. backbuffer: enables updating in the . . . backbuffer
removal on and off. backface: turns backfacing polygon . . . backface
off. backface: turns backfacing polygon removal on and . . . backface
getbackface: returns whether backfacing polygons will appear. . . . . getbackface
sets the color of the textport background. pagecolor: . . . . . pagecolor
sets the writemask for the textport background. pagewritemask: . . . . . pagewritemask
/registers the screen background process. . . . . imakebackground
the current value of the linestyle backup flag. getlsbackup: returns . . . . getlsbackup
wintitle: adds a title bar to the current graphics window. . . . wintitle
patchbasis: sets current basis matrices. . . . . patchbasis
defbasis: defines a basis matrix. . . . . defbasis
curvebasis: selects a basis matrix used to draw curves. . . . . curvebasis
minimum pixel radius. bbox2: specifies bounding box and . . . bbox2
places the current graphics window behind all other windows. winpush: . . . winpush
ringbell: rings the keyboard bell. . . . . ringbell
sets the duration of the keyboard bell. setbell: . . . . . setbell
the indentifier of the window beneath the cursor. winat: returns . . . . winat
returns the number of available bitplanes. getplanes: . . . . . getplanes
provides access to the pop-up menu bitplanes. pupmode: . . . . . pupmode
access to a subset of available bitplanes. /grants write . . . . . RGBwritemask
writes and displays all bitplanes. singlebuffer: . . . . . singlebuffer
grants write permission to available bitplanes. writemask: . . . . . writemask
blanktime: sets the screen blanking timeout. . . . . blanktime
and off. blankscreen: turns screen refresh on . . . blankscreen
timeout. blanktime: sets the screen blanking . . . blanktime
a selectable rate. blink: changes a color map entry at . . . blink
from the queue. blkqread: reads multiple entries . . . . . blkqread
Geometry Pipeline is empty. finish: blocks the user process until the . . . . finish
radius. bbox2: specifies bounding box and minimum pixel bbox2
bbox2: specifies bounding box and minimum pixel radius. bbox2
the lights on the dial and button box. setdblights: sets . . . . . setdblights
dbtext: sets the dial and button box text. . . . . dbtext
enables updating in the back buffer. backbuffer: . . . . . backbuffer
enables updating in the front buffer. frontbuffer: . . . . . frontbuffer
sets the display mode to double buffer mode. doublebuffer: . . . . . doublebuffer
defines a minimum time between buffer swaps. swapinterval: . . . . . swapinterval
getbuffer: indicates which buffers are enabled for writing. . . . . getbuffer
exchanges the front and back buffers. swapbuffers: . . . . . swapbuffers
sets the lights on the dial and button box. setdblights: . . . . . setdblights
dbtext: sets the dial and button box text. . . . . dbtext
returns the state (up or down) of a button. getbutton: . . . . . getbutton
qdevice: queues a device (keyboard, button, or valuator). . . . . qdevice
tie: ties two valuators to a button. . . . . tie
RGBmode: sets a display mode that bypasses the color map. . . . . RGBmode
setfastcom: sends data in 8 bits per byte.. . . . . setfastcom
setslowcom: sends data in 6 bits per byte.. . . . . setslowcom
to the current graphics window and call process. /the input focus . . . . . winattach
within an object. callfunc: calls a function from . . . . . callfunc
object. callobj: draws an instance of an . . . . . callobj

object. callfunc: calls a function from within an . . . . . callfunc
/specifies that a graphics window change size in discrete steps. . . . . stepunit
selectable rate. blink: changes a color map entry at a . . . . . blink
mapcolor: changes a color map entry. . . . . mapcolor
current graphics/ winconstraints: changes the constraints of the . . . . . winconstraints
current graphics/ winposition: changes the size and position of the . . . winposition
getdescender: returns the character characteristics. . . . . getdescender
defines a clipping mask for fine character clipping. scrmask: . . . . . scrmask
getheight: returns the maximum character height in the current/ . . . . . getheight
cmov: updates the current character position. . . . . cmov
getcpos: returns the current character position. . . . . getcpos
getcursor: returns the cursor characteristics. . . . . getcursor
getdescender: returns the character characteristics. . . . . getdescender
gRGBcursor: returns the cursor characteristics in RGB mode. . . . . gRGBcursor
mode. RGBcursor: sets the characteristics of the cursor in RGB . . . RGBcursor
setcursor: sets the cursor characteristics. . . . . setcursor
charstr: draws a string of raster characters on the screen. . . . . charstr
characters on the screen. charstr: draws a string of raster . . . . . charstr
queue. qtest: checks the contents of the event . . . . . qtest
size in memory. chunksize: specifies minimum object . . chunksize
circ: outlines a circle. . . . . circ
circf: draws a filled circle. . . . . circf
circ: outlines a circle. . . . . circ
circf: draws a filled circle. . . . . circf
arc: draws a circular arc. . . . . arc
arcf: draws a filled circular arc. . . . . arcf
clear: clears the viewport. . . . . clear
hitcode to zero. clearhitcode: sets the system . . . . . clearhitcode
clear: clears the viewport. . . . . clear
clkoff: turns off the keyboard click. . . . . clkoff
clkon: turns on the keyboard click. . . . . clkon
clipping. scrmask: defines a clipping mask for fine character . . . . . scrmask
a clipping mask for fine character clipping. scrmask: defines . . . . . scrmask
click. clkoff: turns off the keyboard . . . . . clkoff
clkon: turns on the keyboard click. . . . clkon
pclos: polygon close. . . . . pclos
closeobj: closes an object. . . . . closeobj
closeobj: closes an object. . . . . closeobj
window. winclose: closes the identified graphics . . . . . winclose
position. cmov: updates the current character . . . cmov
getcolor: returns the current color. . . . . getcolor
RGBcolor: sets the current color in RGB mode. . . . . RGBcolor
color: sets the color index in the current mode. . . . . color
shaderange: sets range of color indices used in depth-cueing. . . . shaderange
multimap: organizes the color map as 16 small maps. . . . . multimap
onemap: organizes the color map as one large map. . . . . onemap
rate. blink: changes a color map entry at a selectable . . . . . blink
getmcolor: returns a color map entry. . . . . getmcolor
mapcolor: changes a color map entry. . . . . mapcolor
the system without altering the color map. gbegin: initializes . . . . . gbegin
returns the number of the current color map. getmap: . . . . . getmap

Permuted Index

getcmmode: returns the current color map mode. . . . . getcmmode
a display mode that bypasses the color map. RGBmode: sets . . . . . RGBmode
cyclemap: cycles through color maps at a specified rate. . . . . cyclemap
setmap: selects one of the 16 small color maps. . . . . setmap
textcolor: sets the color of text drawn in the textport. . . . . textcolor
pagecolor: sets the color of the textport background. . . . . pagecolor
specifies the current pop-up drawing color. pupcolor: . . . . . pupcolor
current mode. color: sets the color index in the . . . . color
the last two pixels of a line are colored. lsbackup: controls whether . . . lsbackup
storage of an object. compactify: compacts the memory . . . compactify
object. compactify: compacts the memory storage of an . . . compactify
textinit: initializes the console textport. . . . . textinit
window. winconstraints: changes the constraints of the current graphics . . . . winconstraints
qtest: checks the contents of the event queue. . . . . qtest
resetls: controls the continuity of linestyles. . . . . resetls
linestyles. resetls: controls the continuity of . . . . . resetls
of a line are colored. lsbackup: controls whether the last two pixels . . . lsbackup
a point on the screen into 2-D world coordinates. mapw2: maps mapw2
the screen into a line in 3-D world coordinates. mapw: maps a point on . . . mapw
the viewer's position in polar coordinates. polarview: defines . . . . . polarview
positions as absolute screen coordinates. /interprets graphics . . . . screenspace
screen. rectcopy: copies a rectangle of pixels on the . . . . rectcopy
winmove: moves the lower-left corner of the current graphics/ . . . . . winmove
returns the linestyle repeat count. getlsrepeat: . . . . . getlsrepeat
window manager. getport: creates a graphics window under the . . . getport
winopen: creates a graphics window. . . . . winopen
newtag: creates a new tag in an object. . . . . newtag
qenter: creates an event queue entry. . . . . qenter
makeobj: creates an object. . . . . makeobj
crv: draws a curve. . . . . crv
segments. crvn: draws a series of curve . . . . . crvn
cursor. curorigin: sets the origin of a . . . . . curorigin
patchbasis: sets current basis matrices. . . . . patchbasis
cmov: updates the current character position. . . . . cmov
getcpos: returns the current character position. . . . . getcpos
getcolor: returns the current color. . . . . getcolor
RGBcolor: sets the current color in RGB mode. . . . . RGBcolor
getmap: returns the number of the current color map. . . . . getmap
getcmmode: returns the current color map mode. . . . . getcmmode
getdisplaymode: returns the current display mode. . . . . getdisplaymode
getmonitor: returns the current display monitor. . . . . getmonitor
getgpos: returns the current graphics position. . . . . getgpos
specified point. move: moves the current graphics position to a . . . . . move
/attaches the input focus to the current graphics window and call/ . . . . winattach
other windows. winpush: places the current graphics window behind all . . . winpush
all other/ winpop: moves the current graphics window in front of . . . winpop
viewport to the dimensions of the current graphics window. /sets the . . . reshapeviewport
/changes the constraints of the current graphics window. . . . . winconstraints
returns the identifier of the current graphics window. winget: . . . . winget
moves the lower-left corner of the current graphics window. winmove: . . . winmove
/changes the size and position of the current graphics window. . . . . winposition

winset: sets the current graphics window. . . . . winset
wintitle: adds a title bar to the current graphics window. . . . . wintitle
getlstyle: returns the current linestyle. . . . . getlstyle
sets a repeat factor for the current linestyle. lsrepeat: . . . . . lsrepeat
getlwidth: returns the current linewidth. . . . . getlwidth
color: sets the color index in the current mode. . . . . color
getopenobj: returns the current open object. . . . . getopenobj
if a given tag is used within the current open object. /indicates . . . . . istag
spclos: draws the current open, shaded polygon. . . . . spclos
getpattern: returns the index of the current pattern. . . . . getpattern
setshade: sets the current polygon shade. . . . . setshade
pupcolor: specifies the current pop-up drawing color. . . . . pupcolor
the maximum character height in the current raster font. /returns . . . . . getheight
getfont: returns the current raster font number. . . . . getfont
gRGBcolor: returns the current RGB value. . . . . gRGBcolor
gRGBmask: returns the current RGB writemask. . . . . gRGBmask
getscrmask: returns the current screenmask. . . . . getscrmask
getshade: returns the current shade. . . . . getshade
getvaluator: returns the current state of a valuator. . . . . getvaluator
gethitcode: returns the current system hitcode. . . . . gethitcode
gettp: returns the location of the current textport. . . . . gettp
getmatrix: returns the current transformation matrix. . . . . getmatrix
multmatrix: premultiplies the current transformation matrix. . . . . multmatrix
getresetls: returns the current value of resetls. . . . . getresetls
backup/ getlsbackup: returns the current value of the linestyle . . . . . getlsbackup
getviewport: returns the current viewport. . . . . getviewport
pushviewport: duplicates the current viewport. . . . . pushviewport
getwritemask: returns the current writemask. . . . . getwritemask
cursoff: turns off the cursor. . . . . cursoff
curson: turns on the cursor. . . . . curson
getcursor: returns the cursor characteristics. . . . . getcursor
gRGBcursor: returns the cursor characteristics in RGB mode. . . . gRGBcursor
setcursor: sets the cursor characteristics. . . . . setcursor
curorigin: sets the origin of a cursor. . . . . curorigin
cursoff: turns off the cursor. . . . . cursoff
curson: turns on the cursor. . . . . curson
defcursor: defines a cursor glyph. . . . . defcursor
sets the characteristics of the cursor in RGB mode. RGBcursor: . . . RGBcursor
attachcursor: attaches the cursor to two valuators. . . . . attachcursor
of the window beneath the cursor. /returns the indentifier . . . . . winat
crv: draws a curve. . . . . crv
rcrv: draws a curve. . . . . rcrv
curveit: draws a curve segment. . . . . curveit
number of line segments that draw a curve segment. /sets the . . . . . curveprecision
crvn: draws a series of curve segments. . . . . crvn
rcrvn: draws a series of curve segments. . . . . rcrvn
used to draw curves. curvebasis: selects a basis matrix . . . . curvebasis
curveit: draws a curve segment. . . . . curveit
line segments that draw a curve/ curveprecision: sets the number of . . . . curveprecision
sets the precision at which curves are drawn. patchprecision: . . . . patchprecision
selects a basis matrix used to draw curves. curvebasis: . . . . . curvebasis

Permuted Index

filling polygons, rectangles, and curves. /selects a pattern for . . . . . setpattern
patchcurves: sets the number of curves that represent a patch. . . . . patchcurves
at a specified rate. cyclemap: cycles through color maps . . cyclemap
specified rate. cyclemap: cycles through color maps at a . . . . . cyclemap
setslowcom: sends data in 6 bits per byte.. . . . . setslowcom
setfastcom: sends data in 8 bits per byte.. . . . . setfastcom
freepup: returns a menu and its data structures to the system. . . . . freepup
text. dbtext: sets the dial and button box . . . dbtext
defbasis: defines a basis matrix. . . . . defbasis
defcursor: defines a cursor glyph. . . . . defcursor
transformation. ortho,: define an orthographic projection . . . . ortho
defbasis: defines a basis matrix. . . . . defbasis
character clipping. scrmask: defines a clipping mask for fine . . . . . scrmask
defcursor: defines a cursor glyph. . . . . defcursor
deflinestyle: defines a linestyle. . . . . deflinestyle
defpup: defines a menu. . . . . defpup
buffer swaps. swapinterval: defines a minimum time between . . . . swapinterval
transformation. perspective: defines a perspective projection . . . . . perspective
transformation. window: defines a perspective projection . . . . . window
defrasterfont: defines a raster font. . . . . defrasterfont
lookat: defines a viewing transformation. . . . . lookat
defpattern: defines patterns. . . . . defpattern
polar coordinates. polarview: defines the viewer's position in . . . . . polarview
deflinestyle: defines a linestyle. . . . . deflinestyle
defpattern: defines patterns. . . . . defpattern
defpup: defines a menu. . . . . defpup
font. defrasterfont: defines a raster . . . . . defrasterfont
delobj: deletes an object. . . . . delobj
objdelete: deletes routines from an object. . . . . objdelete
deltag: deletes tags from objects. . . . . deltag
delobj: deletes an object. . . . . delobj
deltag: deletes tags from objects. . . . . deltag
getdcm: indicates whether depth-cue mode is on or off. . . . . getdcm
depthcue: turns depth-cue mode on and off. . . . . depthcue
and off. depthcue: turns depth-cue mode on . . . depthcue
sets range of color indices used in depth-cueing. shaderange: . . . . . shaderange
Graphics Library. introduction: description of routines in the . . . . . intro
serial port to an external graphics device. devport: assigns a . . . . . devport
unqdevice: disables the specified device from making entries in the/ . . . . unqdevice
isqueued: indicates if the specified device is queued. . . . . isqueued
valuator). qdevice: queues a device (keyboard, button, or . . . . . qdevice
external graphics device. devport: assigns a serial port to an . . . . devport
setdblights: sets the lights on the dial and button box. . . . . setdblights
dbtext: sets the dial and button box text. . . . . dbtext
window. /sets the viewport to the dimensions of the current graphics . . . reshapeviewport
picksize: sets the dimensions of the picking region. . . . . picksize
making entries in the/ unqdevice: disables the specified device from . . . . unqdevice
a graphics window change size in discrete steps. /specifies that . . . . . stepunit
lampoff: turns off the keyboard display lights. . . . . lampoff
lampon: turns on the keyboard display lights. . . . . lampon
maketag: numbers a routine in the display list. . . . . maketag

objreplace: overwrites existing display list routines with new ones. . . . objreplace
getdisplaymode: returns the current display mode. . . . . getdisplaymode
map. RGBmode: sets a display mode that bypasses the color . . . RGBmode
doublebuffer: sets the display mode to double buffer mode. . . doublebuffer
getmonitor: returns the current display monitor. . . . . getmonitor
singlebuffer: writes and displays all bitplanes. . . . . singlebuffer
dopup: displays the specified pop-up menu. . . . dopup
noport: specifies that a program does not require a graphics window. . . . noport
menu. dopup: displays the specified pop-up . . . dopup
sets the display mode to double buffer mode. doublebuffer: . . . doublebuffer
to double buffer mode. doublebuffer: sets the display mode . . . doublebuffer
getbutton: returns the state (up or down) of a button. . . . . getbutton
gflush: forces all unsent routines down the network. . . . . gflush
stack. pushmatrix: pushes down the transformation matrix . . . . . pushmatrix
the number of line segments that draw a curve segment. /sets . . . . . curveprecision
selects a basis matrix used to draw curves. curvebasis: . . . . . curvebasis
draw: draws a line. . . . . draw
pdr: polygon draw. . . . . pdr
rdr: relative draw. . . . . rdr
rpdr: relative polygon draw. . . . . rpdr
specifies the current pop-up drawing color. pupcolor: . . . . . pupcolor
font: selects a raster font for drawing text strings. . . . . font
textcolor: sets the color of text drawn in the textport. . . . . textcolor
/grants write permission for text drawn in the textport. . . . . textwritemask
the precision at which curves are drawn. patchprecision: sets . . . . . patchprecision
arc: draws a circular arc. . . . . arc
crv: draws a curve. . . . . crv
rcrv: draws a curve. . . . . rcrv
curveit: draws a curve segment. . . . . curveit
circf: draws a filled circle. . . . . circf
arcf: draws a filled circular arc. . . . . arcf
polf: draws a filled polygon. . . . . polf
draw: draws a line. . . . . draw
pnt: draws a point. . . . . pnt
rpatch: draws a rational surface patch. . . . . rpatch
crvn: draws a series of curve segments. . . . . crvn
rcrvn: draws a series of curve segments. . . . . rcrvn
splf: draws a shaded filled polygon. . . . . splf
on the screen. charstr: draws a string of raster characters . . . . charstr
patch: draws a surface patch. . . . . patch
callobj: draws an instance of an object. . . . . callobj
polygon. spclos: draws the current open, shaded . . . . . spclos
pushviewport: duplicates the current viewport. . . . . pushviewport
setbell: sets the duration of the keyboard bell. . . . . setbell
editobj: opens an object for editing. . . . . editobj
editing. editobj: opens an object for . . . . . editobj
qreset: empties the event queue. . . . . qreset
until the Geometry Pipeline is empty. /blocks the user process . . . . . finish
indicates which buffers are enabled for writing. getbuffer: . . . . . getbuffer
backbuffer: enables updating in the back buffer. . . . backbuffer
buffer. frontbuffer: enables updating in the front . . . . . frontbuffer

Permuted Index

mode. endfeedback: turns off feedback . . . . endfeedback
endfullscrn: ends full-screen mode. . . . endfullscrn
endpick: turns off picking mode. . . . endpick
endpupmode: ends pop-up mode. . . . endpupmode
endfullscrn: ends full-screen mode. . . . . endfullscrn
endpupmode: ends pop-up mode. . . . . endpupmode
zbuffer: starts or ends z-buffer mode. . . . . zbuffer
endselect: turns off selecting mode. . . . endselect
fullscrn: gives a program the entire screen as a window. . . . . fullscrn
blkqread: reads multiple entries from the queue. . . . . blkqread
/the specified device from making entries in the event queue. . . . . unqdevice
blink: changes a color map entry at a selectable rate. . . . . blink
getmcolor: returns a color map entry. . . . . getmcolor
qread: reads the first entry in the event queue. . . . . qread
mapcolor: changes a color map entry. . . . . mapcolor
qenter: creates an event queue entry. . . . . qenter
qenter: creates an event queue entry. . . . . qenter
qread: reads the first entry in the event queue. . . . . qread
qreset: empties the event queue. . . . . qreset
qtest: checks the contents of the event queue. . . . . qtest
device from making entries in the event queue. /disables the specified . . . unqdevice
buffers. swapbuffers: exchanges the front and back . . . . . swapbuffers
new ones. objreplace: overwrites existing display list routines with . . . . objreplace
addtopup: adds items to an existing pop-up menu. . . . . addtopup
devport: assigns a serial port to an external graphics device. . . . . devport
lsrepeat: sets a repeat factor for the current linestyle. . . . . lsrepeat
endfeedback: turns off feedback mode. . . . . endfeedback
feedback: turns on feedback mode. . . . . feedback
feedback: turns on feedback mode. . . . feedback
circf: draws a filled circle. . . . . circf
arcf: draws a filled circular arc. . . . . arcf
polf: draws a filled polygon. . . . . polf
splf: draws a shaded filled polygon. . . . . splf
setpattern: selects a pattern for filling polygons, rectangles, and/ . . . . setpattern
rectf: fills a rectangular area. . . . . rectf
noise: filters valuator motion. . . . . noise
scrmask: defines a clipping mask for fine character clipping. . . . . scrmask
until the Geometry Pipeline is/ finish: blocks the user process . . . . . finish
qread: reads the first entry in the event queue. . . . . qread
value of the linestyle backup flag. /returns the current . . . . . getlsbackup
rot: rotates graphical primitives (floating point version). . . . . rot
and/ winattach: attaches the input focus to the current graphics window . . winattach
defrasterfont: defines a raster font. . . . . defrasterfont
font: selects a raster font for drawing text strings. . . . . font
height in the current raster font. /returns the maximum character . . getheight
getfont: returns the current raster font number. . . . . getfont
drawing text strings. font: selects a raster font for . . . . . font
network. gflush: forces all unsent routines down the . . . gflush
keeps a graphical process in the foreground. foreground: . . . . . foreground
process in the foreground. foreground: keeps a graphical . . . . . foreground
structures to the system. freepup: returns a menu and its data . . . freepup

swapbuffers: exchanges the front and back buffers. . . . . swapbuffers
frontbuffer: enables updating in the front buffer. . . . . frontbuffer
moves the current graphics window in front of all other windows. winpop: . . . winpop
front buffer. frontbuffer: enables updating in the . . . frontbuffer
to a graphics window. fudge: specifies fudge values added . . . fudge
window. fudge: specifies fudge values added to a graphics . . . . fudge
endfullscrn: ends full-screen mode. . . . . endfullscrn
screen as a window. fullscrn: gives a program the entire . . . fullscrn
callfunc: calls a function from within an object. . . . . callfunc
without altering the color map. gbegin: initializes the system . . . . . gbegin
gconfig: reconfigures the system. . . . . gconfig
use as an object identifier. genobj: returns a unique integer for . . . genobj
use as a tag. gentag: returns a unique integer for . . . gentag
blocks the user process until the Geometry Pipeline is empty. finish: . . . finish
passes a single token through the Geometry Pipeline. passthrough: . . . . passthrough
backfacing polygons will appear. getbackface: returns whether . . . . . getbackface
are enabled for writing. getbuffer: indicates which buffers . . . . getbuffer
down) of a button. getbutton: returns the state (up or . . . . getbutton
map mode. getcmmode: returns the current color . . getcmmode
getcolor: returns the current color. . . . . getcolor
character position. getcpos: returns the current . . . . . getcpos
characteristics. getcursor: returns the cursor . . . . . getcursor
mode is on or off. getdcm: indicates whether depth-cue . . . getdcm
setdepth. getdepth: returns the parameters of . . . getdepth
characteristics. getdescender: returns the character . . . getdescender
one time. getdev: reads a list of valuators at . . . . getdev
display mode. getdisplaymode: returns the current . . . getdisplaymode
font number. getfont: returns the current raster . . . . getfont
graphics position. getgpos: returns the current . . . . . getgpos
character height in the current/ getheight: returns the maximum . . . . getheight
system hitcode. gethitcode: returns the current . . . . . gethitcode
value of the linestyle backup flag. getlsbackup: returns the current . . . . . getlsbackup
repeat count. getlsrepeat: returns the linestyle . . . . . getlsrepeat
linestyle. getlstyle: returns the current . . . . . getlstyle
linewidth. getlwidth: returns the current . . . . . getlwidth
current color map. getmap: returns the number of the . . . . getmap
transformation matrix. getmatrix: returns the current . . . . . getmatrix
entry. getmcolor: returns a color map . . . . . getmcolor
available memory. getmem: returns the amount of . . . . . getmem
display monitor. getmonitor: returns the current . . . . . getmonitor
object. getopenobj: returns the current open . . . getopenobj
graphics window. getorigin: returns the position of a . . . . getorigin
nondisplayed monitor type. getothermonitor: returns the . . . . . getothermonitor
current pattern. getpattern: returns the index of the . . . getpattern
available bitplanes. getplanes: returns the number of . . . . getplanes
under the window manager. getport: creates a graphics window . . . getport
value of resetls. getresetls: returns the current . . . . . getresetls
screenmask. getscrmask: returns the current . . . . . getscrmask
getshade: returns the current shade. . . . getshade
graphics window. getsize: returns the size of a . . . . . getsize
current textport. gettp: returns the location of the . . . . gettp

Permuted Index

state of a valuator. getvaluator: returns the current . . . . . getvaluator
viewport. getviewport: returns the current . . . . . getviewport
writemask. getwritemask: returns the current . . . . getwritemask
z-buffering is on or off. getzbuffer: indicates whether . . . . . getzbuffer
gexit: terminates a program. . . . . gexit
down the network. gflush: forces all unsent routines . . . . gflush
ginit: initializes the system. . . . . ginit
object. isobj: indicates whether a given object number identifies an . . . . isobj
open object. istag: indicates if a given tag is used within the current . . . istag
a window. fullscrn: gives a program the entire screen as . . . fullscrn
pushattributes: saves the global state attributes. . . . . pushattributes
initial values. greset: resets all global state attributes to their . . . . . greset
defcursor: defines a cursor glyph. . . . . defcursor
available bitplanes. RGBwritemask: grants write access to a subset of . . . . RGBwritemask
drawn in the/ textwritemask: grants write permission for text . . . . . textwritemask
bitplanes. writemask: grants write permission to available . . . writemask
version). rot: rotates graphical primitives (floating point . . . rot
rotate: rotates graphical primitives. . . . . rotate
translate: translates graphical primitives. . . . . translate
foreground: keeps a graphical process in the foreground. . . . foreground
assigns a serial port to an external graphics device. devport: . . . . . devport
description of routines in the Graphics Library. introduction: . . . . . intro
getgpos: returns the current graphics position. . . . . getgpos
point. move: moves the current graphics position to a specified . . . . . move
screen/ screenspace: interprets graphics positions as absolute . . . . . screenspace
/the input focus to the current graphics window and call process. . . . . winattach
winpush: places the current graphics window behind all other/ . . . . winpush
discrete/ stepunit: specifies that a graphics window change size in . . . . . stepunit
specifies fudge values added to a graphics window. fudge: . . . . . fudge
getorigin: returns the position of a graphics window. . . . . getorigin
getsize: returns the size of a graphics window. . . . . getsize
other/ winpop: moves the current graphics window in front of all . . . . . winpop
specifies the aspect ratio of a graphics window. keepaspect: . . . . . keepaspect
specifies the maximum size of a graphics window. maxsize: . . . . . maxsize
specifies the minimum size of a graphics window. minsize: . . . . . minsize
that a program does not require a graphics window. noport: specifies . . . noport
the preferred location and size of a graphics window. /specifies . . . . . prefposition
specifies the preferred size of a graphics window. prefsize: . . . . . prefsize
to the dimensions of the current graphics window. /sets the viewport . . . reshapeviewport
manager. getport: creates a graphics window under the window . . . getport
winclose: closes the identified graphics window. . . . . winclose
the constraints of the current graphics window. /changes . . . . . winconstraints
the identifier of the current graphics window. winget: returns . . . . winget
the lower-left corner of the current graphics window. winmove: moves . . . winmove
winopen: creates a graphics window. . . . . winopen
the size and position of the current graphics window. /changes . . . . . winposition
winset: sets the current graphics window. . . . . winset
adds a title bar to the current graphics window. wintitle: . . . . . wintitle
attributes to their initial values. greset: resets all global state . . . . . greset
value. gRGBcolor: returns the current RGB . . gRGBcolor
characteristics in RGB mode. gRGBcursor: returns the cursor . . . . . gRGBcursor

writemask. gRGBmask: returns the current RGB . . gRGBmask
selecting mode. gselect: puts the system in . . . . . gselect
period. gsync: waits for a vertical retrace . . . . gsync
/returns the maximum character height in the current raster font. . . . . getheight
returns the current system hitcode. gethitcode: . . . . . gethitcode
clearhitcode: sets the system hitcode to zero. . . . . clearhitcode
winclose: closes the identified graphics window. . . . . winclose
unique integer for use as an object identifier. genobj: returns a . . . . . genobj
window. winget: returns the identifier of the current graphics . . . . winget
whether a given object number identifies an object. /indicates . . . . . isobj
current open/ istag: indicates if a given tag is used within the . . . . . istag
isqueued: indicates if the specified device is queued. . . . . isqueued
ismex: returns TRUE if the window manager is running. . . . ismex
an area of the window for an image. viewport: allocates . . . . . viewport
screen background process. imakebackground: registers the . . . . . imakebackground
the cursor. winat: returns the indentifier of the window beneath . . . . winat
color: sets the color index in the current mode. . . . . color
getpattern: returns the index of the current pattern. . . . . getpattern
within the current open/ istag: indicates if a given tag is used . . . . . istag
queued. isqueued: indicates if the specified device is . . . . isqueued
number identifies an object. isobj: indicates whether a given object . . . . isobj
on or off. getdcm: indicates whether depth-cue mode is . . . getdcm
or off. getzbuffer: indicates whether z-buffering is on . . . getzbuffer
for writing. getbuffer: indicates which buffers are enabled . . . getbuffer
shaderange: sets range of color indices used in depth-cueing. . . . . shaderange
setvaluator: assigns an initial value to a valuator. . . . . setvaluator
all global state attributes to their initial values. greset: resets . . . . . greset
menu. newpup: allocates and initializes a structure for a new . . . . . newpup
textinit: initializes the console textport. . . . . textinit
initnames: initializes the name stack. . . . . initnames
ginit: initializes the system. . . . . ginit
altering the color map. gbegin: initializes the system without . . . . . gbegin
zclear: initializes the z-buffer. . . . . zclear
stack. initnames: initializes the name . . . . . initnames
window and/ winattach: attaches the input focus to the current graphics . . . . winattach
specified location. objinsert: inserts routines in an object at a . . . . . objinsert
callobj: draws an instance of an object. . . . . callobj
gentag: returns a unique integer for use as a tag. . . . . gentag
genobj: returns a unique integer for use as an object/ . . . . . genobj
absolute screen/ screenspace: interprets graphics positions as . . . . . screenspace
routines in the Graphics Library. introduction: description of . . . . . intro
manager is running. ismex: returns TRUE if the window . . . ismex
object number identifies an object. isobj: indicates whether a given . . . . . isobj
device is queued. isqueued: indicates if the specified . . . isqueued
used within the current open/ istag: indicates if a given tag is . . . . . istag
addtopup: adds items to an existing pop-up menu. . . . . addtopup
ratio of a graphics window. keepaspect: specifies the aspect . . . . . keepaspect
foreground. foreground: keeps a graphical process in the . . . . . foreground
ringbell: rings the keyboard bell. . . . . ringbell
setbell: sets the duration of the keyboard bell. . . . . setbell
qdevice: queues a device (keyboard, button, or valuator). . . . . qdevice

clkoff: turns off the keyboard click. . . . . clkoff
clkon: turns on the keyboard click. . . . . clkon
lampoff: turns off the keyboard display lights. . . . . lampoff
lampon: turns on the keyboard display lights. . . . . lampon
display lights. lampoff: turns off the keyboard . . . . . lampoff
display lights. lampon: turns on the keyboard . . . . . lampon
of routines in the Graphics Library. introduction: description . . . . intro
turns off the keyboard display lights. lampoff: . . . . . lampoff
turns on the keyboard display lights. lampon: . . . . . lampon
setdblights: sets the lights on the dial and button box. . . . . setdblights
whether the last two pixels of a line are colored. /controls . . . . . lsbackup
draw: draws a line. . . . . draw
/maps a point on the screen into a line in 3-D world coordinates. . . . . mapw
curveprecision: sets the number of line segments that draw a curve/ . . . . curveprecision
returns the current value of the linestyle backup flag. getlsbackup: . . . getlsbackup
deflinestyle: defines a linestyle. . . . . deflinestyle
getlstyle: returns the current linestyle. . . . . getlstyle
sets a repeat factor for the current linestyle. lsrepeat: . . . . . lsrepeat
setlinestyle: selects a linestyle pattern. . . . . setlinestyle
getlsrepeat: returns the linestyle repeat count. . . . . getlsrepeat
resetls: controls the continuity of linestyles. . . . . resetls
getlwidth: returns the current linewidth. . . . . getlwidth
linewidth: specifies the linewidth. . . . . linewidth
linewidth: specifies the linewidth. . . . . linewidth
numbers a routine in the display list. maketag: . . . . . maketag
getdev: reads a list of valuators at one time. . . . . getdev
/overwrites existing display list routines with new ones. . . . . objreplace
matrix. loadmatrix: loads a transformation . . . loadmatrix
of the name stack. loadname: loads the name on the top . . . loadname
loadmatrix: loads a transformation matrix. . . . . loadmatrix
name stack. loadname: loads the name on the top of the . . . . loadname
window. /specifies the preferred location and size of a graphics . . . . . prefposition
routines in an object at a specified location. objinsert: inserts . . . . . objinsert
gettp: returns the location of the current textport. . . . . gettp
transformation. lookat: defines a viewing . . . . . lookat
graphics window. winmove: moves the lower-left corner of the current . . . . . winmove
two pixels of a line are colored. lsbackup: controls whether the last . . . lsbackup
the current linestyle. lsrepeat: sets a repeat factor for . . . . . lsrepeat
makeobj: creates an object. . . . . makeobj
display list. maketag: numbers a routine in the . . . . maketag
/disables the specified device from making entries in the event queue. . . . . unqdevice
a graphics window under the window manager. getport: creates . . . . . getport
ismex: returns TRUE if the window manager is running. . . . . ismex
multimap: organizes the color map as 16 small maps. . . . . multimap
onemap: organizes the color map as one large map. . . . . onemap
blink: changes a color map entry at a selectable rate. . . . . blink
getmcolor: returns a color map entry. . . . . getmcolor
mapcolor: changes a color map entry. . . . . mapcolor
system without altering the color map. gbegin: initializes the . . . . . gbegin
the number of the current color map. getmap: returns . . . . . getmap
getcmmode: returns the current color map mode. . . . . getcmmode

organizes the color map as one large map. onemap: . . . . . onemap
display mode that bypasses the color map. RGBmode: sets a . . . . . RGBmode
mapcolor: changes a color map entry. . . mapcolor
world coordinates. mapw2: maps a point on the screen into 2-D . . . mapw2
line in 3-D world/ mapw: maps a point on the screen into a . . . . mapw
cyclemap: cycles through color maps at a specified rate. . . . . cyclemap
organizes the color map as 16 small maps. multimap: . . . . . multimap
selects one of the 16 small color maps. setmap: . . . . . setmap
into a line in 3-D world/ mapw: maps a point on the screen . . . . mapw
into 2-D world coordinates. mapw2: maps a point on the screen . . . mapw2
scrmask: defines a clipping mask for fine character clipping. . . . . scrmask
patchbasis: sets current basis matrices. . . . . patchbasis
defbasis: defines a basis matrix. . . . . defbasis
returns the current transformation matrix. getmatrix: . . . . . getmatrix
loadmatrix: loads a transformation matrix. . . . . loadmatrix
the current transformation matrix. multmatrix: premultiplies . . . . multmatrix
popmatrix: pops the transformation matrix stack. . . . . popmatrix
pushes down the transformation matrix stack. pushmatrix: . . . . . pushmatrix
curvebasis: selects a basis matrix used to draw curves. . . . . curvebasis
current/ getheight: returns the maximum character height in the . . . . getheight
maxsize: specifies the maximum size of a graphics window. . . maxsize
of a graphics window. maxsize: specifies the maximum size . . maxsize
specifies minimum object size in memory. chunksize: . . . . . chunksize
returns the amount of available memory. getmem: . . . . . getmem
compactify: compacts the memory storage of an object. . . . . compactify
adds items to an existing pop-up menu. addtopup: . . . . . addtopup
system. freepup: returns a menu and its data structures to the . . . . freepup
provides access to the pop-up menu bitplanes. pupmode: . . . . . pupmode
defpup: defines a menu. . . . . defpup
dopup: displays the specified pop-up menu. . . . . dopup
initializes a structure for a new menu. newpup: allocates and . . . . . newpup
chunksize: specifies minimum object size in memory. . . . . chunksize
bbox2: specifies bounding box and minimum pixel radius. bbox2
minsize: specifies the minimum size of a graphics window. . . minsize
swapinterval: defines a minimum time between buffer swaps. . . swapinterval
of a graphics window. minsize: specifies the minimum size . . . minsize
scale: scales and mirrors objects. . . . . scale
sets the color index in the current mode. color: . . . . . color
the display mode to double buffer mode. doublebuffer: sets . . . . . doublebuffer
endfeedback: turns off feedback mode. . . . . endfeedback
endfullscrn: ends full-screen mode. . . . . endfullscrn
endpick: turns off picking mode. . . . . endpick
endpupmode: ends pop-up mode. . . . . endpupmode
endselect: turns off selecting mode. . . . . endselect
feedback: turns on feedback mode. . . . . feedback
returns the current color map mode. getcmmode: . . . . . getcmmode
returns the current display mode. getdisplaymode: . . . . . getdisplaymode
the cursor characteristics in RGB mode. gRGBcursor: returns . . . . . gRGBcursor
puts the system in selecting mode. gselect: . . . . . gselect
getdcm: indicates whether depth-cue mode is on or off. . . . . getdcm
depthcue: turns depth-cue mode on and off. . . . . depthcue

Permuted Index

pick: puts the system in picking mode. . . . . pick
sets the current color in RGB mode. RGBcolor: . . . . . RGBcolor
characteristics of the cursor in RGB mode. RGBcursor: sets the . . . . . RGBcursor
RGBmode: sets a display mode that bypasses the color map. . . . RGBmode
doublebuffer: sets the display mode to double buffer mode. . . . . doublebuffer
zbuffer: starts or ends z-buffer mode. . . . . zbuffer
returns the current display monitor. getmonitor: . . . . . getmonitor
returns the nondisplayed monitor type. getothermonitor: . . . . . getothermonitor
setmonitor: sets the monitor type. . . . . setmonitor
noise: filters valuator motion. . . . . noise
position to a specified point. move: moves the current graphics . . . . move
pmv: polygon move. . . . . pmv
rmv: relative move. . . . . rmv
rpmv: relative polygon move. . . . . rpmv
to a specified point. move: moves the current graphics position . . . move
front of all other windows. winpop: moves the current graphics window in . . winpop
current graphics window. winmove: moves the lower-left corner of the . . . . winmove
16 small maps. multimap: organizes the color map as . . multimap
blkqread: reads multiple entries from the queue. . . . . blkqread
current transformation matrix. multmatrix: premultiplies the . . . . . multmatrix
popname: pops a name off the name stack. . . . . popname
pushname: pushes a new name on the name stack. . . . . pushname
loadname: loads the name on the top of the name stack. . . . loadname
initnames: initializes the name stack. . . . . initnames
loads the name on the top of the name stack. loadname: . . . . . loadname
popname: pops a name off the name stack. . . . . popname
pushname: pushes a new name on the name stack. . . . . pushname
forces all unsent routines down the network. gflush: . . . . . gflush
and initializes a structure for a new menu. newpup: allocates . . . . . newpup
pushname: pushes a new name on the name stack. . . . . pushname
existing display list routines with new ones. objreplace: overwrites . . . . objreplace
newtag: creates a new tag in an object. . . . . newtag
structure for a new menu. newpup: allocates and initializes a . . . . newpup
object. newtag: creates a new tag in an . . . . . newtag
noise: filters valuator motion. . . . . noise
getothermonitor: returns the nondisplayed monitor type. . . . . getothermonitor
does not require a graphics window. noport: specifies that a program . . . . . noport
/specifies that a program does not require a graphics window. . . . . noport
returns the current raster font number. getfont: . . . . . getfont
indicates whether a given object number identifies an object. isobj: . . . . isobj
getplanes: returns the number of available bitplanes. . . . . getplanes
patch. patchcurves: sets the number of curves that represent a . . . . patchcurves
curve/ curveprecision: sets the number of line segments that draw a . . . curveprecision
getmap: returns the number of the current color map. . . . . getmap
list. maketag: numbers a routine in the display . . . . maketag
object. objdelete: deletes routines from an . . . objdelete
objinsert: inserts routines in an object at a specified location. . . . . objinsert
calls a function from within an object. callfunc: . . . . . callfunc
callobj: draws an instance of an object. . . . . callobj
closeobj: closes an object. . . . . closeobj
compacts the memory storage of an object. compactify: . . . . . compactify

delobj: deletes an	object.	delobj
editobj: opens an	object for editing.	editobj
getopenobj: returns the current open	object.	getopenobj
a unique integer for use as an	object identifier. genobj: returns	genobj
a given object number identifies an	object. isobj: indicates whether	isobj
tag is used within the current open	object. istag: indicates if a given	istag
makeobj: creates an	object.	makeobj
newtag: creates a new tag in an	object.	newtag
isobj: indicates whether a given	object number identifies an object. . . .	isobj
objdelete: deletes routines from an	object.	objdelete
chunksize: specifies minimum	object size in memory.	chunksize
deltag: deletes tags from	objects.	deltag
scale: scales and mirrors	objects.	scale
object at a specified location.	objinsert: inserts routines in an	objinsert
display list routines with new/	objreplace: overwrites existing	objreplace
onemap: organizes the color map as	one large map.	onemap
setmap: selects	one of the 16 small color maps.	setmap
getdev: reads a list of valuators at	one time.	getdev
one large map.	onemap: organizes the color map as . . .	onemap
display list routines with new	ones. /overwrites existing	objreplace
getopenobj: returns the current	open object.	getopenobj
given tag is used within the current	open object. istag: indicates if a	istag
spclos: draws the current	open, shaded polygon.	spclos
editobj:	opens an object for editing.	editobj
maps. multimap:	organizes the color map as 16 small . . .	multimap
map. onemap:	organizes the color map as one large . . .	onemap
curorigin: sets the	origin of a cursor.	curorigin
projection transformation.	ortho,: define an orthographic	ortho
transformation. ortho,: define an	orthographic projection	ortho
circ:	outlines a circle.	circ
poly:	outlines a polygon.	poly
rect:	outlines a rectangular region.	rect
routines with new ones. objreplace:	overwrites existing display list	objreplace
textport background.	pagecolor: sets the color of the	pagecolor
for the textport background.	pagewritemask: sets the writemask . . .	pagewritemask
screen. writepixels:	paints a row of pixels on the	writepixels
screen. writeRGB:	paints a row of pixels on the	writeRGB
getdepth: returns the	parameters of setdepth.	getdepth
viewport, screenmask, and setdepth	parameters. popviewport: restores	popviewport
Geometry Pipeline. passthrough:	passes a single token through the	passthrough
through the Geometry Pipeline.	passthrough: passes a single token	passthrough
	patch: draws a surface patch.	patch
patch: draws a surface	patch.	patch
number of curves that represent a	patch. patchcurves: sets the	patchcurves
rpatch: draws a rational surface	patch.	rpatch
matrices.	patchbasis: sets current basis	patchbasis
curves that represent a patch.	patchcurves: sets the number of	patchcurves
at which curves are drawn.	patchprecision: sets the precision	patchprecision
rectangles,/ setpattern: selects a	pattern for filling polygons, ...	setpattern
returns the index of the current	pattern. getpattern:	getpattern
setlinestyle: selects a linestyle	pattern.	setlinestyle

Permuted Index

defpattern: defines patterns. . . . . defpattern
pclos: polygon close. . . . . pclos
pdr: polygon draw. . . . . pdr
setfastcom: sends data in 8 bits per byte.. . . . . setfastcom
setslowcom: sends data in 6 bits per byte.. . . . . setslowcom
gsync: waits for a vertical retrace period. . . . . gsync
textwritemask: grants write permission for text drawn in the/ . . . . textwritemask
writemask: grants write permission to available bitplanes. . . . . writemask
projection transformation. perspective: defines a perspective . . . . perspective
perspective: defines a perspective projection/ . . . . . perspective
transformation. window: defines a perspective projection . . . . . window
mode. pick: puts the system in picking . . . . . pick
endpick: turns off picking mode. . . . . endpick
pick: puts the system in picking mode. . . . . pick
picksize: sets the dimensions of the picking region. . . . . picksize
picking region. picksize: sets the dimensions of the . . . picksize
the user process until the Geometry Pipeline is empty. finish: blocks . . . . finish
a single token through the Geometry Pipeline. passthrough: passes . . . . . passthrough
specifies bounding box and minimum pixel radius. bbox2: bbox2
/controls whether the last two pixels of a line are colored. . . . . lsbackup
rectcopy: copies a rectangle of pixels on the screen. . . . . rectcopy
writepixels: paints a row of pixels on the screen. . . . . writepixels
writeRGB: paints a row of pixels on the screen. . . . . writeRGB
returns values of specific pixels. readpixels: . . . . . readpixels
readRGB: returns values of specific pixels. . . . . readRGB
behind all other windows. winpush: places the current graphics window . . . winpush
pmv: polygon move. . . . . pmv
pnt: draws a point. . . . . pnt
graphics position to a specified point. move: moves the current . . . . . move
coordinates. mapw2: maps a point on the screen into 2-D world . . . mapw2
3-D world coordinates. mapw: maps a point on the screen into a line in . . . . mapw
pnt: draws a point. . . . . pnt
graphical primitives (floating point version). rot: rotates . . . . . rot
xfpt: transforms points. . . . . xfpt
defines the viewer's position in polar coordinates. polarview: . . . . . polarview
position in polar coordinates. polarview: defines the viewer's . . . . . polarview
polf: draws a filled polygon. . . . . polf
poly: outlines a polygon. . . . . poly
pclos: polygon close. . . . . pclos
pdr: polygon draw. . . . . pdr
rpdr: relative polygon draw. . . . . rpdr
pmv: polygon move. . . . . pmv
rpmv: relative polygon move. . . . . rpmv
polf: draws a filled polygon. . . . . polf
poly: outlines a polygon. . . . . poly
backface: turns backfacing polygon removal on and off. . . . . backface
setshade: sets the current polygon shade. . . . . setshade
draws the current open, shaded polygon. spclos: . . . . . spclos
splf: draws a shaded filled polygon. . . . . splf
/selects a pattern for filling polygons, rectangles, and curves. . . . . setpattern
returns whether backfacing polygons will appear. getbackface: . . . getbackface

stack. popattributes: pops the attribute . . . . . popattributes
matrix stack. popmatrix: pops the transformation . . . popmatrix
stack. popname: pops a name off the name . . . popname
popname: pops a name off the name stack. . . . . popname
popattributes: pops the attribute stack. . . . . popattributes
stack. popmatrix: pops the transformation matrix . . . . . popmatrix
pupcolor: specifies the current pop-up drawing color. . . . . pupcolor
addtopup: adds items to an existing pop-up menu. . . . . addtopup
pupmode: provides access to the pop-up menu bitplanes. . . . . pupmode
dopup: displays the specified pop-up menu. . . . . dopup
endpupmode: ends pop-up mode. . . . . endpupmode
screenmask, and setdepth/ popviewport: restores viewport, ... popviewport
devport: assigns a serial port to an external graphics device. . . . devport
cmov: updates the current character position. . . . . cmov
returns the current character position. getcpos: . . . . . getcpos
returns the current graphics position. getgpos: . . . . . getgpos
polarview: defines the viewer's position in polar coordinates. . . . . polarview
getorigin: returns the position of a graphics window. . . . . getorigin
winposition: changes the size and position of the current graphics/ . . . . winposition
move: moves the current graphics position to a specified point. . . . . move
screenspace: interprets graphics positions as absolute screen/ . . . . . screenspace
patchprecision: sets the precision at which curves are drawn. . . . patchprecision
prefposition: specifies the preferred location and size of a/ . . . . . prefposition
prefsize: specifies the preferred size of a graphics window. . . . prefsize
preferred location and size of a/ prefposition: specifies the . . . . . prefposition
size of a graphics window. prefsize: specifies the preferred . . . . . prefsize
transformation matrix. multmatrix: premultiplies the current . . . . . multmatrix
rot: rotates graphical primitives (floating point version). . . . rot
rotate: rotates graphical primitives. . . . . rotate
translate: translates graphical primitives. . . . . translate
registers the screen background process. imakebackground: . . . . . imakebackground
foreground: keeps a graphical process in the foreground. . . . . foreground
is empty. finish: blocks the user process until the Geometry Pipeline . . . finish
the current graphics window and call process. /the input focus to . . . . . winattach
window. noport: specifies that a program does not require a graphics . . . noport
gexit: terminates a program. . . . . gexit
window. fullscrn: gives a program the entire screen as a . . . . . fullscrn
ortho,: define an orthographic projection transformation. . . . . ortho
perspective: defines a perspective projection transformation. . . . . perspective
window: defines a perspective projection transformation. . . . . window
bitplanes. pupmode: provides access to the pop-up menu . . . pupmode
pop-up drawing color. pupcolor: specifies the current . . . . . pupcolor
pop-up menu bitplanes. pupmode: provides access to the . . . . pupmode
state attributes. pushattributes: saves the global . . . . . pushattributes
pushname: pushes a new name on the name stack. . . pushname
matrix stack. pushmatrix: pushes down the transformation . . . . pushmatrix
transformation matrix stack. pushmatrix: pushes down the . . . . . pushmatrix
name stack. pushname: pushes a new name on the . . pushname
viewport. pushviewport: duplicates the current . . . pushviewport
pick: puts the system in picking mode. . . . . pick
gselect: puts the system in selecting mode. . . . . gselect

Permuted Index

button, or valuator). qdevice: queues a device (keyboard, . . . qdevice
entry. qenter: creates an event queue . . . . . qenter
event queue. qread: reads the first entry in the . . . . qread
qreset: empties the event queue. . . . . qreset
event queue. qtest: checks the contents of the . . . . . qtest
reads multiple entries from the queue. blkqread: . . . . . blkqread
qenter: creates an event queue entry. . . . . qenter
reads the first entry in the event queue. qread: . . . . . qread
qreset: empties the event queue. . . . . qreset
checks the contents of the event queue. qtest: . . . . . qtest
from making entries in the event queue. /the specified device . . . . . unqdevice
indicates if the specified device is queued. isqueued: . . . . . isqueued
or valuator). qdevice: queues a device (keyboard, button, . . . qdevice
bounding box and minimum pixel radius. bbox2: specifies bbox2
depth-cueing. shaderange: sets range of color indices used in . . . . . shaderange
charstr: draws a string of raster characters on the screen. . . . . charstr
defrasterfont: defines a raster font. . . . . defrasterfont
strings. font: selects a raster font for drawing text . . . . . font
character height in the current raster font. /returns the maximum . . . . getheight
getfont: returns the current raster font number. . . . . getfont
a color map entry at a selectable rate. blink: changes . . . . . blink
through color maps at a specified rate. cyclemap: cycles . . . . . cyclemap
keepaspect: specifies the aspect ratio of a graphics window. . . . . keepaspect
rpatch: draws a rational surface patch. . . . . rpatch
rcrv: draws a curve. . . . . rcrv
segments. rcrvn: draws a series of curve . . . . . rcrvn
rdr: relative draw. . . . . rdr
specific pixels. readpixels: returns values of . . . . . readpixels
pixels. readRGB: returns values of specific . . . readRGB
time. getdev: reads a list of valuators at one . . . . . getdev
queue. blkqread: reads multiple entries from the . . . . . blkqread
queue. qread: reads the first entry in the event . . . . . qread
gconfig: reconfigures the system. . . . . gconfig
rect: outlines a rectangular region. . . . . rect
rectcopy: copies a rectangle of pixels on the screen. . . . rectcopy
a pattern for filling polygons, rectangles, and curves. /selects . . . . . setpattern
rectf: fills a rectangular area. . . . . rectf
rect: outlines a rectangular region. . . . . rect
pixels on the screen. rectcopy: copies a rectangle of . . . . . rectcopy
rectf: fills a rectangular area. . . . . rectf
blankscreen: turns screen refresh on and off. . . . . blankscreen
sets the dimensions of the picking region. picksize: . . . . . picksize
rect: outlines a rectangular region. . . . . rect
process. imakebackground: registers the screen background . . . . . imakebackground
rdr: relative draw. . . . . rdr
rmv: relative move. . . . . rmv
rpdr: relative polygon draw. . . . . rpdr
rpmv: relative polygon move. . . . . rpmv
backface: turns backfacing polygon removal on and off. . . . . backface
getlsrepeat: returns the linestyle repeat count. . . . . getlsrepeat
linestyle. lsrepeat: sets a repeat factor for the current . . . . . lsrepeat

sets the number of curves that represent a patch. patchcurves: . . . . . patchcurves
specifies that a program does not require a graphics window. noport: . . . noport
linestyles. resetls: controls the continuity of . . . . resetls
returns the current value of resetls. getresetls: . . . . . getresetls
to their initial values. greset: resets all global state attributes . . . . . greset
to the dimensions of the current/ reshapeviewport: sets the viewport . . . reshapeviewport
setdepth parameters. popviewport: restores viewport, screenmask, and . . . popviewport
gsync: waits for a vertical retrace period. . . . . gsync
getmcolor: returns a color map entry. . . . . getmcolor
structures to the system. freepup: returns a menu and its data . . . . . freepup
a tag. gentag: returns a unique integer for use as . . . . gentag
an object identifier. genobj: returns a unique integer for use as . . . . genobj
memory. getmem: returns the amount of available . . . . . getmem
characteristics. getdescender: returns the character . . . . . getdescender
position. getcpos: returns the current character . . . . . getcpos
getcolor: returns the current color. . . . . getcolor
getcmmode: returns the current color map mode. . . . getcmmode
getdisplaymode: returns the current display mode. . . . getdisplaymode
getmonitor: returns the current display monitor. . . . getmonitor
position. getgpos: returns the current graphics . . . . . getgpos
getlstyle: returns the current linestyle. . . . . getlstyle
getlwidth: returns the current linewidth. . . . . getlwidth
getopenobj: returns the current open object. . . . . getopenobj
number. getfont: returns the current raster font . . . . . getfont
gRGBcolor: returns the current RGB value. . . . . gRGBcolor
gRGBmask: returns the current RGB writemask. . . . gRGBmask
getscrmask: returns the current screenmask. . . . . getscrmask
getshade: returns the current shade. . . . . getshade
valuator. getvaluator: returns the current state of a . . . . . getvaluator
gethitcode: returns the current system hitcode. . . . gethitcode
matrix. getmatrix: returns the current transformation . . . . getmatrix
resetls. getresetls: returns the current value of . . . . . getresetls
linestyle backup flag. getlsbackup: returns the current value of the . . . . . getlsbackup
getviewport: returns the current viewport. . . . . getviewport
getwritemask: returns the current writemask. . . . . getwritemask
getcursor: returns the cursor characteristics. . . . . getcursor
in RGB mode. gRGBcursor: returns the cursor characteristics . . . . gRGBcursor
current graphics window. winget: returns the identifier of the . . . . . winget
window beneath the cursor. winat: returns the indentifier of the . . . . . winat
pattern. getpattern: returns the index of the current . . . . . getpattern
getlsrepeat: returns the linestyle repeat count. . . . . getlsrepeat
textport. gettp: returns the location of the current . . . . gettp
in the current raster/ getheight: returns the maximum character height . . getheight
type. getothermonitor: returns the nondisplayed monitor . . . . getothermonitor
bitplanes. getplanes: returns the number of available . . . . . getplanes
color map. getmap: returns the number of the current . . . . getmap
getdepth: returns the parameters of setdepth. . . . getdepth
window. getorigin: returns the position of a graphics . . . . getorigin
window. getsize: returns the size of a graphics . . . . . getsize
button. getbutton: returns the state (up or down) of a . . . . getbutton
text string. strwidth: returns the width of the specified . . . . strwidth

Permuted Index

is running. ismex:	returns TRUE if the window manager . .	ismex
readpixels:	returns values of specific pixels.	readpixels
readRGB:	returns values of specific pixels.	readRGB
will appear. getbackface:	returns whether backfacing polygons . .	getbackface
the cursor characteristics in	RGB mode. gRGBcursor: returns	gRGBcursor
RGBcolor: sets the current color in	RGB mode.	RGBcolor
the characteristics of the cursor in	RGB mode. RGBcursor: sets	RGBcursor
gRGBcolor: returns the current	RGB value.	gRGBcolor
gRGBmask: returns the current	RGB writemask.	gRGBmask
RGB mode.	RGBcolor: sets the current color in . . .	RGBcolor
of the cursor in RGB mode.	RGBcursor: sets the characteristics . . .	RGBcursor
bypasses the color map.	RGBmode: sets a display mode that . . .	RGBmode
a subset of available bitplanes.	RGBwritemask: grants write access to . .	RGBwritemask
	ringbell: rings the keyboard bell.	ringbell
ringbell:	rings the keyboard bell.	ringbell
	rmv: relative move.	rmv
(floating point version).	rot: rotates graphical primitives	rot
primitives.	rotate: rotates graphical	rotate
(floating point version). rot:	rotates graphical primitives	rot
rotate:	rotates graphical primitives.	rotate
maketag: numbers a	routine in the display list.	maketag
gflush: forces all unsent	routines down the network.	gflush
objdelete: deletes	routines from an object.	objdelete
location. objinsert: inserts	routines in an object at a specified	objinsert
introduction: description of	routines in the Graphics Library.	intro
overwrites existing display list	routines with new ones. objreplace: . . .	objreplace
writepixels: paints a	row of pixels on the screen.	writepixels
writeRGB: paints a	row of pixels on the screen.	writeRGB
patch.	rpatch: draws a rational surface	rpatch
	rpdr: relative polygon draw.	rpdr
	rpmv: relative polygon move.	rpmv
TRUE if the window manager is	running. ismex: returns	ismex
pushattributes:	saves the global state attributes.	pushattributes
	scale: scales and mirrors objects.	scale
scale:	scales and mirrors objects.	scale
fullscrn: gives a program the entire	screen as a window.	fullscrn
imakebackground: registers the	screen background process.	imakebackground
blanktime: sets the	screen blanking timeout.	blanktime
a string of raster characters on the	screen. charstr: draws	charstr
graphics positions as absolute	screen coordinates. /interprets	screenspace
textport: allocates an area of the	screen for the textport.	textport
mapw2: maps a point on the	screen into 2-D world coordinates. . . .	mapw2
mapw: maps a point on the	screen into a line in 3-D world/	mapw
copies a rectangle of pixels on the	screen. rectcopy:	rectcopy
blankscreen: turns	screen refresh on and off.	blankscreen
paints a row of pixels on the	screen. writepixels:	writepixels
paints a row of pixels on the	screen. writeRGB:	writeRGB
popviewport: restores viewport,	screenmask, and setdepth parameters. . .	popviewport
getscrmask: returns the current	screenmask.	getscrmask
positions as absolute screen/	screenspace: interprets graphics	screenspace
fine character clipping.	scrmask: defines a clipping mask for . . .	scrmask

curveit: draws a curve segment. . . . . curveit
of line segments that draw a curve segment. /sets the number . . . . . curveprecision
crvn: draws a series of curve segments. . . . . crvn
rcrvn: draws a series of curve segments. . . . . rcrvn
/sets the number of line segments that draw a curve segment. . . . curveprecision
changes a color map entry at a selectable rate. blink: . . . . . blink
endselect: turns off selecting mode. . . . . endselect
gselect: puts the system in selecting mode. . . . . gselect
curves. curvebasis: selects a basis matrix used to draw . . . curvebasis
setlinestyle: selects a linestyle pattern. . . . . setlinestyle
polygons, rectangles,/ setpattern: selects a pattern for filling . . . . . setpattern
text strings. font: selects a raster font for drawing . . . . . font
maps. setmap: selects one of the 16 small color . . . . setmap
setslowcom: sends data in 6 bits per byte.. . . . . setslowcom
setfastcom: sends data in 8 bits per byte.. . . . . setfastcom
device. devport: assigns a serial port to an external graphics . . . . devport
crvn: draws a series of curve segments. . . . . crvn
rcrvn: draws a series of curve segments. . . . . rcrvn
keyboard bell. setbell: sets the duration of the . . . . . setbell
characteristics. setcursor: sets the cursor . . . . . setcursor
dial and button box. setdblights: sets the lights on the . . . . setdblights
getdepth: returns the parameters of setdepth. . . . . getdepth
restores viewport, screenmask, and setdepth parameters. popviewport: . . . popviewport
setdepth: sets up a 3-D viewport. . . . . setdepth
byte.. setfastcom: sends data in 8 bits per . . . setfastcom
pattern. setlinestyle: selects a linestyle . . . . . setlinestyle
color maps. setmap: selects one of the 16 small . . . setmap
setmonitor: sets the monitor type. . . . . setmonitor
filling polygons, rectangles, and/ setpattern: selects a pattern for . . . . . setpattern
the color map. RGBmode: sets a display mode that bypasses . . . . RGBmode
linestyle. lsrepeat: sets a repeat factor for the current . . . . lsrepeat
patchbasis: sets current basis matrices. . . . . patchbasis
depth-cueing. shaderange: sets range of color indices used in . . . . shaderange
cursor in RGB mode. RGBcursor: sets the characteristics of the . . . . . RGBcursor
mode. color: sets the color index in the current . . . . color
textport. textcolor: sets the color of text drawn in the . . . . textcolor
background. pagecolor: sets the color of the textport . . . . . pagecolor
RGBcolor: sets the current color in RGB mode. . . . RGBcolor
winset: sets the current graphics window. . . . . winset
setshade: sets the current polygon shade. . . . . setshade
setcursor: sets the cursor characteristics. . . . . setcursor
dbtext: sets the dial and button box text. . . . . dbtext
region. picksize: sets the dimensions of the picking . . . . picksize
buffer mode. doublebuffer: sets the display mode to double . . . . . doublebuffer
bell. setbell: sets the duration of the keyboard . . . . setbell
button box. setdblights: sets the lights on the dial and . . . . . setdblights
setmonitor: sets the monitor type. . . . . setmonitor
represent a patch. patchcurves: sets the number of curves that . . . . . patchcurves
that draw a curve/ curveprecision: sets the number of line segments . . . . curveprecision
curorigin: sets the origin of a cursor. . . . . curorigin
are drawn. patchprecision: sets the precision at which curves . . . . patchprecision

Permuted Index

blanktime: sets the screen blanking timeout. . . . . blanktime
clearhitcode: sets the system hitcode to zero. . . . . clearhitcode
of the current/ reshapeviewport: sets the viewport to the dimensions . . . reshapeviewport
background. pagewritemask: sets the writemask for the textport . . . . pagewritemask
setdepth: sets up a 3-D viewport. . . . . setdepth
shade. setshade: sets the current polygon . . . . setshade
byte.. setslowcom: sends data in 6 bits per . . . setslowcom
value to a valuator. setvaluator: assigns an initial . . . . . setvaluator
getshade: returns the current shade. . . . . getshade
setshade: sets the current polygon shade. . . . . setshade
splf: draws a shaded filled polygon. . . . . splf
spclos: draws the current open, shaded polygon. . . . . spclos
indices used in depth-cueing. shaderange: sets range of color . . . . . shaderange
Pipeline. passthrough: passes a single token through the Geometry . . . passthrough
all bitplanes. singlebuffer: writes and displays . . . . singlebuffer
graphics/ winposition: changes the size and position of the current . . . . . winposition
that a graphics window change size in discrete steps. /specifies . . . . . stepunit
chunksize: specifies minimum object size in memory. . . . . chunksize
getsize: returns the size of a graphics window. . . . . getsize
maxsize: specifies the maximum size of a graphics window. . . . . maxsize
minsize: specifies the minimum size of a graphics window. . . . . minsize
/specifies the preferred location and size of a graphics window. . . . . prefposition
prefsize: specifies the preferred size of a graphics window. . . . . prefsize
setmap: selects one of the 16 small color maps. . . . . setmap
organizes the color map as 16 small maps. multimap: . . . . . multimap
shaded polygon. spclos: draws the current open, ... spclos
readpixels: returns values of specific pixels. . . . . readpixels
readRGB: returns values of specific pixels. . . . . readRGB
in the/ unqdevice: disables the specified device from making entries . . . unqdevice
isqueued: indicates if the specified device is queued. . . . . isqueued
inserts routines in an object at a specified location. objinsert: . . . . . objinsert
the current graphics position to a specified point. move: moves . . . . . move
dopup: displays the specified pop-up menu. . . . . dopup
cycles through color maps at a specified rate. cyclemap: . . . . . cyclemap
strwidth: returns the width of the specified text string. . . . . strwidth
pixel radius. bbox2: specifies bounding box and minimum . . bbox2
graphics window. fudge: specifies fudge values added to a . . . . fudge
memory. chunksize: specifies minimum object size in . . . . chunksize
change size in discrete/ stepunit: specifies that a graphics window . . . . stepunit
require a graphics window. noport: specifies that a program does not . . . . noport
graphics window. keepaspect: specifies the aspect ratio of a . . . . . keepaspect
color. pupcolor: specifies the current pop-up drawing . . . pupcolor
linewidth: specifies the linewidth. . . . . linewidth
graphics window. maxsize: specifies the maximum size of a . . . . maxsize
graphics window. minsize: specifies the minimum size of a . . . . . minsize
size of a graphics/ prefposition: specifies the preferred location and . . . prefposition
graphics window. prefsize: specifies the preferred size of a . . . . . prefsize
splf: draws a shaded filled polygon. . . . splf
initnames: initializes the name stack. . . . . initnames
the name on the top of the name stack. loadname: loads . . . . . loadname
popattributes: pops the attribute stack. . . . . popattributes

pops the transformation matrix stack. popmatrix: . . . . . popmatrix
popname: pops a name off the name stack. . . . . popname
down the transformation matrix stack. pushmatrix: pushes . . . . . pushmatrix
pushes a new name on the name stack. pushname: . . . . . pushname
zbuffer: starts or ends z-buffer mode. . . . . zbuffer
pushattributes: saves the global state attributes. . . . . pushattributes
values. greset: resets all global state attributes to their initial . . . . . greset
getvaluator: returns the current state of a valuator. . . . . getvaluator
getbutton: returns the state (up or down) of a button. . . . . getbutton
window change size in discrete steps. /specifies that a graphics . . . . . stepunit
window change size in discrete/ stepunit: specifies that a graphics . . . . stepunit
compactify: compacts the memory storage of an object. . . . . compactify
screen. charstr: draws a string of raster characters on the . . . . charstr
the width of the specified text string. strwidth: returns . . . . . strwidth
a raster font for drawing text strings. font: selects . . . . . font
newpup: allocates and initializes a structure for a new menu. . . . . newpup
freepup: returns a menu and its data structures to the system. . . . . freepup
specified text string. strwidth: returns the width of the . . . . strwidth
/grants write access to a subset of available bitplanes. . . . . RGBwritemask
patch: draws a surface patch. . . . . patch
rpatch: draws a rational surface patch. . . . . rpatch
back buffers. swapbuffers: exchanges the front and . . swapbuffers
between buffer swaps. swapinterval: defines a minimum time . . swapinterval
a minimum time between buffer swaps. swapinterval: defines . . . . . swapinterval
menu and its data structures to the system. freepup: returns a . . . . . freepup
gconfig: reconfigures the system. . . . . gconfig
ginit: initializes the system. . . . . ginit
gethitcode: returns the current system hitcode. . . . . gethitcode
clearhitcode: sets the system hitcode to zero. . . . . clearhitcode
pick: puts the system in picking mode. . . . . pick
gselect: puts the system in selecting mode. . . . . gselect
map. gbegin: initializes the system without altering the color . . . . gbegin
a unique integer for use as a tag. gentag: returns . . . . . gentag
newtag: creates a new tag in an object. . . . . newtag
object. istag: indicates if a given tag is used within the current open . . . istag
deltag: deletes tags from objects. . . . . deltag
gexit: terminates a program. . . . . gexit
dbtext: sets the dial and button box text. . . . . dbtext
textcolor: sets the color of text drawn in the textport. . . . . textcolor
/grants write permission for text drawn in the textport. . . . . textwritemask
returns the width of the specified text string. strwidth: . . . . . strwidth
selects a raster font for drawing text strings. font: . . . . . font
drawn in the textport. textcolor: sets the color of text . . . . . textcolor
textport. textinit: initializes the console . . . . . textinit
screen for the textport. textport: allocates an area of the . . . . textport
pagecolor: sets the color of the textport background. . . . . pagecolor
sets the writemask for the textport background. pagewritemask: . . pagewritemask
returns the location of the current textport. gettp: . . . . . gettp
sets the color of text drawn in the textport. textcolor: . . . . . textcolor
textinit: initializes the console textport. . . . . textinit
an area of the screen for the textport. textport: allocates . . . . . textport

Permuted Index

permission for text drawn in the	textport. /grants write	textwritemask
tpoff: turns off the	textport.	tpoff
tpon: turns on the	textport.	tpon
permission for text drawn in the/	textwritemask: grants write	textwritemask
all global state attributes to	their initial values. /resets	greset
rate. cyclemap: cycles	through color maps at a specified	cyclemap
passthrough: passes a single token	through the Geometry Pipeline.	passthrough
	tie: ties two valuators to a button.	tie
tie:	ties two valuators to a button.	tie
swapinterval: defines a minimum	time between buffer swaps.	swapinterval
reads a list of valuators at one	time. getdev:	getdev
blanktime: sets the screen blanking	timeout.	blanktime
window. wintitle: adds a	title bar to the current graphics	wintitle
passthrough: passes a single	token through the Geometry Pipeline. . .	passthrough
loadname: loads the name on the	top of the name stack.	loadname
	tpoff: turns off the textport.	tpoff
	tpon: turns on the textport.	tpon
lookat: defines a viewing	transformation.	lookat
getmatrix: returns the current	transformation matrix.	getmatrix
loadmatrix: loads a	transformation matrix.	loadmatrix
premultiplies the current	transformation matrix. multmatrix: . . .	multmatrix
popmatrix: pops the	transformation matrix stack.	popmatrix
pushmatrix: pushes down the	transformation matrix stack.	pushmatrix
define an orthographic projection	transformation. ortho,:	ortho
defines a perspective projection	transformation. perspective:	perspective
defines a perspective projection	transformation. window:	window
xfpt:	transforms points.	xfpt
primitives.	translate: translates graphical	translate
translate:	translates graphical primitives.	translate
running. ismex: returns	TRUE if the window manager is	ismex
and off. backface:	turns backfacing polygon removal on . .	backface
depthcue:	turns depth-cue mode on and off. . . .	depthcue
endfeedback:	turns off feedback mode.	endfeedback
endpick:	turns off picking mode.	endpick
endselect:	turns off selecting mode.	endselect
cursoff:	turns off the cursor.	cursoff
clkoff:	turns off the keyboard click.	clkoff
lights. lampoff:	turns off the keyboard display	lampoff
tpoff:	turns off the textport.	tpoff
feedback:	turns on feedback mode.	feedback
curson:	turns on the cursor.	curson
clkon:	turns on the keyboard click.	clkon
lights. lampon:	turns on the keyboard display	lampon
tpon:	turns on the textport.	tpon
blankscreen:	turns screen refresh on and off.	blankscreen
returns the nondisplayed monitor	type. getothermonitor:	getothermonitor
setmonitor: sets the monitor	type.	setmonitor
getport: creates a graphics window	under the window manager.	getport
gentag: returns a	unique integer for use as a tag.	gentag
identifier. genobj: returns a	unique integer for use as an object . . .	genobj
device from making entries in the/	unqdevice: disables the specified . . .	unqdevice

gflush: forces all unsent routines down the network. . . . gflush
setdepth: sets up a 3-D viewport. . . . . setdepth
getbutton: returns the state (up or down) of a button. . . . . getbutton
position. cmov: updates the current character . . . . . cmov
backbuffer: enables updating in the back buffer. . . . . backbuffer
frontbuffer: enables updating in the front buffer. . . . . frontbuffer
gentag: returns a unique integer for use as a tag. . . . . gentag
genobj: returns a unique integer for use as an object identifier. . . . . genobj
sets range of color indices used in depth-cueing. shaderange: . . . . shaderange
curvebasis: selects a basis matrix used to draw curves. . . . . curvebasis
istag: indicates if a given tag is used within the current open object. . . . istag
Pipeline is/ finish: blocks the user process until the Geometry . . . . . finish
returns the current state of a valuator. getvaluator: . . . . . getvaluator
noise: filters valuator motion. . . . . noise
a device (keyboard, button, or valuator). qdevice: queues . . . . . qdevice
assigns an initial value to a valuator. setvaluator: . . . . . setvaluator
getdev: reads a list of valuators at one time. . . . . getdev
attaches the cursor to two valuators. attachcursor: . . . . . attachcursor
tie: ties two valuators to a button. . . . . tie
gRGBcolor: returns the current RGB value. . . . . gRGBcolor
getresetls: returns the current value of resetls. . . . . getresetls
getlsbackup: returns the current value of the linestyle backup flag. . . . . getlsbackup
setvaluator: assigns an initial value to a valuator. . . . . setvaluator
fudge: specifies fudge values added to a graphics window. . . . fudge
state attributes to their initial values. greset: resets all global . . . . . greset
readpixels: returns values of specific pixels. . . . . readpixels
readRGB: returns values of specific pixels. . . . . readRGB
graphical primitives (floating point version). rot: rotates . . . . . rot
gsync: waits for a vertical retrace period. . . . . gsync
coordinates. polarview: defines the viewer's position in polar . . . . . polarview
lookat: defines a viewing transformation. . . . . lookat
window for an image. viewport: allocates an area of the . . . . viewport
clear: clears the viewport. . . . . clear
getviewport: returns the current viewport. . . . . getviewport
pushviewport: duplicates the current viewport. . . . . pushviewport
parameters. popviewport: restores viewport, screenmask, and setdepth . . . popviewport
setdepth: sets up a 3-D viewport. . . . . setdepth
current/ reshapeviewport: sets the viewport to the dimensions of the . . . . reshapeviewport
gsync: waits for a vertical retrace period. . . . . gsync
identifies an/ isobj: indicates whether a given object number . . . . . isobj
appear. getbackface: returns whether backfacing polygons will . . . . getbackface
getdcm: indicates whether depth-cue mode is on or off. . . getdcm
line are/ lsbackup: controls whether the last two pixels of a . . . . . lsbackup
getzbuffer: indicates whether z-buffering is on or off. . . . . getzbuffer
writing. getbuffer: indicates which buffers are enabled for . . . . . getbuffer
/sets the precision at which curves are drawn. . . . . patchprecision
strwidth: returns the width of the specified text string. . . . . strwidth
returns whether backfacing polygons will appear. getbackface: . . . . . getbackface
the window beneath the cursor. winat: returns the indentifier of . . . . . winat
to the current graphics window and/ winattach: attaches the input focus . . . winattach
graphics window. winclose: closes the identified . . . . . winclose

constraints of the current graphics/ winconstraints: changes the . . . . . winconstraints
input focus to the current graphics window and call process. /the . . . . . winattach
winpush: places the current graphics window behind all other windows. . . . winpush
returns the indentifier of the window beneath the cursor. winat: . . . winat
stepunit: specifies that a graphics window change size in discrete/ . . . . . stepunit
projection transformation. window: defines a perspective . . . . . window
viewport: allocates an area of the window for an image. . . . . viewport
fudge values added to a graphics window. fudge: specifies . . . . . fudge
a program the entire screen as a window. fullscrn: gives . . . . . fullscrn
returns the position of a graphics window. getorigin: . . . . . getorigin
returns the size of a graphics window. getsize: . . . . . getsize
winpop: moves the current graphics window in front of all other/ . . . . . winpop
the aspect ratio of a graphics window. keepaspect: specifies . . . . . keepaspect
creates a graphics window under the window manager. getport: . . . . . getport
ismex: returns TRUE if the window manager is running. . . . . ismex
the maximum size of a graphics window. maxsize: specifies . . . . . maxsize
the minimum size of a graphics window. minsize: specifies . . . . . minsize
program does not require a graphics window. noport: specifies that a . . . . noport
location and size of a graphics window. /specifies the preferred . . . . prefposition
the preferred size of a graphics window. prefsize: specifies . . . . . prefsize
dimensions of the current graphics window. /sets the viewport to the . . . . reshapeviewport
getport: creates a graphics window under the window manager. . . . getport
closes the identified graphics window. winclose: . . . . . winclose
constraints of the current graphics window. winconstraints: changes the . . winconstraints
identifier of the current graphics window. winget: returns the . . . . . winget
corner of the current graphics window. /moves the lower-left . . . . . winmove
winopen: creates a graphics window. . . . . winopen
and position of the current graphics window. /changes the size . . . . . winposition
winset: sets the current graphics window. . . . . winset
a title bar to the current graphics window. wintitle: adds . . . . . wintitle
window in front of all other windows. /moves the current graphics . . winpop
graphics window behind all other windows. /places the current . . . . . winpush
the current graphics window. winget: returns the identifier of . . . . . winget
of the current graphics window. winmove: moves the lower-left corner . . winmove
winopen: creates a graphics window. . . . winopen
window in front of all other/ winpop: moves the current graphics . . . winpop
position of the current graphics/ winposition: changes the size and . . . . winposition
window behind all other windows. winpush: places the current graphics . . . winpush
window. winset: sets the current graphics . . . . winset
current graphics window. wintitle: adds a title bar to the . . . . . wintitle
callfunc: calls a function from within an object. . . . . callfunc
/indicates if a given tag is used within the current open object. . . . . istag
gbegin: initializes the system without altering the color map. . . . . gbegin
maps a point on the screen into 2-D world coordinates. mapw2: mapw2
on the screen into a line in 3-D world coordinates. /maps a point . . . . mapw
available/ RGBwritemask: grants write access to a subset of . . . . . RGBwritemask
the textport. textwritemask: grants write permission for text drawn in . . . . textwritemask
bitplanes. writemask: grants write permission to available . . . . . writemask
background. pagewritemask: sets the writemask for the textport . . . . . pagewritemask
getwritemask: returns the current writemask. . . . . getwritemask
to available bitplanes. writemask: grants write permission . . . writemask

gRGBmask: returns the current RGB writemask. . . . . gRGBmask
on the screen. writepixels: paints a row of pixels . . . . writepixels
the screen. writeRGB: paints a row of pixels on . . . writeRGB
singlebuffer: writes and displays all bitplanes. . . . . singlebuffer
which buffers are enabled for writing. getbuffer: indicates . . . . . getbuffer
xfpt: transforms points. . . . . xfpt
zbuffer: starts or ends z-buffer mode. . . . . zbuffer
mode. zbuffer: starts or ends z-buffer . . . . . zbuffer
zclear: initializes the z-buffer. . . . . zclear
getzbuffer: indicates whether z-buffering is on or off. . . . . getzbuffer
zclear: initializes the z-buffer. . . . . zclear
sets the system hitcode to zero. clearhitcode: . . . . . clearhitcode

NAME

addtopup – adds items to an existing pop-up menu

SPECIFICATION

C
```
addtopup(pup, str, args)
long pup;
char *str;
long args;
```

FORTRAN
```
subroutine addtop(pup, str, length, args)
integer pup
character* str(*)
integer*4 length
integer*4 args
```

Pascal
```
procedure addtopup(pup: longint; str: pstring128;
        args: longint);
```

DESCRIPTION

addtopup adds items to an existing pop-up menu. *str* specifies the menu items; *pup* identifies the menu. **addtopup** appends the new menu items to the bottom of the existing menu. *pup* is the identifier returned by **newpup** or **defpup.**

args is a single argument that matches the argument *str* specifies. In FORTRAN and Pascal, menus are built by a call to **newpup** followed by a call to **addtopup** (**addtop** in FORTRAN) for each menu item that requires an argument, i.e., a function address or a menu identifier.

The following code segments construct the same pop-up menu using C and FORTRAN.

In C:

```
submenu = defpup("rotate %f | translate %f | scale %f ",
                        dorotate, dotranslate, doscale);
menu = defpup("sample %t | persp | xform %m | greset %f ",
                        submenu, greset);
```

In FORTRAN:

```
submenu = newpup()
call addtop(submen, "rotate %f", 9,  dorota)
call addtop(submen, "translate %f", 12, dotran)
call addtop(submen, "scale %f", 8, doscal)
menu = newpup()
call addtop(menu, "sample %t | persp | xform %m", 28,
+                   submenu)
call addtop(menu, "greset %f", 8, greset)
```

For a complete description of menu formats, see **defpup.**

SEE ALSO

defpup, dopup, freepup, newpup

Using mex, Chapter 3, Making Pop-Up Menus

NOTE

This routine is available only in immediate mode under the window manager.

NAME

arc – draws a circular arc

SPECIFICATION

C

```
arc(x, y, radius, startang, endang)
Coord x, y, radius;
Angle startang, endang;

arci(x, y, radius, startang, endang)
Icoord x, y, radius;
Angle startang, endang;

arcs(x, y, radius, startang, endang)
Scoord x, y, radius;
Angle startang, endang;
```

FORTRAN

```
subroutine arc(x, y, radius, stang, endang)
real x, y, radius
integer*4 stang, endang

subroutine arci(x, y, radius, stang, endang)
integer*4 x, y, radius, stang, endang

subroutine arcs(x, y, radius, stang, endang)
integer*2 x, y, radius
integer*4 stang, endang
```

Pascal

```
procedure arc(x, y, radius: Coord;
        startang, endang: Angle);

procedure arci(x, y, radius: Icoord;
        startang, endang: Angle);

procedure arcs(x, y, radius: Scoord;
        startang, endang: Angle);
```

DESCRIPTION

arc draws a circular arc. The parameters of an arc are the center point (*x,y*), radius (*radius*), starting angle (*startang*), and ending angle

(*endang*). The angle of the arc is measured from the positive *x* axis and specified in integral tenths of degrees (e.g., 90 degrees equal 900 tenths of degrees). Positive angles describe counterclockwise rotations. Since an arc is a 2-D shape, these routines have only 2-D forms. The arc is drawn in the *x*-*y* plane, with *z*=0, and uses the current color, linestyle, linewidth, and writemask. It is drawn counterclockwise from *startang* to *endang*. For example, an arc from 10 degrees to 5 degrees (100 to 50 tenths of degrees) is almost a complete circle. After **arc** executes, the graphics position is undefined.

SEE ALSO

arcf, circ, circf, crv

Programming Guide, Section 3.7, Circles and Arcs

NAME

arcf – draws a filled circular arc

SPECIFICATION

C

```
arcf(x, y, radius, startang, endang)
Coord x, y, radius;
Angle startang, endang;

arcfi(x, y, radius, startang, endang)
Icoord x, y, radius;
Angle startang, endang;

arcfs(x, y, radius, startang, endang)
Scoord x, y, radius;
Angle startang, endang;
```

FORTRAN

```
subroutine arcf(x, y, radius, stang, endang)
real x, y, radius
integer*4 stang, endang

subroutine arcfi(x, y, radius, stang, endang)
integer*4 x, y, radius, stang, endang

subroutine arcfs(x, y, radius, stang, endang)
integer*2 x, y, radius
integer*4 stang, endang
```

Pascal

```
procedure arcf(x, y, radius: Coord; startang,
        endang: Angle);

procedure arcfi(x, y, radius: Icoord; startang,
        endang: Angle);

procedure arcfs(x, y, radius: Scoord; startang,
        endang: Angle);
```

DESCRIPTION

arcf draws a filled circular arc (pie section). The arc is specified as a center point (*x,y*), a radius (*radius*), a starting angle (*startang*), and an

ending angle (*endang*). The angle of the arc is measured from the *x* axis and specified in integral tenths of degrees. Positive angles describe counterclockwise rotations. The arc is drawn using the current color and writemask, and is filled with the current texture. Since an arc is a 2-D shape, these routines have only 2-D forms. The arc is in the *x-y* plane with *z*=0. Arcs are drawn counterclockwise from *startang* to *endang*, so the arc from 10 degrees to 5 degrees is a nearly complete circle.

SEE ALSO

arc, circ, circf, crv

Programming Guide, Section 3.7, Circles and Arcs

NAME

attachcursor – attaches the cursor to two valuators

SPECIFICATION

C
```
attachcursor(vx, vy)
Device vx, vy;
```

FORTRAN
```
subroutine attach(vx, vy)
integer*4 vx, vy
```

Pascal
```
procedure attachcursor(vx, vy: Device);
```

DESCRIPTION

attachcursor attaches the cursor to the movement of two valuators. *vx* and *vy* are both valuator device numbers. (See Appendix A, Valuators, for a list of device numbers.) The first valuator (*vx*) controls the horizontal motion of the cursor; the second valuator (*vy*) determines vertical motion. The values of *vx* and *vy* determine the cursor position in screen coordinates.

SEE ALSO

noise, tie

Programming Guide, Section 7.2, Initializing a Device

NOTE

This routine is available only in immediate mode.

NAME

backbuffer – enables updating in the back buffer

SPECIFICATION

C
```
backbuffer(b)
Boolean b;
```

FORTRAN
```
subroutine backbu(b)
logical b
```

Pascal
```
procedure backbuffer(b: Boolean);
```

DESCRIPTION

backbuffer enables updating in the back bitplane buffer. When the value of *b* is TRUE (1), the default, the back buffer is enabled. When the value of *b* is FALSE (0), the back buffer is not enabled. This routine is useful only in double buffer mode, and is ignored in single buffer mode and RGB mode.

gconfig sets **backbuffer** to TRUE (1).

SEE ALSO

doublebuffer, frontbuffer, getbuffer

Programming Guide, Section 6.1, Display Modes

NAME

backface – turns backfacing polygon removal on and off

SPECIFICATION

C
```
backface(b)
Boolean b;
```

FORTRAN
```
subroutine backfa(b)
logical b
```

Pascal
```
procedure backface(b: Boolean);
```

DESCRIPTION

backface initiates or terminates backfacing filled polygon removal. A backfacing polygon is a polygon whose vertices appear in clockwise order in screen coordinates. When backfacing polygon removal is on, the system displays only polygons whose vertices appear in counterclockwise order.

The backface utility improves the performance of programs that represent solid objects as collections of polygons. Backfacing polygon removal does not always remove all hidden surfaces because some frontfacing polygons can still be obscured. When a polygon shrinks to the point where its vertices are coincident, its orientation is indeterminate and it is displayed. Backface removal is useful for simple convex objects. For more general images, you can achieve hidden surface removal using another technique, perhaps in conjunction with backface removal.

Matrices that negate coordinates, such as scale (-1.0, 1.0, 1.0) reverse the directional order of a polygon's points and can cause to do the opposite of what was intended.

SEE ALSO

zbuffer

Programming Guide, Section 12.2, Backfacing Polygon Removal

NAME

bbox2 – specifies bounding box and minimum pixel radius

SPECIFICATION

C

```
bbox2(xmin, ymin, x1, y1, x2, y2)
Screencoord xmin, ymin;
Coord x1, y1, x2, y2;

bbox2i(xmin, ymin, x1, y1, x2, y2)
Screencoord xmin, ymin;
Icoord x1, y1, x2, y2;

bbox2s(xmin, ymin, x1, y1, x2, y2)
Screencoord xmin, ymin;
Scoord x1, y1, x2, y2;
```

FORTRAN

```
subroutine bbox2(xmin, ymin, x1, y1, x2, y2)
integer*4 xmin, ymin
real x1, y1, x2, y2

subroutine bbox2i(xmin, ymin, x1, y1, x2, y2)
integer*4 xmin, ymin, x1, y1, x2, y2

subroutine bbox2s(xmin, ymin, x1, y1, x2, y2)
integer*4 xmin, ymin
integer*2 x1, y1, x2, y2
```

Pascal

```
procedure bbox2(xmin, ymin: Screencoord;
        x1, y1, x2, y2: Coord);

procedure bbox2i(xmin, ymin: Screencoord;
        x1, y1, x2, y2: Icoord);

procedure bbox2s(xmin, ymin: Screencoord;
        z1, y1, x2, y2: Scoord);
```

DESCRIPTION

bbox2 controls the execution of routines in a Graphics Library object. Its arguments are the coordinates of a bounding box in object space and

minimum horizontal and vertical feature sizes in pixels. **bbox2** transforms the bounding box to screen coordinates and compares it with the viewport. If the bounding box is completely outside the viewport, the system ignores routines between **bbox2** and the end of the object. Otherwise, the system compares the transformed bounding box with the minimum feature size. If the bounding box is too small in both the *x* and *y* dimensions, the rest of the routines in the object are ignored. Otherwise, interpretation of the object continues.

SEE ALSO

Programming Guide, Section 8.2, Using Objects

NAME

blankscreen – turns screen refresh on and off

SPECIFICATION

C
```
blankscreen(b)
Boolean bool;
```

FORTRAN
```
subroutine blanks(b)
logical bool
```

Pascal
```
procedure blankscreen(bool: Boolean);
```

DESCRIPTION

blankscreen turns screen refresh on and off. *b* = TRUE(1) stops display; *b* = FALSE(0) restarts display.

When bitplanes are simultaneously viewed and updated (as in single buffer mode, RGB mode, or when the front buffer is displayed in double buffer mode), there is competition for memory which reduces performance. This is most true for noninterlaced monitors. To speed up drawing in these cases, turn off the display while drawing.

SEE ALSO

Programming Guide, Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

NAME

blanktime – sets the screen blanking timeout

SPECIFICATION

C
```
blanktime(nframes)
long nframes;
```

FORTRAN
```
subroutine blankt(nframes)
integer*4 nframes
```

Pascal
```
procedure blanktime(nframes: longint);
```

DESCRIPTION

The screen blanks (turns black) after the system receives no input for about 15 minutes. This protects the color display. **blanktime** changes the amount of time the system waits before blanking the screen. It can also disable the screen blanking feature.

nframes specifies the screen blanking timeout in frame times based on the standard 60-Hz monitor. For software compatibility, the factor of 60 is used, regardless of the monitor type. When calculating the value of *nframes*, simply multiply the desired blanking latency period (in seconds) by 60. For example, when *nframes* is 1800, the blanking latency period is 5 minutes. If there are 60 frames per second, *nframes* is 60 times the number of seconds that the system waits before blanking the screen. When *nframes* is zero, screen blanking is disabled.

SEE ALSO

blanktime(1G)

Programming Guide, Section 2.1, Initialization

NAME

blink – changes a color map entry at a selectable rate

SPECIFICATION

C
```
blink(rate, i, red, green, blue)
short rate, red, green, blue;
Colorindex i;
short red, green, blue
```

FORTRAN
```
subroutine blink(rate, i, red, green, blue)
integer*4 rate, i, red, green, blue
```

Pascal
```
procedure blink(rate: Short: color: Colorindex;
        red, green, blue: Short);
```

DESCRIPTION

blink specifies blink rate, color map index, and red, green, and blue values. *rate* is measured in terms of vertical retraces. The system updates the color located at index *i* in the current color map according to *rate*. For example, if *rate* is 3, the color changes (blinks) every third vertical retrace. Its value alternates between the original value and the new value supplied by *red*, *green*, and *blue*. The length of time between retraces varies according to the monitor used. On the standard 60Hz monitor, there are 60 retraces per second. When using this monitor, a rate of 60 would cause the color to change once every second.

Up to 20 colors can blink simultaneously, each at a different rate. You can change the blink rate by calling **blink** with the same *i* and a different *rate*. To terminate blinking and restore the original color when *i* specifies a blinking colormap entry, call **blink** with *rate* = 0. To terminate all blinking colors simultaneously, call **blink** with *rate*=-1. When you set *rate* to -1, the other parameters are ignored.

SEE ALSO

mapcolor, color

Programming Guide, Section 6.2, Color Maps

NOTE

This routine available only in immediate mode.

NAME

blkqread – reads multiple entries from the queue

SPECIFICATION

C
```
long blkqread(data, n)
short *data
short n;
```

FORTRAN
```
integer*4 function blkqre(data, n)
integer*2 data(*)
integer*4 n
```

Pascal
```
function blkqread(var data: Short; n: longint):
        longint;
```

DESCRIPTION

blkqread reads multiple entries from the input queue. *data* is an array of short integers, and *n* is the size of the array data. **blkqread** returns the number of queue entries read, and *data* is filled alternatively with device numbers and device values. Note that the number of entries read is at *n*/2.

SEE ALSO

qread

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

callfunc – calls a function from within an object

SPECIFICATION

C **callfunc(fctn, nargs, arg1, arg2, ..., argn)**
int (*fctn)();
long nargs, arg1, arg2, ..., argn;

FORTRAN **available only in C**

Pascal **available only in C**

DESCRIPTION

callfunc calls an arbitrary function from within an object. The function *fctn (nargs, arg1, arg2, ..., argn)* is called when **callfunc** executes in the object.

SEE ALSO

Programming Guide, Section 8.3, Object Editing

NOTE

This routine cannot be called remotely.

NAME

callobj – draws an instance of an object

SPECIFICATION

C
```
callobj(obj)
Object obj;
```

FORTRAN
```
subroutine callob(obj)
integer*4 obj
```

Pascal
```
procedure callobj(obj: Object);
```

DESCRIPTION

callobj draws an instance of a previously defined object. *obj* is the object identifier. If **callobj** specifies an undefined object, the system ignores the routine.

Global state attributes are not saved before a call to **callobj**. For example, if you change a variable within an object, such as color, the change can affect the caller as well. Use **pushattributes** and **popattributes** to preserve global state attributes across **callobj** calls.

SEE ALSO

makeobj, popattributes, pushattributes

Programming Guide, Section 8.2, Using Objects

NAME

charstr – draws a string of raster characters on the screen

SPECIFICATION

C
```
charstr(str)
String str;
```

FORTRAN
```
subroutine charst(str, length)
character*(*) str
integer*4 length
```

Pascal
```
procedure charstr(str: pstring128);
```

DESCRIPTION

charstr draws a string of text (*str*) using a raster font. The current character position is the position of the first character in the string. After each character is drawn, the character's spacing parameter updates the current character position. The text string is drawn in the current raster font and color, using the current writemask. The system ignores characters that are not defined in the current raster font.

In FORTRAN, *str* is the name of the text string and *length* is the number of characters in that string.

SEE ALSO

cmov, defrasterfont, font

Programming Guide, Section 3.8, Text

NAME

chunksize – specifies minimum object size in memory

SPECIFICATION

C
```
chunksize(chunk)
long chunk;
```

FORTRAN
```
subroutine chunks(chunk)
integer*4 chunk
```

Pascal
```
procedure chunksize(chunk: longint);
```

DESCRIPTION

chunksize specifies the minimum object size in memory. You can call it only once after graphics initialization (i.e., **ginit** or **winopen**) and before the first **makeobj**. *chunk* is the unit size (in bytes) by which an object grows.

Use **chunksize** only if there is a limited amount of memory. **chunksize** is typically used when an application has many objects. If **chunksize** is set too small, large items (e.g., large polygons) do not fit into the display list as each must fit entirely into a single chunk. Some experimentation may be necessary to determine the optimal chunksize for an application. The default chunk size is 1020 bytes.

SEE ALSO

compactify, ginit, makeobj

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

circ – outlines a circle

SPECIFICATION

C

```
circ(x, y, radius)
Coord x, y, radius;

circi(x, y, radius)
Icoord x, y, radius;

circs(x, y, radius)
Scoord x, y, radius;
```

FORTRAN

```
subroutine circ(x, y, radius)
real x, y, radius

subroutine circi(x, y, radius)
integer*4 x, y, radius

subroutine circs(x, y, radius)
integer*2 x, y, radius
```

Pascal

```
procedure circ(x, y, radius: Coord);

procedure circi(x, y, radius: Icoord);

procedure circs(x, y, radius: Scoord);
```

DESCRIPTION

circ outlines a circle. The circle has a center at point (*x*,*y*) and a radius (*radius*), which are specified in world coordinates. Since a circle is a 2-D shape, these routines have only 2-D forms (note that circles rotated outside the 2-D *x*-*y* plane appear as ellipses). The circle is in the *x*-*y* plane, with *z*=0. The system draws the circle using the current color,

linestyle, linewidth, and writemask.

SEE ALSO

arc, arcf, circf, crv

Programming Guide, Section 3.7, Circles and Arcs

NAME

circf – draws a filled circle

SPECIFICATION

C

```
circf(x, y, radius)
Coord x, y, radius;

circfi(x, y, radius)
Icoord x, y, radius;

circfs(x, y, radius)
Scoord x, y, radius;
```

FORTRAN

```
subroutine circf(x, y, radius)
real x, y, radius

subroutine circfi(x, y, radius)
integer*4 x, y, radius

subroutine circfs(x, y, radius)
integer*2 x, y, radius
```

Pascal

```
procedure circf(x, y, radius: Coord);

procedure circfi(x, y, radius: Icoord);

procedure circfs(x, y, radius: Scoord);
```

DESCRIPTION

circf draws a filled circle, using the current color, writemask, and pattern. The circle has its center point at *(x,y)* and a radius (*radius*), which are both specified in world coordinates. Since a circle is a 2-D shape, these routines have only 2-D forms (note that circles rotated outside of the 2-D *x-y* planes appear as ellipses). The circle is drawn in the *x-y* plane, with *z*=0.

SEE ALSO

arc, arcf, circ, crv

Programming Guide, Section 3.7, Circles and Arcs

NAME

clear – clears the viewport

SPECIFICATION

C **clear()**

FORTRAN **subroutine clear**

Pascal **procedure clear;**

DESCRIPTION

clear sets the screen area in the current viewport to the current color using the current writemask and pattern.

SEE ALSO

rect, rectf

Programming Guide, Section 3.2, Clearing the Viewport

NAME

clearhitcode – sets the system hitcode to zero

SPECIFICATION

C **clearhitcode()**

FORTRAN **subroutine clearh**

Pascal **procedure clearhitcode;**

DESCRIPTION

clearhitcode clears the global variable *hitcode,* which records clipping plane hits in picking and selecting modes.

SEE ALSO

gethitcode, gselect, pick

Programming Guide, Section 9.2, Picking

NOTE

This routine is available only in immediate mode.

NAME

clkoff – turns off the keyboard click

SPECIFICATION

C **clkoff()**

FORTRAN **subroutine clkoff**

Pascal **procedure clkoff;**

DESCRIPTION

clkoff turns off the keyboard click.

SEE ALSO

clkon, lampoff, lampon, ringbell, setbell

Programming Guide, Section 7.5, Controlling Peripheral Input/Output Devices

NOTE

This routine is available only in immediate mode.

NAME

clkon – turns on the keyboard click

SPECIFICATION

C **clkon()**

FORTRAN **subroutine clkon**

Pascal **procedure clkon;**

DESCRIPTION

clkon turns on the keyboard click.

SEE ALSO

clkoff, lampoff, lampon, ringbell, setbell

Programming Guide, Section 7.5, Controlling Peripheral Input/Output Devices

NOTE

This routine is available only in immediate mode.

NAME

closeobj – closes an object

SPECIFICATION

C **closeobj()**

FORTRAN **subroutine closeo**

Pascal **procedure closeobj;**

DESCRIPTION

closeobj closes an object that is open. Use **makeobj** to create and open a new object. All display list routines between **makeobj** and **closeobj** become part of the object definition. Use **editobj** to open an existing object for editing. Use **closeobj** to terminate the editing session.

If no object is open, **closeobj** is ignored.

SEE ALSO

editobj, makeobj

Programming Guide, Section 8.1, Defining an Object

NOTE

This routine is available only in immediate mode.

NAME

cmov – updates the current character position

SPECIFICATION

C

```
cmov(x, y, z)
Coord x, y, z;

cmovi(x, y, z)
Icoord x, y, z;

cmovs(x, y, z)
Scoord x, y, z;

cmov2(x, y)
Coord x, y;

cmov2i(x, y)
Icoord x, y;

cmov2s(x, y)
Scoord x, y;
```

FORTRAN

```
subroutine cmov(x, y, z)
real x, y, z

subroutine cmovi(x, y, z)
integer*4 x, y, z

subroutine cmovs(x, y, z)
integer*2 x, y, z

subroutine cmov2(x, y)
real x, y

subroutine cmov2i(x, y)
integer*4 x, y

subroutine cmov2s(x, y)
integer*2 x, y
```

Pascal **procedure cmove(x, y, z: Coord);**
procedure cmovi(x, y, z: Icoord);
procedure cmovs(x, y, z: Scoord);
procedure cmov2(x, y: Coord);
procedure cmov2i(x, y: Icoord);
procedure cmov2s(x, y: Scoord);

DESCRIPTION

cmov moves the current character position to a specified point (as **move** sets the current line drawing position). *x*, *y*, *z* are integers, shorts, or real numbers in 2-D or 3-D space that specify a point in world coordinates. **cmov** transforms the specified world coordinates into screen coordinates, which become the new character position. If the transformed point is outside the viewport, the character position is undefined.

cmov does not affect the current graphics position.

SEE ALSO

charstr, move, readpixels, readRGB, writepixels, writeRGB

Programming Guide, Section 3.8, Text

NAME

color – sets the color index in the current mode

SPECIFICATION

C
```
color(c)
Colorindex c;
```

FORTRAN
```
subroutine color(c)
integer*4 c
```

Pascal
```
procedure color(c: Colorindex);
```

DESCRIPTION

color sets the current color. The current color (*c*) is an index into a color map. In onemap mode, the color can be from 0 to 4095. In multimap mode, **color** is used in conjunction with a color map number and is between 0 and 255. The number of bitplanes determines the number of colors.

SEE ALSO

getcolor, RGBcolor, RGBwritemask, writemask

Programming Guide, Section 6.2, Color Maps

NAME

compactify – compacts the memory storage of an object

SPECIFICATION

C
```
compactify(obj)
Object obj;
```

FORTRAN
```
subroutine compac(obj)
integer*4 obj
```

Pascal
```
procedure compactify(obj: Object);
```

DESCRIPTION

When you modify an open object using the object editing routines, its storage can become fragmented and inefficient. If there is too much wasted space, the system automatically calls **compactify** during the **closeobj** operation. **compactify** performs the compaction explicitly. Unless there is insertion to or deletion from an object, compaction is never necessary. Since **compactify** requires a significant amount of time, do not call it unless storage space is critical.

SEE ALSO

chunksize, closeobj

Programming Guide, Section 8.3, Object Editing

NOTE

This routine available only in immediate mode.

NAME

crv – draws a curve

SPECIFICATION

C
```
crv(geom)
Coord geom[4][3];
```

FORTRAN
```
subroutine crv(geom)
real geom(3,4)
```

Pascal
```
procedure crv(var geom: Coord);
```

DESCRIPTION

crv draws a cubic spline curve segment using the current curve basis and precision. **rcrv** draws the rational spline. *geom* specifies the four control points of the curve segment.

SEE ALSO

curvebasis, curveprecision, crvn, defbasis, rcrv, rcrvn

Programming Guide, Section 11.2, Drawing Curves

NAME

crvn – draws a series of curve segments

SPECIFICATION

C
```
crvn(n, geom)
long n;
Coord geom[][3];
```

FORTRAN
```
subroutine crvn(n, geom)
integer*4 n
real geom(3,n)
```

Pascal
```
procedure crvn(n: longint; var geom: Coord);
```

DESCRIPTION

crvn draws a series of cubic spline curve segments using the current basis and precision. **rcrvn** draws the rational spline. The control points specified in *geom* determine the shapes of the curve segments and are used four at a time. For example, if *n* is 6, three curve segments are drawn, the first using points 0,1,2,3 as control points, and the second and third segments are controlled by points 1,2,3,4 and 2,3,4,5, respectively. If the current basis is a B-spline, Cardinal spline, or basis with similar properties, the curve segments are joined end to end and appear as a single curve.

SEE ALSO

crv, crvn, curvebasis, curveprecision, defbasis, rcrvn

Programming Guide, Section 11.2, Drawing Curves

NAME

curorigin – sets the origin of a cursor

SPECIFICATION

C
```
curorigin(n, xorigin, yorigin)
short n, xorigin, yorigin;
```

FORTRAN
```
subroutine curori(n, xorigin, yorigin)
integer*4 n, xorigin, yorigin
```

Pascal
```
procedure curorigin(n, xorigin, yorigin: Short);
```

DESCRIPTION

curorigin sets the origin of a cursor. The origin is the point on the cursor that aligns with the current cursor valuators. The lower left corner of the cursor has coordinates (0,0). Before calling **curorigin**, the cursor must be defined with **defcursor**. *n* is an index into the cursor table created by **defcursor**. **curorigin** does not take effect until you call **setcursor**.

The default of **curorigin** is at (0,0) for user-defined glyphs.

SEE ALSO

attachcursor, defcursor, setcursor

Programming Guide, Section 6.4, Cursors

NOTE

This routine is available only in immediate mode.

NAME

cursoff – turns off the cursor

SPECIFICATION

C **cursoff()**

FORTRAN **subroutine cursof**

Pascal **procedure cursoff;**

DESCRIPTION

cursoff turns off the cursor so that it is no longer visible. **cursoff** should precede drawing routines that write into the currently displayed cursor bitplanes.

The cursor is always displayed by default. Before the cursor is drawn on the screen, the IRIS saves the image that the cursor covers. When the cursor moves, the system restores the saved image. If the image changes while the cursor is displayed, the saved image may no longer be valid. This is a concern in single buffer and RGB modes, and in double buffer mode when the front buffer is enabled.

SEE ALSO

curson, getcursor, setcursor

Programming Guide, Section 6.4, Cursors

NOTE

This routine is available only in immediate mode.

NAME

curson – turns on the cursor

SPECIFICATION

C **curson()**

FORTRAN **subroutine curson**

Pascal **procedure curson;**

DESCRIPTION

curson automatically updates and displays a cursor. **curson** is usually paired with **cursoff.** You can call **curson** when the automatic cursor is already visible.

SEE ALSO

attachcursor, cursoff, getcursor, setcursor

Programming Guide, Section 6.4, Cursors

NOTE

This routine is available only in immediate mode.

NAME

curvebasis – selects a basis matrix used to draw curves

SPECIFICATION

C
```
curvebasis(basisid)
short basisid;
```

FORTRAN
```
subroutine curveb(basisid)
integer*4 basisid
```

Pascal
```
procedure curvebasis(basisid: Short);
```

DESCRIPTION

curvebasis selects a basis matrix (defined by **defbasis**) as the current basis matrix to draw curve segments.

SEE ALSO

crv, crvn, curveprecision, defbasis

Programming Guide, Section 11.2, Drawing Curves

NAME

curveit – draws a curve segment

SPECIFICATION

C
```
curveit(iterationcount)
short iterationcount;
```

FORTRAN
```
subroutine curvei(count)
integer*4 count
```

Pascal
```
procedure curveit(iterationcount: Short);
```

DESCRIPTION

curveit iterates the forward differences of the matrix on top of the transformation matrix stack *iterationcount* times; it issues a draw routine with each iteration. **curveit** accesses low-level hardware capabilities for curve drawing.

SEE ALSO

crv, resetls

Programming Guide, Section 11.2, Drawing Curves

NAME

curveprecision – sets the number of line segments that draw a curve segment

SPECIFICATION

C
```
curveprecision(nsegments)
short nsegments;
```

FORTRAN
```
subroutine curvep(nsegments)
integer*4 nsegments
```

Pascal
```
procedure curveprecision(nsegments: short);
```

DESCRIPTION

curveprecision sets the number of line segments used to draw a curve. Whenever **crv, crvn, rcrv,** or **rcrvn** executes, a number of straight line segments (*nsegments*) approximates each curve segment. The greater the value of *nsegments*, the smoother the curve, but the longer the drawing.

SEE ALSO

crv, curvebasis, crvn, rcrv, rcrvn

Programming Guide, Section 11.2, Drawing Curves

NAME

cyclemap – cycles through color maps at a specified rate

SPECIFICATION

C **cyclemap(duration, map, nextmap)**
short duration, map, nextmap;

FORTRAN **subroutine cyclem(duration, map, nextmap)**
integer*4 duration, map, nextmap

Pascal **procedure cyclemap(duration, map, nextmap: Scoord);**

DESCRIPTION

cyclemap specifies the duration in vertical retraces the affected map (*map*) and the next map to use (*nextmap*) when the duration is over. For example, the following routines set up multimap mode and cycle between two maps, leaving map 1 on for ten vertical retraces and map 3 on for five retraces. **cyclemap** must be used in multimap mode.

```
...
multimap();
gconfig();
cyclemap(10, 1, 3);
cyclemap(5, 3, 1);
...
```

SEE ALSO

blink, gconfig, multimap

Programming Guide, Section 6.2, Color Maps

NOTE

This routine is available only in immediate mode.

NAME

dbtext – sets the dial and button box text

SPECIFICATION

C
```
dbtext(str)
char *str
```

FORTRAN
```
subroutine dbtext(str)
character*(8) str
```

Pascal
```
procedure dbtext(var str: Byte);
```

DESCRIPTION

dbtext places up to eight characters of text into the text window on the dial and button box. Use only digits, spaces, and uppercase letters.

SEE ALSO

setdblights

Programming Guide, Section 7.5, Controlling Peripheral Input/Output Devices

NOTE

This routine is available only in immediate mode.

NAME

defbasis – defines a basis matrix

SPECIFICATION

C
```
defbasis(id, mat)
long id;
Matrix mat;
```

FORTRAN
```
subroutine defbas(id, mat)
integer*4 id
real mat(4,4)
```

Pascal
```
procedure defbasis(id: Short; var mat: Matrix);
```

DESCRIPTION

defbasis defines basis matrices to generate curves and patches. *matrix* is saved and is associated with *id*. Use *id* in subsequent calls to **curvebasis** and **patchbasis.**

When using multiple windows, patterns, cursors, and fonts are available to all windows.

SEE ALSO

crv, crvn, curvebasis, curveprecision, patch, patchbasis, patchcurves, patchprecision

Programming Guide, Section 11.2, Drawing Curves

NOTE

This routine is available only in immediate mode.

NAME

defcursor – defines a cursor glyph

SPECIFICATION

C
```
defcursor(n, curs)
short n;
Cursor curs;
```

FORTRAN
```
subroutine defcur(n, curs)
integer*4 n
integer*2 curs(16)
```

Pascal
```
procedure defcursor(n: Short; var curs: Cursor);
```

DESCRIPTION

defcursor defines a cursor glyph. The arguments are a name (*n*) and a 16x16 bitmap. The cursor origin is at the cursor's lower-left corner by default; use **curorigin** to reset it. The cursor origin is the position influenced by valuators attached to the cursor, and is also the position **pick** uses for the picking region. The name is used in subsequent cursor routines. An arrow is defined as cursor 0 by default and cannot be redefined. To replace a cursor, the new cursor must have the same index as the previous one.

When using multiple windows, patterns, cursors, and fonts are available to all windows.

SEE ALSO

curorigin, getcursor, pick, setcursor

Programming Guide, Section 6.4, Cursors

NOTE

This routine can be used only in immediate mode.

NAME

deflinestyle – defines a linestyle

SPECIFICATION

C
```
deflinestyle(n, ls)
short n;
Linestyle ls;
```

FORTRAN
```
subroutine deflin(n, ls)
integer*4 n, ls
```

Pascal
```
procedure deflinestyle(n: Short; ls: Linestyle);
```

DESCRIPTION

deflinestyle defines a linestyle which is a write-enabled pattern that is applied when lines are drawn. The least-significant bit of the linestyle is applied first. *n* specifies an index into a table where the linestyles are stored, and *ls* specifies a 16-bit pattern. You can define up to 2^{15} linestyles. By default, index 0 contains the pattern 0xFFFF, which draws solid lines and cannot be redefined. There is no performance penalty for drawing non-solid lines. To replace a linestyle, respecify the previous index.

When using multiple windows, patterns, cursors, and fonts are available to all windows.

SEE ALSO

defcursor, defpattern, defrasterfont, getlstyle, lsrepeat, setlinestyle

Programming Guide , Section 5.1, Linestyles

NOTE

This routine can be used only in immediate mode.

NAME

defpattern – defines patterns

SPECIFICATION

C
```
defpattern(n, size, mask)
short n, size;
short *mask;
```

FORTRAN
```
subroutine defpat(n, size, mask)
integer*4 n, size
integer*2 mask((size*size)/16)
```

Pascal
```
procedure defpattern(n, size: Short; var mask: Short;
```

DESCRIPTION

defpattern defines an arbitrary pattern. *n* specifies an index into a table of patterns. *size* specifies the size of the pattern: 16, 32, or 64 for a 16x16-, 32x32-, or 64x64-bit pattern, respectively. *mask* is an array of shorts that form the actual bit pattern. The pattern is described from left to right and bottom to top, just as characters are described in a raster font. By default, pattern 0 is a 16x16 solid pattern that cannot be changed. There is no performance penalty for non-solid patterns.

When using multiple windows, patterns, cursors, and fonts are available to all windows.

SEE ALSO

defcursor, defrasterfont, deflinestyle, getpattern, setpattern

Programming Guide, Section 5.2, Patterns

NOTE

This routine is available only in immediate mode.

NAME

defpup – defines a menu

SPECIFICATION

C
```
int defpup(str [, args ] ... )
char *str;
long args;
```

FORTRAN **available only in C**

Pascal **available only in C**

DESCRIPTION

defpup defines a pop-up menu in the window manager; it returns a positive menu identifier. *str* is a string that describes each menu item. Menu items include *title, submenu,* or *selectable item. title* is the string displayed at the top of the menu. *submenu* is an item that invokes a submenu with further choices if rolled off to either side. A *selectable item,* selected with the right mouse button, can be a numeric value, function, or an implied numeric value associated with it.

The optional arguments [, *args*] provide submenu identifiers returned by **defpup** and **newpup**, and/or handling function addresses dictated by *str*. *str* is made up of label/type pairs separated by vertical bars. Each label/type pair includes a menu label and an optional typing character that is preceded by a percent sign (%). The item types are:

%t	title string.
%F	menu function invoked by any item selection.
%f	item function invoked by selecting the associated item with the mouse button.
%m	submenu brought up by rolling off the associated item
%n	takes no arguments.
%x#	numeric item selected by selecting the associated item with the mouse button.

An example best illustrates the use of the item types.

menu=defpup("foo %t |item 1|item 2 |item 3 |item 4");

defines a pop-up menu with title *foo* and four items. You can use a menu of this type as follows:

```
switch (dopup(menu)) {
  case 1 : /* item 1 */
           handling code
           break;
  case 2 : /* item 2 */
           handling code
           break;
  case 3 : /* item 3 */
           handling code
           break;
  case 4 : /* item 4 */
           handling code
           break;
}
```

A more complex example is:

```
menu=defpup("
  foobar %t %F|item 1%n|item 2%m|item 3%f|item 4%x234",
    menufunc, submenu, func);
```

defines a menu with title *foobar* and four items. Invoked by:

menuval = dopup(menu);

Selecting menu item 1 causes **dopup** to return **menufunc(1)**.

Rolling off menu item 2 displays *submenu*, which provides additional selections. **dopup** returns **menufunc(dopup(submenu))** when another selection is made; otherwise *submenu* disappears and selections are made from *menu*.

Buttoning item 3 executes *func* with 3 as its argument. **dopup** returns **menufunc(func(3))**.

Buttoning item 4 causes **dopup** to return **menufunc(234)**.

If no item is selected, then **dopup** returns –1.

pupmode or *full-screen mode* executes any function that is invoked from a pop-up menu. If a menu handling function uses the pop-up planes and/or regions of the screen beyond the process's graphics window, it must make calls to **pupmode** and/or **fullscrn** .

SEE ALSO

addtopup, dopup, freepup, newpup

Using mex, Chapter 3, Making Pop-Up Menus

NOTE

This routine is available only in immediate mode.

FORTRAN and Pascal do not support this routine; use **newpup** and **addtopup**.

NAME

defrasterfont – defines a raster font

SPECIFICATION

C

```
defrasterfont(n, ht, nc, chars, nr, raster)
short n, ht, nc, nr;
Fontchar chars[];
short raster[];
```

FORTRAN

```
subroutine defras(n, ht, nc, chars, nr, raster)
integer*4 n, ht, nc, nr
integer*2 raster(nr), chars(4*nc)
```

Pascal

```
procedure defrasterfont(n, ht, nc: Short; var chars:
        Fontchar; nr: Short; var raster: Short);
```

DESCRIPTION

defrasterfont defines a raster font. *n* is an index into the font table; *ht* is an integer that specifies the maximum height of characters in the font in pixels. *n* becomes the font's internal name. *nc* gives the number of characters in the font and the number of elements in *chars* array.

chars contains a description of each character in the font. The description includes the height and width of the character in pixels, the offsets from the character origin to the lower-left corner of the character's bounding box, an offset into the array of rasters, and the amount to add to the current character position *x* after drawing the character.

raster is an array of *nr* shorts of bitmap information. It is a one-dimensional array of mask shorts, ordered left to right and bottom to top. Mask bits are left-justified in the character's bounding box.

To replace a raster font, specify the index of the previous font as the index for the new font. To delete a raster font, define a font with no characters. The default font, 0, is a fixed-pitch font with a height of 16 and width of 9. Font 0 cannot be redefined.

When using multiple windows, patterns, cursors, and fonts are available to all windows.

SEE ALSO

charstr, font, getdescender, getfont, getheight, strwidth

Programming Guide, Section 5.3, Fonts

NOTE

This routine is available only in immediate mode.

NAME

delobj – deletes an object

SPECIFICATION

C
```
delobj(obj)
Object obj;
```

FORTRAN
```
subroutine delobj(obj)
integer*4 obj
```

Pascal
```
procedure delobj(obj: Object);
```

DESCRIPTION

delobj deletes an object. Deleting an object frees most of its display list storage; the object number remains undefined until it is used to create a new object. The system ignores calls to objects that don't exist.

SEE ALSO

compactify, makeobj

Programming Guide, Section 8.1, Defining an Object

NOTE

This routine is available only in immediate mode.

NAME

deltag – deletes tags from objects

SPECIFICATION

C
```
deltag(t)
Tag t;
```

FORTRAN
```
subroutine deltag(t)
integer*4 t
```

Pascal
```
procedure deltag(t: Tag);
```

DESCRIPTION

deltag removes the tag *t* from the object currently open for editing. You cannot delete the special tags STARTTAG and ENDTAG.

SEE ALSO

editobj, maketag

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

depthcue – turns depth-cue mode on and off

SPECIFICATION

C
```
depthcue(mode)
short mode;
```

FORTRAN
```
subroutine depthc(mode)
logical mode
```

Pascal
```
procedure depthcue(mode: Boolean);
```

DESCRIPTION

depthcue sets the current depth-cue mode. If *mode* is TRUE(1) all lines, points, characters, and polygons that the system draws are depth cued. This means the z values and the range of color indices specified by **shaderange** or **RGBrange** determine the color of the lines, points, characters, or polygons. The z values, whose range is set by **setdepth**, are mapped linearly into the range of color indices. In this mode, lines that vary greatly in z value span the range of colors specified by **shaderange**.

For depth cueing to work properly, the color map locations **shaderange** specifies must be loaded with a series of colors that gradually increase or decrease in intensity.

SEE ALSO

setdepth, shaderange,

Programming Guide, Section 13.2, Depth-Cueing

NAME

devport – assigns a serial port to an external graphics device

SPECIFICATION

C
```
devport(dev,portno)
Device dev;
long portno;
```

FORTRAN
```
subroutine devpor(dev,portno)
integer*4 dev, portno
```

Pascal
```
procedure devport(dev, portno: longint);
```

DESCRIPTION

There are four serial ports on the back panel of the IRIS, ports 0 through 3. Port 0 connects the keyboard to the IRIS system. The remaining ports connect additional terminals or graphics peripherals, such as a dial and button box or a digitizing tablet.

The system software assumes that a dial and button box or a digitizing tablet will be connected to port 3. However, if both a dial and button box and a digitizing tablet are used with the IRIS, you must tell the system which port is being used for each device. **devport** assigns a serial port to the specified device. You must specify a device number for the peripheral and a port number between 1 and 3. (See Appendix A for device numbers.) For example, **devport(DIAL0,2)** followed by **devport(BPAD0,3)** indicates the dial and button box is connected to port 2 and the digitizing tablet is connected to port 3.

SEE ALSO

The manual page for the following routine is located in the *Unix Programmer's Manual,* Volume I: devport(1)

NOTE

Use this routine before using any input peripherals.

NAME

dopup – displays the specified pop-up menu

SPECIFICATION

C
```
long dopup(pup)
long pup;
```

FORTRAN
```
integer*4 function dopup(pup)
integer*4 pup
```

Pascal
```
procedure dopup(pup: longint): longint;
```

DESCRIPTION

dopup displays the specified pop-up menu until the user makes a selection. If the calling program has the input focus, the menu is displayed and **dopup** returns the value resulting from the item selection. The value can be returned by a submenu, a function, or a number bound directly to an item. If no selection is made, **dopup** returns –1.

Item selection is performed by either selecting or rolling off the side of a menu item. When the menu is defined, **defpup** or **addtopup** specify the list of menu entries and their corresponding actions. See **defpup** for details.

SEE ALSO

addtopup, defpup, freepup, newpup

Using mex , Chapter 3, Making Pop-Up Menus

NOTE

This routine is available only in immediate mode under the window manager.

NAME

doublebuffer – sets the display mode to double buffer mode

SPECIFICATION

C **doublebuffer()**

FORTRAN **subroutine double**

Pascal **procedure doublebuffer;**

DESCRIPTION

doublebuffer sets the display mode to double buffer mode. It does not take effect until **gconfig** is called. In double buffer mode, the bitplanes are partitioned into two groups, the front bitplanes and the back bitplanes. Double buffer mode displays only the front bitplanes. Drawing routines normally update only the back bitplanes; **frontbuffer** and **backbuffer** can override the default.

gconfig sets **frontbuffer** = FALSE (0) and **backbuffer** = TRUE (1) in double buffer mode.

SEE ALSO

backbuffer, frontbuffer, gconfig, getbuffer, getdisplaymode, RGBmode, singlebuffer, swapbuffers

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

NAME

draw – draws a line

SPECIFICATION

C

```
draw(x, y, z)
Coord x, y, z;

drawi(x, y, z)
Icoord x, y, z;

draws(x, y, z)
Scoord x, y, z;

draw2(x, y)
Coord x, y;

draw2i(x, y)
Icoord x, y;

draw2s(x, y)
Scoord x, y;
```

FORTRAN

```
subroutine draw(x, y, z)
real x, y, z

subroutine drawi(x, y, z)
integer*4 x, y, z

subroutine draws(x, y, z)
integer*2 x, y, z

subroutine draw2(x, y)
real x, y

subroutine draw2i(x, y)
integer*4 x, y

subroutine draw2s(x, y)
integer*2 x, y
```

Pascal **procedure draw(x, y, x: Coord);**
procedure drawi(x, y, z: Icoord);:
procedure draws(x, y, z: Scoord);
procedure draw2(x, y: Coord);
procedure draw2i(x, y: Icoord);
procedure draw2s(x, y: Scoord);

DESCRIPTION

draw connects the point *x*, *y*, *z* and the current graphics position with a line segment. It uses the current linestyle, linewidth, color (if in depth-cue mode, the depth-cued color is used), and writemask.

draw updates the current graphics position to the specified point.

Do not place routines that invalidate the current graphics position within sequences of moves and draws.

SEE ALSO

pnt, rdr, rmv, move

Programming Guide, Section 3.4, Lines

NAME

editobj – opens an object for editing

SPECIFICATION

C
```
editobj(obj)
Object obj;
```

FORTRAN
```
subroutine editob(obj)
integer*4 obj
```

Pascal
```
procedure editobj(obj: Object);
```

DESCRIPTION

editobj opens an object for editing. A pointer acts as a cursor that appends new routines. The pointer is initially set to the end of the object. The system appends graphics routines to the object until either a **closeobj** or a pointer positioning routine (**objdelete, objinsert,** or **objreplace**) executes. Usually, you need not be concerned about storage allocation. Objects grow and shrink automatically as routines are added and deleted.

If **editobj** specifies an undefined object, the system displays an error message.

SEE ALSO

compactify, objdelete, objinsert, objreplace

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

endfeedback – turns off feedback mode

SPECIFICATION

C
```
long endfeedback(buffer)
short buffer[];
```

FORTRAN
```
integer*4 function endfee(buffer)
integer*2 buffer(*)
```

Pascal
```
function endfeedback(var buffer: Short): longint;
```

DESCRIPTION

endfeedback turns off feedback mode. *buffer* contains the values output by the Geometry Pipeline during the feedback session. **endfeedback** returns the number of shorts in *buffer*.

SEE ALSO

feedback

Programming Guide, Section 10.2, Feedback Mode

NOTE

This routine is available only in immediate mode.

NAME

endfullscrn – ends full-screen mode

SPECIFICATION

C **endfullscrn()**

FORTRAN **subroutine endful**

Pascal **procedure endfullscrn;**

DESCRIPTION

endfullscrn ends full-screen mode and returns the screenmask and viewport to the boundaries of the current graphics window. **endfullscrn** leaves the current transformation unchanged.

SEE ALSO

endpupmode, fullscrn, pupmode

Using Mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

endpick – turns off picking mode

SPECIFICATION

C
```
long endpick(buffer)
short buffer[];
```

FORTRAN
```
integer*4 function endpic(buffer)
integer*2 buffer(*)
```

Pascal
```
function endpick(var buffer: Short): longint;
```

DESCRIPTION

endpick turns off picking mode. When a drawing routine draws in the picking region, the contents of the name stack are stored in *buffer*, along with the number of names in the stack. For example, if the name stack contained 5, 9, 17 when a hit occurred, the numbers 3, 5, 9, 17 are added to the buffer. The magnitude of the value returned by **endpick** is the number of such name lists in the buffer. If the value returned is positive, then all hits are recorded in the name lists; if it is negative, the buffer is not large enough to hold all the hits that occurred.

SEE ALSO

clearhitcode, gethitcode, initnames, loadname, pick

Programming Guide, Section 9.2, Picking

NOTE

This routine is available only in immediate mode.

BUGS

When using a debugger, do not stop the graphics between calls to **pick** and **endpick** because the graphics are frozen and the results cannot appear on the screen.

NAME

endpupmode – ends pop-up mode

SPECIFICATION

C **endpupmode()**

FORTRAN **subroutine endpup**

Pascal **procedure endpupmode;**

DESCRIPTION

endpupmode disables writing to the pop-up menu bitplanes.

SEE ALSO

endfullscrn, fullscrn, pupmode

Using mex, Chapter 3, Making Pop-Up Menus

NOTE

This routine is available only in immediate mode under the window manager.

NAME

endselect – turns off selecting mode

SPECIFICATION

C
```
long endselect(buffer)
short buffer[];
```

FORTRAN
```
integer*4 function endsel(buffer)
integer*2 buffer(*)
```

Pascal
```
function endselect(var buffer: Short): longint;
```

DESCRIPTION

endselect turns off selecting mode. The buffer stores any hits drawing routines generate between **gselect** and **endselect**. Every hit that occurs causes the entire contents of the name stack to be recorded in the buffer, preceded by the number of names in the stack. Thus, if the name stack contains 5, 9, 17 when a hit occurs, the numbers 3, 5, 9, 17 are added to the buffer. The magnitude of the value returned by **endselect** is the number of such name lists in the buffer. If the value returned is positive, then all hits are recorded in the name lists; if it is negative, the buffer is not large enough to hold all the hits that occurred.

SEE ALSO

clearhitcode, gethitcode, gselect, initnames, loadname

Programming Guide, Section 9.3, Selecting

NOTE

This routine is available only in immediate mode.

BUGS

When using a debugger, do not stop the graphics between **gselect** and **endselect** because the graphics are frozen and the results cannot appear on the screen.

NAME

feedback – turns on feedback mode

SPECIFICATION

C
```
feedback(buffer, size)
short buffer[];
long size;
```

FORTRAN
```
subroutine feedba(buffer, size)
integer*2 buffer(*)
integer*4 size
```

Pascal
```
procedure feedback(var buffer: Short; size: longint);
```

DESCRIPTION

feedback puts the system in feedback mode. In feedback mode, *buffer* stores the output of the Geometry Pipeline rather than sending it to the raster display system. *size* specifies the maximum number of values that *buffer* can store. The system does not draw on the screen in feedback mode.

SEE ALSO

endfeedback

Programming Guide, Section 10.2, Feedback Mode

NOTE

This routine is available only in immediate mode.

NAME

finish – blocks the user process until the Geometry Pipeline is empty

SPECIFICATION

C **finish()**

FORTRAN **subroutine finish**

Pascal **procedure finish;**

DESCRIPTION

finish blocks the host process until all previous routines execute. It forces all unsent routines the Geometry Pipeline to the bitplanes. Then, it sends a final token and blocks the process until the token goes through the pipeline and an acknowledgment has been sent.

finish is useful when there are network and pipeline delays.

SEE ALSO

gflush

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

font – selects a raster font for drawing text strings

SPECIFICATION

C
```
font(fntnum)
short fntnum;
```

FORTRAN
```
subroutine font(fntnum)
integer*4 fntnum
```

Pascal
```
procedure font(fntnum: Short);
```

DESCRIPTION

font selects the raster font that **charstr** uses when it draws a text string. *fntnum* is an index into the font table that **defrasterfont** builds. This font remains in effect until another **font** executes. Font 0 is the default.

If **font** specifies a font number that is not defined, the system selects font 0.

SEE ALSO

charstr, defrasterfont, getdescender, getfont, getheight, strwidth

Programming Guide, Section 5.3, Fonts

NAME

foreground – keeps a graphical process in the foreground

SPECIFICATION

C **foreground()**

FORTRAN **subroutine foregr**

Pascal **procedure foreground**

DESCRIPTION

Call **foreground** before calling **winopen** or **getport**. It keeps the process in the foreground, so that you can interact with it from the keyboard. **winopen** normally runs a process in the background. When the process is in the foregound, it interacts in the usual way with the UNIX input/output routines.

SEE ALSO

getport, winopen

Using mex , Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

freepup – returns a menu and its data structures to the system

SPECIFICATION

C
```
freepup(pup)
long pup;
```

FORTRAN
```
subroutine freepu(pup)
integer*4 pup
```

Pascal
```
procedure freepup(pup: longint);
```

DESCRIPTION

freepup returns the data structures associated with a pop-up menu to the system, making more memory available.

SEE ALSO

addtopup, defpup, dopup, newpup

Using mex, Chapter 3, Making Pop-Up Menus

NOTE

This routine available only in immediate mode under the window manager.

NAME

frontbuffer – enables updating in the front buffer

SPECIFICATION

C
```
frontbuffer(b)
Boolean b;
```

FORTRAN
```
subroutine frontb(b)
logical b
```

Pascal
```
procedure frontbuffer(b: Boolean);
```

DESCRIPTION

frontbuffer enables updating in the front buffer. When the value of *b* is FALSE (0), (the default value), the front buffer is not enabled. When the value of *b* is TRUE (0), the front buffer is enabled. This routine is useful only in double buffer mode. It is ignored in single buffer mode.

gconfig sets **frontbuffer** to FALSE (0).

SEE ALSO

backbuffer, doublebuffer, getbuffer

Programming Guide, Section 6.1, Display Modes

NAME

fudge – specifies fudge values added to a graphics window

SPECIFICATION

C
```
fudge(xfudge, yfudge)
long xfudge, yfudge;
```

FORTRAN
```
subroutine fudge(xfudge, yfudge)
integer*4 xfudge, yfudge
```

Pascal
```
procedure fudge(xfudge, yfudge: longint);
```

DESCRIPTION

fudge specifies fudge values that are added to the dimensions of a graphics window when it is resized. Typically, you use **fudge** to create window borders. **fudge** is useful in conjunction with **stepunit** and **keepaspect.**

With **stepunit** the window size for integers *n* and *m* is:

width = xunit*n + xfudge
height = yunit*m + yfudge

With **keepaspect** the window size is (*w*, *h*) where:

(w–xfudge)*yaspect = (h–yfudge)*xaspect

Call **fudge** at the beginning of a graphics program that runs under the window manager. If you do not call **winopen**, or if the window manager is not running, the system ignores **fudge**.

SEE ALSO

keepaspect, stepunit, winopen

Using mex , Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

fullscrn – gives a program the entire screen as a window

SPECIFICATION

C **fullscrn()**

FORTRAN **subroutine fullsc()**

Pascal **procedure fullscrn;**

DESCRIPTION

fullscrn gives a program the entire screen as a window under the window manager. It makes the call:

viewport(0,XMAXSCREEN,0,YMAXSCREEN);

and sets up the default **ortho**, which **winopen** defines. **fullscrn** eliminates all protections that prevent graphics processes from drawing on each other. Use it with caution or a sense of humor.

SEE ALSO

endfullscrn, endpupmode, pupcolor, pupmode

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

gbegin – initializes the system without altering the color map

SPECIFICATION

C **gbegin()**

FORTRAN **subroutine gbegin**

Pascal **procedure gbegin;**

DESCRIPTION

gbegin initializes the graphics environment for the calling program as **ginit** does, however, **gbegin** does not change the color map. This is useful under the window manager because it does not interfere with other programs that use the current color map. **gbegin** also does not call **setcursor**.

Under the window manager, it is preferable to use **winopen** for initialization functions.

SEE ALSO

ginit, greset, winopen

Programming Guide, Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

NAME

gconfig – reconfigures the system

SPECIFICATION

C **gconfig()**

FORTRAN **subroutine gconfi**

Pascal **procedure gconfig;**

DESCRIPTION

gconfig sets the modes that you request. You must call **gconfig** for **doublebuffer**, **multimap**, **onemap**, **RGBmode**, and **singlebuffer** to take effect. After a **gconfig** call, writemask, color, cursor color, and cursor writemask are reset to their default values. The contents of the color map do not change.

SEE ALSO

doublebuffer, multimap, onemap, RGBmode, singlebuffer,

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

NAME

genobj – returns a unique integer for use as an object identifier

SPECIFICATION

C **Object genobj()**

FORTRAN **integer*4 function genobj()**

Pascal **function genobj: Object;**

DESCRIPTION

Object identifiers can be up to 31 bits and must be unique within a program. **genobj** generates unique 31-bit integer numbers. Be cautious if you use a combination of user-defined and **genobj**-defined numbers to generate object numbers. **genobj** will not generate an object name that is currently in use. If there is any question, use **isobj** before using your own numbers.

SEE ALSO

callobj, gentag, isobj, makeobj

Programming Guide, Section 8.1, Defining an Object

NOTE

This routine is available only in immediate mode.

NAME

gentag – returns a unique integer for use as a tag

SPECIFICATION

C **Tag gentag()**

FORTRAN **integer*4 function gentag()**

Pascal **function gentag: Tag;**

DESCRIPTION

gentag generates a unique integer to use as a tag. Tags must be unique within an object.

gentag provides a unique 31-bit integer tag which acts as a label. **gentag** generates unique tags, although if you later define a tag with the same value, the first tag is lost.

SEE ALSO

genobj, istag

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

getbackface – returns whether backfacing polygons will appear

SPECIFICATION

C **long getbackface()**

FORTRAN **integer*4 getbac()**

Pascal **function getbackface: longint;**

DESCRIPTION

getbackface returns the state of backfacing filled polygon removal. If backface removal is on, the system draws only those polygons that face the viewer. 1 indicates backfacing polygon removal is enabled; otherwise **getbackface** returns zero.

SEE ALSO

backface

Programming Guide, Section 12.2, Backfacing Polygon Removal

NOTE

This routine is available only in immediate mode.

NAME

getbuffer – indicates which buffers are enabled for writing

SPECIFICATION

C **long getbuffer()**

FORTRAN **integer*4 function getbuf()**

Pascal **function getbuffer: longint;**

DESCRIPTION

getbuffer indicates which buffers are enabled for writing in double buffer mode. 1, the default, indicates the back buffer is enabled; 2 indicates the front buffer is enabled; and 3 indicates that both buffers are enabled.

getbuffer returns 0 if both buffers are disabled or if the system is not in double buffer mode.

Value	Buffer Enabled	Symbolic Name
0	none	NOBUFFER
1	back buffer	BCKBUFFER
2	front buffer	FRNTBUFFER
3	both buffers	BOTHBUFFERS

SEE ALSO

backbuffer, doublebuffer, frontbuffer

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

NAME

getbutton – returns the state (up or down) of a button

SPECIFICATION

C	**Boolean getbutton(num)** **Device num;**
FORTRAN	**logical function getbut(num)** **integer*4 num**
Pascal	**function getbutton(num: Device): Boolean;**

DESCRIPTION

getbutton returns the state of the button numbered *num*. 0 indicates the button is up; 1 indicates it is down. If *num* is invalid, -1 is returned.

SEE ALSO

Programming Guide, Section 7.3, Polling a Device

NOTE

This routine is available only in immediate mode.

NAME

getcmmode – returns the current color map mode

SPECIFICATION

C **Boolean getcmmode()**

FORTRAN **logical function getcmm()**

Pascal **function getcmmode: Boolean;**

DESCRIPTION

getcmmode returns the current color map mode. 0 indicates multimap mode; 1 indicates onemap mode.

Value	Color Map Mode	Symbolic Name
0	multimap	CMAPMULTI
1	onemap	CMAPONE

SEE ALSO

multimap, onemap

Programming Guide, Section 6.2, Color Maps

NOTE

This routine is available only in immediate mode.

NAME

getcolor – returns the current color

SPECIFICATION

C **long getcolor()**

FORTRAN **integer*4 function getcol()**

Pascal **function getcolor: longint;**

DESCRIPTION

getcolor returns the current color. It is an index into the color map, and is meaningful in both single buffer and double buffer mode. **getcolor** is ignored in RGB mode.

SEE ALSO

color, doublebuffer, singlebuffer

Programming Guide, Section 6.3, Colors and Writemasks

NOTE

This routine is available only in immediate mode.

NAME

getcpos – returns the current character position

SPECIFICATION

C
```
getcpos(ix, iy)
Screencoord *ix, *iy;
```

FORTRAN
```
subroutine getcpo(ix, iy)
integer*2 ix, iy
```

Pascal
```
procedure getcpos(var ix, iy: Screencoord);
```

DESCRIPTION

getcpos returns the current character position.

SEE ALSO

getgpos

Programming Guide, Section 3.1, Current Drawing Positions

NOTE

This routine is available only in immediate mode.

NAME

getcursor – returns the cursor characteristics

SPECIFICATION

C
```
getcursor(index, color, wtm, b)
short *index;
Colorindex *color, *wtm;
Boolean *b;
```

FORTRAN
```
subroutine getcur(index, color, wtm, b)
integer*2 index, color, wtm
logical b
```

Pascal
```
procedure getcursor (var index: Short;
var color, wtm: Colorindex; var b: Boolean;
```

DESCRIPTION

getcursor returns four values: the cursor glyph (*index*); the color (*color*) and the writemask (*wtm*) of the glyph, and a boolean value (*b*) indicates whether the system automatically displays the cursor.

index, *color*, *wtm* and *b* are the addresses of locations where the cursor variables are returned.

The default is the glyph index 0 in the cursor table, displayed with the color 1, drawn in the first available bitplane, and automatically updated on each vertical retrace. This routine is undefined in RGB mode.

SEE ALSO

defcursor, setcursor

Programming Guide, Section 6.4, Cursors

NOTE

This routine is available only in immediate mode.

NAME

getdcm – indicates whether depth-cue mode is on or off

SPECIFICATION

C **Boolean getdcm()**

FORTRAN **logical function getdcm()**

Pascal **function getdcm: Boolean;**

DESCRIPTION

getdcm returns TRUE(1) if the system is in depth-cue mode and FALSE(0) if it is not.

SEE ALSO

depthcue

Programming Guide, Section 13.2, Depth-Cueing

NOTE

This routine is available only in immediate mode.

NAME

getdepth – returns the parameters of **setdepth**

SPECIFICATION

C
```
getdepth(near, far)
Screencoord *near, *far;
```

FORTRAN
```
subroutine getdep(near, far)
integer*2 near, far
```

Pascal
```
procedure getdepth(var near, far: Screencoord);
```

DESCRIPTION

getdepth returns the *near* and *far* parameters of **setdepth.**

SEE ALSO

setdepth

Programming Guide, Section 12.1, Z-Buffer Mode

NOTE

This routine is available only in immediate mode.

NAME

getdescender – returns the character characteristics

SPECIFICATION

C **long getdescender();**

FORTRAN **integer*4 function getdes()**

Pascal **function getdescender(); longint;**

DESCRIPTION

getdescender returns the value of the descender of the character in the current font that has the longest descender. The value returned is the number of pixels that the descender extends below the character's baseline.

Each character in a font is defined using a bitmap that is displayed relative to the current character position. Vertical placement of each character is done using the current character position as the baseline or the line on the page. The portion of a character that extends below the baseline is called a descender. The lowercase characters g and p typically have descenders.

SEE ALSO

getfont, getheight, strwidth

Programming Guide, Section 5.3, Fonts

NOTE

This routine is available only in immediate mode.

NAME

getdev – reads a list of valuators at one time

SPECIFICATION

C
```
getdev(n, devs, vals)
long n;
Device *devs;
short *vals;
```

FORTRAN
```
subroutine getdev(n, devs, vals)
integer*4 n
integer*2 devs(n), vals(n)
```

Pascal
```
procedure getdev(n: longint; var dev: data: Short);
```

DESCRIPTION

getdev allows up to 128 valuators and buttons to be input devices at one time. *n* specifies the number of devices. *devs* is an array of device number constants, such as MOUSEX, BPADX, LEFTMOUSE, etc. *vals* returns the state of each device in the corresponding location.

SEE ALSO

getvaluator

Programming Guide, Section 7.3, Polling a Device

NAME

getdisplaymode – returns the current display mode

SPECIFICATION

C **long getdisplaymode()**

FORTRAN **integer*4 function getdis()**

Pascal **function getdisplaymode: longint;**

DESCRIPTION

getdisplaymode returns the current display mode. 0 indicates RGB single buffer mode; 1 indicates color map, single buffer mode; 2 indicates color map, double buffer mode.

Value	Display Mode	Symbolic Name
0	RGB single buffer mode	DMRGB
1	color map single buffer mode	DMSINGLE
2	color map double buffer mode	DMDOUBLE

SEE ALSO

doublebuffer, RGBmode, singlebuffer

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

NAME

getfont – returns the current raster font number

SPECIFICATION

C	**long getfont()**
FORTRAN	**integer*4 function getfon()**
Pascal	**function getfont: longint;**

DESCRIPTION

getfont returns the index of the current raster font.

SEE ALSO

defrasterfont, font

Programming Guide, Section 5.3, Fonts

NOTE

This routine is available only in immediate mode.

NAME

getgpos – returns the current graphics position

SPECIFICATION

C
```
getgpos(fx, fy, fz, fw)
Coord *fx, *fy, *fz, *fw;
```

FORTRAN
```
subroutine getgpo(fx, fy, fz, fw)
real fx, fy, fz, fw
```

Pascal
```
procedure getgpos(var fx, fy, fz, fw: Coord);
```

DESCRIPTION

getgpos returns the current graphics position after transformation by the current matrix.

SEE ALSO

getcpos

Programming Guide, Section 3.1, Current Drawing Positions

NOTE

This routine is available only in immediate mode.

NAME

getheight – returns the maximum character height in the current raster font

SPECIFICATION

C **long getheight()**

FORTRAN **integer*4 function gethei()**

Pascal **function getheight: longint;**

DESCRIPTION

getheight returns the maximum height of the characters, which **defrasterfont** defines, in the current raster font, including ascenders (in characters such as t and h) and descenders (such as y and p). The height is usually the number of pixels between the top of the tallest ascender and the bottom of the lowest descender.

SEE ALSO

getdescender, getfont, strwidth

Programming Guide, Section 5.3, Fonts

NOTE

This routine is available only in immediate mode.

NAME

gethitcode – returns the current system hitcode

SPECIFICATION

C **long gethitcode()**

FORTRAN **integer*4 function gethit()**

Pascal **function gethitcode: longint;**

DESCRIPTION

gethitcode returns the global variable *hitcode*, which keeps a cumulative record of clipping plane hits. It does not change the hitcode value.

The hitcode is a 6-bit number (see below), with one bit for each clipping plane.

5	4	3	2	1	0
far	near	top	bottom	right	left

SEE ALSO

clearhitcode, gselect, pick

Programming Guide, Section 9.2, Picking

NOTE

This routine is available only in immediate mode.

NAME

getlsbackup – returns the current value of the linestyle backup flag

SPECIFICATION

C **Boolean getlsbackup()**

FORTRAN **logical function getlsb()**

Pascal **function getlsbackup: Boolean;**

DESCRIPTION

getlsbackup returns the current value of the linestyle backup flag. TRUE(1) indicates **lsbackup** is on, and the last three pixels of a line are colored regardless of the linestyle pattern. FALSE(0) indicates **lsbackup** is off and the pattern determines the state of all the pixels in the line.

SEE ALSO

lsbackup

Programming Guide, Section 5.1, Linestyles

NOTE

This routine is available only in immediate mode.

NAME

getlsrepeat – returns the linestyle repeat count

SPECIFICATION

C	**long getlsrepeat()**
FORTRAN	**integer*4 function getlsr()**
Pascal	**function getlsrepeat: longint;**

DESCRIPTION

getlsrepeat returns the current linestyle repeat factor, which is set by **lsrepeat.**

SEE ALSO

lsrepeat

Programming Guide, Section 5.1, Linestyles

NOTE

This routine is available only in immediate mode.

NAME

getlstyle – returns the current linestyle

SPECIFICATION

C **long getlstyle()**

FORTRAN **integer*4 function getlst()**

Pascal **function getlstyle: longint;**

DESCRIPTION

getlstyle returns the current linestyle. The returned value is an index into the linestyle table.

SEE ALSO

deflinestyle, setlinestyle

Programming Guide, Section 5.1, Linestyles

NOTE

This routine is available only in immediate mode.

NAME

getlwidth – returns the current linewidth

SPECIFICATION

C	**long getlwidth()**
FORTRAN	**integer*4 function getlwi()**
Pascal	**function getlwidth: longint;**

DESCRIPTION

getlwidth returns the current linewidth in pixels.

SEE ALSO

linewidth

Programming Guide, Section 5.1, Linestyles

NOTE

This routine is available only in immediate mode.

NAME

getmap – returns the number of the current color map

SPECIFICATION

C **long getmap()**

FORTRAN **integer*4 function getmap()**

Pascal **function getmap: longint;**

DESCRIPTION

getmap returns the number (from 0 to 15) of the current color map. In onemap mode, **getmap** returns zero.

SEE ALSO

multimap, onemap

Programming Guide, Section 6.6, Onemap and Multimap Modes

NOTE

This routine is available only in immediate mode.

NAME

getmatrix – returns the current transformation matrix

SPECIFICATION

C
```
getmatrix(m)
Matrix m;
```

FORTRAN
```
subroutine getmat(m)
real m(4,4)
```

Pascal
```
procedure getmatrix(var m: Matrix);
```

DESCRIPTION

getmatrix copies the transformation matrix from the top of the stack to a user-specified array. The matrix stack does not change.

SEE ALSO

loadmatrix, multmatrix, popmatrix, pushmatrix

Programming Guide, Section 4.5, User-Defined Transformations

NOTE

This routine is available only in immediate mode.

NAME

getmcolor – returns a color map entry

SPECIFICATION

C
```
getmcolor(i, red, green, blue)
Colorindex i;
short *red, *green, *blue;
```

FORTRAN
```
subroutine getmco(i, red, green, blue)
integer*4 i
integer*2 red, green, blue
```

Pascal
```
procedure getmcolor(color: Colorindex; var r, g, b:
        Short);
```

DESCRIPTION

getmcolor returns the red, green, and blue components of a color map entry.

SEE ALSO

mapcolor

Programming Guide, Section 6.2, Color Maps

NOTE

This routine is available only in immediate mode.

NAME

getmem – returns the amount of available memory

SPECIFICATION

C	**long getmem()**
FORTRAN	**integer*4 function getmem()**
Pascal	**function getmem: longint;**

DESCRIPTION

getmem returns the amount of available memory left on the system. On a workstation with virtual memory up to 14 megabytes, it returns 14 megabytes minus the amount that has been used. On a terminal, it returns the amount of free physical memory.

SEE ALSO

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

getmonitor – returns the current display monitor

SPECIFICATION

C **long getmonitor()**

FORTRAN **integer*4 function getmon()**

Pascal **function getmonitor: longint;**

DESCRIPTION

getmonitor returns the type of the current display monitor. Possible display monitors include 50Hz noninterlaced, 60Hz noninterlaced, 30Hz interlaced, NTSC, and PAL. The file *get.h* defines values for HZ30, HZ50, HZ60, NTSC, and PAL.

SEE ALSO

getothermonitor, setmonitor

Programming Guide, Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

NAME

getopenobj – returns the current open object

SPECIFICATION

C	**Object getopenobj()**
FORTRAN	**integer*4 function getope()**
Pascal	**function getopenobj: Object;**

DESCRIPTION

getopenobj returns the number of the object that is currently open for editing. If no object is open, it returns -1.

SEE ALSO

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

getorigin – returns the position of a graphics window

SPECIFICATION

C
```
getorigin(x, y)
long *x, *y;
```

FORTRAN
```
subroutine getori(x, y)
integer*4 x, y
```

Pascal
```
procedure getorigin(var x, y: longint);
```

DESCRIPTION

getorigin returns the position of the lower-left corner of a graphics window. Call **getorigin** after graphics initialization.

When the window manager is not running, **getorigin** returns (0, 0).

SEE ALSO

getport, getsize, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

getothermonitor – returns the nondisplayed monitor type

SPECIFICATION

C **long getothermonitor()**

FORTRAN **integer*4 function getoth()**

Pascal **function getothermonitor: longint;**

DESCRIPTION

getothermonitor returns the nondisplayed monitor type. Possible display monitors include 50Hz noninterlaced, 60Hz noninterlaced, 30Hz interlaced, NTSC, and PAL. The file *get.h* defines values for HZ30, HZ50, HZ60, NTSC, and PAL. It complements **getmonitor**.

SEE ALSO

getmonitor, setmonitor

Programming Guide, Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

NAME

getpattern – returns the index of the current pattern

SPECIFICATION

C **long getpattern()**

FORTRAN **integer*4 function getpat**

Pascal **function getpattern: longint;**

DESCRIPTION

getpattern returns the index of the current pattern from the table of available patterns.

SEE ALSO

defpattern, setpattern

Programming Guide, Section 5.2, Patterns

NOTE

This routine is available only in immediate mode.

NAME

getplanes – returns the number of available bitplanes

SPECIFICATION

C **long getplanes()**

FORTRAN **integer*4 function getpla()**

Pascal **function getplanes: longint;**

DESCRIPTION

getplanes returns the number of bitplanes that are available in the system.

SEE ALSO

doublebuffer, multimap, onemap, RGBmode, singlebuffer

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

NAME

getport – creates a graphics window under the window manager

SPECIFICATION

C
```
getport(arg)
char arg[];
```

FORTRAN
```
subroutine getpor(arg, length)
character*(*) arg
integer*4 length
```

Pascal
```
procedure getport(arg: pstring128);
```

DESCRIPTION

getport does a graphics initialization and creates a graphics window according to the constraints specified in **minsize, maxsize, keepaspect, prefsize, prefposition, stepunit, fudge, noport,** and **foreground.** After they are bound to the graphics window, these constraints are reset to zero.

If no constraints are specified or if the description is incomplete, the window manager prompts for the missing information. If you use the cursor to show the size and location of the graphics window, **getport** puts the graphics program in the background, unless **foreground** is called before **getport.**

When you call **getport**, the pseudo devices INPUTCHANGE and REDRAW are automatically queued.

When you call **getport** outside the window manager, it creates a standard-size graphics window: (xmin 0, xmax 1023, ymin 0, ymax 767).

arg has no function at this time.

In FORTRAN, there is an extra argument, *length,* which is the number of characters in the name string.

SEE ALSO

foreground, fudge, keepaspect, maxsize, minsize, noport, prefposition, prefsize, stepunit, winconstraints, winopen

Using mex , Chapter 2, Programming with mex

NOTE

winopen is preferred over **getport** for manipulating graphics windows.

This routine is available only in immediate mode.

NAME

getresetls – returns the current value of **resetls**

SPECIFICATION

C **long getresetls()**

FORTRAN **logical function getres()**

Pascal **function getresetls: Boolean;**

DESCRIPTION

returns the current value of the reset linestyle flag. TRUE(1), the default value, indicates the linestyle is reinitialized for each line segment. FALSE(0) indicates the pattern rotates continuously across line segment boundaries.

SEE ALSO

resetls

Programming Guide, Section 5.1, Linestyles

NOTE

This routine is available only in immediate mode.

NAME

getscrmask – returns the current screenmask

SPECIFICATION

C
```
getscrmask(left, right, bottom, top)
Screencoord *left, *right, *bottom, *top;
```

FORTRAN
```
subroutine getscr(left, right, bottom, top)
integer*2 left, right, bottom, top
```

Pascal
```
procedure getscrmask(var left, right, bottom, top:
        Screencoord);
```

DESCRIPTION

getscrmask returns the screen coordinates of the current screenmask.

SEE ALSO

popviewport, pushviewport, scrmask

Programming Guide, Section 4.4, Viewports

NOTE

This routine is available only in immediate mode.

NAME

getshade – returns the current shade

SPECIFICATION

C	**long getshade()**
FORTRAN	**integer*4 function getsha()**
Pascal	**function getshade: longint;**

DESCRIPTION

getshade returns the current shading value. It is an index into the color map and is meaningful in color map mode but not in RGB mode.

SEE ALSO

setshade, spclos

Programming Guide, Section 13.1, Shading

NOTE

This routine is available only in immediate mode.

NAME

getsize – returns the size of a graphics window

SPECIFICATION

C
```
getsize(x, y)
long *x, *y;
```

FORTRAN
```
subroutine getsiz(x, y)
integer*4 x, y
```

Pascal
```
procedure getsize(var x, y: longint);
```

DESCRIPTION

getsize returns the dimensions (in pixels) of the graphics window used by a graphics program. Call **getsize** after **getport**.

SEE ALSO

getorigin, getport, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

gettp – returns the location of the current textport

SPECIFICATION

C
```
gettp(left, right, bottom, top)
Screencoord *left, *right, *bottom, *top;
```

FORTRAN
```
subroutine gettp(left, right, bottom, top)
integer*2 left, right, bottom, top
```

Pascal
```
procedure gettp(var left, right, bottom, top:
        Screencoord);
```

DESCRIPTION

gettp returns the location of the current texport in screen coordinates.

SEE ALSO

textport

Programming Guide, Section 14, Textports

NOTE

This routine is available only in immediate mode.

NAME

getvaluator – returns the current state of a valuator

SPECIFICATION

C
```
long getvaluator(dev)
Device dev;
```

FORTRAN
```
integer*4 function getval(dev)
integer*4 dev
```

Pascal
```
function getvaluator(dev: Device); longint;
```

DESCRIPTION

getvaluator returns the current value (an integer) of the valuator *dev*.

SEE ALSO

getbutton

Programming Guide, Section 7.3, Polling a Device

NOTE

This routine is available only in immediate mode

NAME

getviewport – returns the current viewport

SPECIFICATION

C **getviewport(left, right, bottom, top)**
Screencoord *left, *right, *bottom, *top;

FORTRAN **subroutine getvie(left, right, bottom, top)**
integer*2 left, right, bottom, top

Pascal **procedure getviewport(var left, right, bottom, top:**
Screencoord);

DESCRIPTION

getviewport returns the current viewport and reads the top of the viewport stack. *left, right, bottom, top* are the addresses of four memory locations. These are assigned the left, right, bottom, and top coordinates of the viewport.

SEE ALSO

popviewport, pushviewport, viewport

Programming Guide, Section 4.4, Viewports

NOTE

This routine is available only in immediate mode.

NAME

getwritemask – returns the current writemask

SPECIFICATION

C **long getwritemask()**

FORTRAN **integer*4 function getwri()**

Pascal **function getwritemask: longint;**

DESCRIPTION

getwritemask returns the current writemask. It is an integer with up to 12 significant bits, one for each available bitplane. This routine is undefined in RGB mode.

SEE ALSO

RGBwritemask, writemask

Programming Guide, Section 6.3, Colors and Writemasks

NOTE

This routine is available only in immediate mode.

NAME

getzbuffer – indicates whether z-buffering is on or off

SPECIFICATION

C **long getzbuffer()**

FORTRAN **logical function getzbu()**

Pascal **function getzbuffer: Boolean;**

DESCRIPTION

getzbuffer returns the current value of the z-buffer flag. FALSE (0), the default value, indicates that z-buffering is off. TRUE (1), indicates that z-buffering is on.

SEE ALSO

setdepth, zbuffer, zclear

Programming Guide, Section 12.1, Z-Buffer Mode

NOTE

This routine is available only in immediate mode.

NAME

gexit – terminates a program

SPECIFICATION

C **gexit()**

FORTRAN **subroutine gexit**

Pascal **procedure gexit;**

DESCRIPTION

gexit is the final graphics routine in a program. It waits for the Geometry Pipeline to empty.

SEE ALSO

finish, ginit, greset, winopen

Programming Guide, Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

NAME

gflush – forces all unsent routines down the network

SPECIFICATION

C **gflush()**

FORTRAN **subroutine gflush**

Pascal **procedure gflush;**

DESCRIPTION

At the host, communication software buffers most graphics routines for efficient block transfer of data to the IRIS. **gflush** delivers all buffered, untransmitted graphics data to the IRIS. Certain graphics routines (notably those that return values) flush the host buffer when they execute. Use **gflush** only on an IRIS terminal.

SEE ALSO

finish

Programming Guide , Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

gflush has no effect if it is run locally.

NAME

ginit – initializes the system

SPECIFICATION

C **ginit()**

FORTRAN **subroutine ginit**

Pascal **procedure ginit;**

DESCRIPTION

ginit initializes the hardware, allocates memory for symbol tables and display lists, and sets up default values for global state attributes (as **greset** does). It has no arguments. **winopen** performs the same function as **ginit** under the window manager.

Call **ginit** once, before any other Graphics Library routine.

SEE ALSO

gbegin, gexit, greset, winopen

Programming Guide, Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

NAME

greset – resets all global state attributes to their initial values

SPECIFICATION

C **greset()**

FORTRAN **subroutine greset**

Pascal **procedure greset;**

DESCRIPTION

greset resets all global state attributes to their initial values; you can call these attributes at any time. See the following table for a listing of global state attributes.

Attribute	Initial Value
available bitplanes	all bitplanes[1]
backface mode	off
blinking	turned off
color	undefined
color map mode	one map
cursor	0 (arrow)[2]
depthcue mode	off
display mode	single buffer
font	0[3]
linestyle	0 (solid)
linestyle backup	off
linewidth	1 pixel
lsrepeat	1
pattern	0 (solid)
picking size	10×10 pixels
reset linstyle	on
RGB color	undefined
RGB writemask	undefined
shaderanges	0,7,0,1023
viewport	entire screen
writemask	all planes enabled[1]
zbuffer mode	off

1. If there are more than 3 bitplane boards installed, the number of available bitplanes is set to 12.
2. The color is set to 1 and the writemask and cursor are set to 0xfff.
3. Rasterfont 0 is a Helvetica-like font.

greset puts a 2-D orthographic projection transformation on the matrix stack with left, right, bottom, and top set to the boundaries of the screen. It also turns on the cursor; ties it to MOUSEX and MOUSEY; and unqueues all buttons, valuators, and the keyboard. Each button is set to FALSE and untied from valuators. Each valuator is set to XMAXSCREEN/2; the range is 0..XMAXSCREEN. MOUSEY is set to YMAXSCREEN/2 and has range 0..YMAXSCREEN.

greset also defines certain entry in the color map, as follows:

Index	Name	RGB Value		
		Red	*Green*	*Blue*
0	BLACK	0	0	0
1	RED	255	0	0
2	GREEN	0	255	0
3	YELLOW	255	255	0
4	BLUE	0	0	255
5	MAGENTA	255	0	255
6	CYAN	0	255	255
7	WHITE	255	255	255
all others	unnamed		undefined	

SEE ALSO

gbegin, ginit, winopen

Programming Guide, Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

NAME

gRGBcolor – returns the current RGB value

SPECIFICATION

C
```
gRGBcolor(red, green, blue)
short *red, *green, *blue;
```

FORTRAN
```
subroutine gRGBco(red, green, blue)
integer*2 red, green, blue
```

Pascal
```
procedure gRGBcolor(var red, green, blue: Short);
```

DESCRIPTION

gRGBcolor returns the current RGB color values. *red, green, blue* are the addresses of three locations that are filled with the red, green, and blue values. The system must be in RGB mode when **gRGBcolor** executes.

SEE ALSO

RGBcolor, RGBmode

Programming Guide, Section 6.3, Colors and Writemasks

NOTE

This routine is available only in immediate mode.

NAME

gRGBcursor – returns the cursor characteristics in RGB mode

SPECIFICATION

C

```
gRGBcursor(index, red, green, blue, redm,
        greenm, bluem, b)
short *index, *red, *green, *blue, *redm,
        *greenm, *bluem;
Boolean *b;
```

FORTRAN

```
subroutine gRGBcu(index, red, green, blue,
        redm, greenm, bluem, b)
integer*2 index, red, green, blue, redm, greenm,
        bluem logical b
```

Pascal

```
procedure gRGBcursor(var index, red, green, blue,
        redm, greenm, bluem: Short; var b: Boolean);
```

DESCRIPTION

gRGBcursor returns the seven parameters of the last executed **RGBcursor** . The parameters are *index, red, green, blue, redm, greenm,* and *bluem.* **gRGBcursor** also returns a boolean *b* that is TRUE(1) if the automatic cursor is on. The system must be in RGB mode when **gRGBcursor** executes.

SEE ALSO

RGBcursor

Programming Guide, Section 6.4, Cursors

NOTE

This routine is available only in immediate mode.

NAME

gRGBmask – returns the current RGB writemask

SPECIFICATION

C
```
gRGBmask(redm, greenm, bluem)
short *redm, *greenm, *bluem;
```

FORTRAN
```
subroutine gRGBma(redm, greenm, bluem)
integer*2 redm, greenm, bluem
```

Pascal
```
procedure gRGBmask(var redm, greenm, bluem:
        Short);
```

DESCRIPTION

gRGBmask returns the current RGB writemask as three 8-bit masks. **gRGBmask** places masks in the low order 8-bits of the locations *redm, greenm,* and *bluem* address. The system must be in RGB mode when this routine executes.

SEE ALSO

getwritemask, RGBwritemask

Programming Guide, Section 6.3, Colors and Writemasks

NOTE

This routine is available only in immediate mode.

NAME

gselect – puts the system in selecting mode

SPECIFICATION

C
```
gselect(buffer, numnames)
short buffer[];
long numnames;
```

FORTRAN
```
subroutine gselect(buffer, numnam)
integer*2 buffer(1)
integer*4 numnam
```

Pascal
```
procedure gselect(var buffer: Short; numnames:
        longint);
```

DESCRIPTION

gselect turns on selecting mode. The current viewing matrix defines the selecting region when **gselect** executes. However, you can construct a viewing matrix after selecting mode has begun. **gselect** and **pick** are identical except **gselect** allows you to create a viewing matrix in selection mode.

numnames specifies the maximum number of names the system saves. Names are 16-bit numbers, which you load on the name stack. Each drawing routine that intersects the selecting region causes the contents of the name stack to be stored in *buffer*.

SEE ALSO

endpick, endselect, initnames, loadname, pick, picksize, popname, pushname

Programming Guide, Section 9.3, Selecting

NOTE

This routine is available only in immediate mode.

BUGS

When using a debugger, do not stop the graphics between **gselect** and **endselect** because the graphics are frozen and results cannot appear on the screen.

NAME

gsync – waits for a vertical retrace period

SPECIFICATION

C **gsync()**

FORTRAN **subroutine gsync**

Pascal **procedure gsync;**

DESCRIPTION

In single buffer mode, rapidly changing scenes should be synchronized with the refresh rate. **gsync** waits for the next vertical retrace period.

SEE ALSO

RGBmode, singlebuffer

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

NAME

imakebackground – registers the screen background process

SPECIFICATION

C **imakebackground()**

FORTRAN **subroutine imakeb**

Pascal **procedure imakebackground;**

DESCRIPTION

imakebackground registers a process that maintains the screen background. Call it before **winopen**. The process draws the background and reads the input queue. Every time the process gets a REDRAW token in the queue, it redraws the background.

SEE ALSO

winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

initnames – initializes the name stack

SPECIFICATION

C **initnames()**

FORTRAN **subroutine initna**

Pascal **procedure initnames;**

DESCRIPTION

initnames clears the name stack for picking and selecting. **initnames** is ignored outside of picking and selecting mode.

SEE ALSO

gselect, pick

Programming Guide, Section 9.2, Picking

NAME

ismex – returns TRUE if the window manager is running

SPECIFICATION

C **Boolean ismex()**

FORTRAN **logical function ismex()**

Pascal **function ismex: Boolean;**

DESCRIPTION

ismex returns TRUE(1) if the window manager is running and FALSE(0) otherwise.

SEE ALSO

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

isobj – indicates whether a given object number identifies an object

SPECIFICATION

C
```
long isobj(obj)
Object obj;
```

FORTRAN
```
logical function isobj(obj)
integer*4 obj
```

Pascal
```
function isobj(obj: Object): Boolean;
```

DESCRIPTION

isobj returns TRUE(1) if *obj* is an object number and FALSE(0) if it is not. After a **makeobj** call for the object, **isobj** returns TRUE(1).

SEE ALSO

genobj, istag, makeobj

Programming Guide, Section 8.1, Defining An Object

NOTE

This routine is available only in immediate mode.

NAME

isqueued – indicates if the specified device is queued

SPECIFICATION

C
```
Boolean isqueued(dev)
Device dev;
```

FORTRAN
```
logical function isqueu(dev)
integer*4 dev
```

Pascal
```
function isqueued(dev: Device): Boolean;
```

DESCRIPTION

isqueued returns TRUE(1) if the specified device is enabled for queueing, and FALSE(0) if it is not enabled.

SEE ALSO

qdevice, qread, unqdevice,

Programming Guide, Section 7.4, The Event Queue

NAME

istag – indicates if a given tag is used within the current open object

SPECIFICATION

C
```
Boolean istag(t)
Tag t;
```

FORTRAN
```
logical function istag(t)
integer*4 t
```

Pascal
```
function istag(t: Tag): Boolean;
```

DESCRIPTION

istag returns TRUE (1) if *t* is used within the current open object and FALSE (0) if it is not. The result is undefined if there is no current open object.

SEE ALSO

gentag, isobj

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

keepaspect – specifies the aspect ratio of a graphics window

SPECIFICATION

C
```
keepaspect(x, y)
long x, y;
```

FORTRAN
```
subroutine keepas(x, y)
integer*4 x, y
```

Pascal
```
procedure keepaspect(x, y: longint);
```

DESCRIPTION

keepaspect specifies the aspect ratio of a graphics window. Call it at the beginning of a graphics program. It takes effect when **winopen** is called. The resulting graphics window maintains the aspect ratio specified in **keepaspect,** even if it changes size. For example, **keepaspect(1, 1)** always results in a square graphics window. You can also call **keepaspect** in conjunction with **winconstraints** to modify the enforced aspect ratio after the window has been created.

If **winopen** is not called, or it the system is not running the window manager, **keepaspect** is ignored.

SEE ALSO

fudge, winconstraints, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

lampoff – turns off the keyboard display lights

SPECIFICATION

C
```
lampoff(lamps)
char lamps;
```

FORTRAN
```
subroutine lampof(lamps)
integer*4 lamps
```

Pascal
```
procedure lampoff(lamps: longint);
```

DESCRIPTION

lampoff turns off any combination of the four user-controlled lamps on the keyboard. The four low-order bits of *lamps* control lamps 1 through 4.

SEE ALSO

clkoff, clkon, lampon, ringbell, setbell

Programming Guide, Section 7.5, Controlling Peripheral Input/Output Devices

NOTE

This routine is available only in immediate mode.

NAME

lampon – turns on the keyboard display lights

SPECIFICATION

C
```
lampon(lamps)
char lamps;
```

FORTRAN
```
subroutine lampon(lamps)
integer*4 lamps
```

Pascal
```
procedure lampon (lamps: longint);
```

DESCRIPTION

lampon turns on any combination of the four user-controlled lamps on the keyboard. The four low-order bits of *lamps* control lamps 1 through 4.

SEE ALSO

clkon, clkoff, lampoff, ringbell, setbell

Programming Guide, Section 7.5, Controlling Peripheral Input/Output Devices

NOTE

This routine is available only in immediate mode.

NAME

linewidth – specifies the linewidth

SPECIFICATION

C
```
linewidth(n)
short n;
```

FORTRAN
```
subroutine linewi(n)
integer*4 n
```

Pascal
```
procedure linewidth(n: longint);
```

DESCRIPTION

linewidth specifies the width of a line. The width is the number of pixels in the *y* direction if the line is less than 45 degrees and in the *x* direction if it is greater than 45 degrees. A wide line is centered (as far as possible) around the true line. If **linewidth** is set to $n \neq 1$, **resetls** must be TRUE.

SEE ALSO

lsbackup, resetls, setlinestyle

Programming Guide, Section 5.1, Linestyles

NAME

loadmatrix – loads a transformation matrix

SPECIFICATION

C
```
loadmatrix(m)
Matrix m;
```

FORTRAN
```
subroutine loadma(m)
real m(4,4)
```

Pascal
```
procedure loadmatrix(var m: Matrix);
```

DESCRIPTION

loadmatrix loads a 4x4 floating point matrix onto the matrix stack; it replaces the current top matrix.

SEE ALSO

multmatrix, popmatrix, pushmatrix

Programming Guide, Section 4.5, User-Defined Transformations

NAME

loadname – loads the name on the top of the name stack

SPECIFICATION

C
```
loadname(name)
short name;
```

FORTRAN
```
subroutine loadna(name)
integer*4 name
```

Pascal
```
procedure loadname(name: longint);
```

DESCRIPTION

loadname replaces the top name in the name stack with a new 16-bit integer name. Each time a routine causes a hit in picking or selecting mode, the system stores the contents of the name stack in a buffer.

loadname is ignored outside of picking and selecting modes.

SEE ALSO

gselect, pick

Programming Guide, Section 9.2, Picking

NAME

lookat – defines a viewing transformation

SPECIFICATION

C
```
lookat(vx, vy, vz, px, py, pz, twist)
Coord vx, vy, vz, px, py, pz;
Angle twist;
```

FORTRAN
```
subroutine lookat(vx, vy, vz, px, py, pz, twist)
real vx, vy, vz, px, py, pz
integer*4 twist
```

Pascal
```
procedure lookat(vx, vy, vz, px py pz: Coord;
        twist: longint);
```

DESCRIPTION

lookat defines the viewpoint and a reference point on the line of sight in world coordinates. The viewpoint is at *(vx, vy, vz)*. The viewpoint and reference point *(px, py, pz)* define the line of sight. *twist* measures right-hand rotation about the z-axis in the eye coordinate system.

SEE ALSO

polarview

Programming Guide, Section 4.2, Viewing Transformations

NAME

lsbackup – controls whether the last two pixels of a line are colored

SPECIFICATION

C **lsbackup(b)**

Boolean b;

FORTRAN **subroutine lsback(b)**

logical b

Pascal **procedure lsbackup(b: Boolean);**

DESCRIPTION

lsbackup is one of two routines that modify the application of the linestyle pattern. If enabled, it overrides the current linestyle to guarantee that the last two pixels in a line are colored. It takes one argument (*b*), a boolean. If *b* is TRUE(1), backup mode is enabled. Set **resetls** to TRUE(1) when backup mode is enabled. If *b* is FALSE(0), the default setting, the linestyle is used as it stands, and lines can have invisible endpoints.

SEE ALSO

deflinestyle, getlsbackup, resetls, setlinestyle

Programming Guide, Section 5.1, Linestyles

NAME

lsrepeat – sets a repeat factor for the current linestyle

SPECIFICATION

C
```
lsrepeat(factor)
long factor;
```

FORTRAN
```
subroutine lsrepe(factor)
integer*4 factor
```

Pascal
```
procedure lsrepeat(factor: longint);
```

DESCRIPTION

lsrepeat sets a repeat factor for the current linestyle. When a line is drawn, pixels are turned on if there is a 1 in the corresponding position of the linestyle mask. For example, the mask 0x5555 specifies that alternate pixels are turned on (assuming the linestyle repeat factor is 1). If the repeat factor is *n*, then the 0x5555 pattern above would draw a line with *n* bits on and *n* bits off, alternately. The valid range of the repeat factor is 1 through 256.

SEE ALSO

getlsrepeat, setlinestyle

Programming Guide, Section 5.1, Linestyles

NAME

makeobj – creates an object

SPECIFICATION

C
```
makeobj(obj)
Object obj;
```

FORTRAN
```
subroutine makeob(obj)
integer*4 obj
```

Pascal
```
procedure makeobj(obj: Object);
```

DESCRIPTION

makeobj creates a graphics object. **makeobj** takes one argument, a 31-bit integer that is associated with the object. If *obj* is the number of an existing object, the contents of that object are deleted.

When **makeobj** executes, the object number is entered into a symbol table and memory is allocated for a display list. Subsequent graphics routines are compiled into the display list instead of executing.

SEE ALSO

callobj, chunksize, closeobj, genobj, isobj

Programming Guide, Section 8.1, Defining an Object

NOTE

This routine is available only in immediate mode.

NAME

maketag – numbers a routine in the display list

SPECIFICATION

C
```
maketag(t)
Tag t;
```

FORTRAN
```
subroutine maketa(t)
integer*4 t
```

Pascal
```
procedure maketag(t: Tag);
```

DESCRIPTION

Use **maketag** to explicitly number routines within an object. To do this, specify a 31-bit number (*t*) with **maketag**. The system assigns this number to the next routine in the display list. A tag is specific only to the object in which you use it. Consequently, you can use the same 31-bit number in different objects without confusion.

SEE ALSO

gentag, istag

Programming Guide, Section 8.3, Object Editing

NAME

mapcolor – changes a color map entry

SPECIFICATION

C
```
mapcolor(i, red, green, blue)
Colorindex i;
short red, green, blue;
```

FORTRAN
```
subroutine mapcol(i, red, green, blue
integer*4 i, red, green, blue
```

Pascal
```
procedure mapcolor(i: Colorindex; red, green,
        blue: Short);
```

DESCRIPTION

mapcolor changes a single color map entry to the specified RGB value. Its arguments include a color map index (*i*) and eight bits each of red, green, and blue intensities. Pixels written with *i* display the specified RGB (*red, green, blue*) intensities.

In multimap mode, only the current color map can be updated with **mapcolor**. The system ignores invalid indices.

SEE ALSO

multimap, onemap, setmap

Programming Guide, Section 6.2, Color Maps

NOTE

This routine is available only in immediate mode.

NAME

mapw – maps a point on the screen into a line in 3-D world coordinates

SPECIFICATION

C
```
mapw(vobj, sx, sy, wx1, wy1, wz1, wx2, wy2, wz2)
Object vobj;
Screencoord sx, sy;
Coord *wx1, *wy1, *wz1, *wx2, *wy2, *wz2;
```

FORTRAN
```
subroutine mapw(vobj, sx, sy, wx1,
        wy1, wz1, wx2, wy2, wz2)
integer*4 vobj, sx, sy
real wx1, wy1, wz1, wx2, wy2, wz2
```

Pascal
```
procedure mapw(vobj: Object; sx sy: longint;
        var wx1, wy1, wz1, wx2, wy2,
```

DESCRIPTION

mapw takes a pair of 2-D screen coordinates and maps them into 3-D world coordinates. Since the *z* coordinate is missing from the screen coordinate system, the point becomes a line in world space. **mapw** computes the inverse mapping from *vobj*, a viewing object.

A viewing object is a graphical object that contains only viewport, projection, viewing transformation, and modeling routines. A correct mapping from screen coordinates to world coordinates requires the viewing object contain the projection and viewing transformations that mapped the displayed object from world to screen coordinates. The system returns a world space line, which is computed from *(sx, sy)* and *vobj*, as two points and stores them in the locations addressed by *wx1*, *wy1*, *wz1* and *wx2*, *wy2*, *wz2*.

SEE ALSO

mapw2

Programming Guide, Section 9.1, Mapping Screen Coordinates to World Coordinates

NOTE

This routine is available only in immediate mode.

NAME

mapw2 – maps a point on the screen into 2-D world coordinates

SPECIFICATION

C
```
mapw2(vobj, sx, sy, wx, wy)
Object vobj;
Screencord sx, sy;
Coord *wx, *wy;
```

FORTRAN
```
subroutine mapw2(vobj, sx, sy, wx, wy)
integer*4 vobj, sx, sy
real wx, wy
```

Pascal
```
procedure mapw2(vobj: Object; sx sy: longint;
        var wx, wy: Coord);
```

DESCRIPTION

mapw2 is the 2-D version of **mapw**. *vobj* is a viewing object containing the viewport, projection, viewing, and modeling transformations that define world space. *sx* and *sy* define a point in screen coordinates. *wx* and *wy* return the corresponding world coordinates. If the transformation is not 2D, the result is undefined.

SEE ALSO

mapw

Programming Guide, Section 9.1, Mapping Screen Coordinates to World Coordinates

NOTE

This routine is available only in immediate mode.

NAME

maxsize – specifies the maximum size of a graphics window

SPECIFICATION

C
```
maxsize(x, y)
long x, y;
```

FORTRAN
```
subroutine maxsiz(x, y)
integer*4 x, y
```

Pascal
```
procedure maxsize(x, y: longint);
```

DESCRIPTION

maxsize specifies the maximum size of a graphics window under the window manager. Call it at the beginning of a graphics program before **winopen**. **maxsize** takes effect when **winopen** is called.

You can also call **maxsize** in conjunction with **winconstraints** to modify the enforced maximum size after the window has been created. The default maximum size is 1024 pixels wide and 768 pixels high. You can reshape the graphics window, but the window manager does not allow it to become larger than the specified maximum size.

If **maxsize** is called without **winopen**, or if the system is not running the window manager, the routine is ignored.

SEE ALSO

minsize, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

minsize – specifies the minimum size of a graphics window

SPECIFICATION

C
```
minsize(x, y)
long x, y;
```

FORTRAN
```
subroutine minsiz(x, y)
integer*4 x, y
```

Pascal
```
procedure minsize(x, y: longint);
```

DESCRIPTION

minsize specifies the minimum size of a graphics window under the window manager. Call it at the beginning of a graphics program that runs under the window manager. It takes effect when **winopen** is called. You can also call **minsize** in conjuction with **winconstraints** to modify the enforced minimum size after the window has been created. The default minimum size is 40 pixels wide and 30 pixels high. You can reshape the window, but the window manager does not allow it to become smaller than the specified minimum size.

If **minsize** is called without **winopen**, or if the system is not running the window manager, the routine is ignored.

SEE ALSO

maxsize, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

move – moves the current graphics position to a specified point

SPECIFICATION

C

```
move(x, y, z)
Coord x, y, z;

movei(x, y, z)
Icoord x, y, z;

moves(x, y, z)
Scoord x, y, z;

move2(x, y)
Coord x, y;

move2i(x, y)
Icoord x, y;

move2s(x, y)
Scoord x, y;
```

FORTRAN

```
subroutine move(x, y, z)
real x, y, z

subroutine movei(x, y, z)
integer*4 x, y, z

subroutine moves(x, y, z)
integer*2 x, y, z

subroutine move2(x, y)
real x, y

subroutine move2i(x, y)
integer*4 x, y

subroutine move2s(x, y)
integer*2 x, y
```

Pascal **procedure move(x, y, z: Coord);**
procedure movei(x, y, z: Icoord);
procedure moves(x, y, z: Scoord);
procedure move2(x, y: Coord);
procedure move2i(x, y: Icoord);
procedure move2s(x, y: Scoord);

DESCRIPTION

move moves (without drawing) the current graphics position to the point that *x*, *y*, *z* specify. **move** has six forms: 3-D floating point, 3-D integer, 2-D floating point, 2-D integer, 3-D short integer, and 2-D short integer. **move2(x,y)** is equivalent to **move(x,y,0.0)**.

SEE ALSO

draw, pnt, rdr, rmv

Programming Guide, Section 3.4, Lines

NAME

multimap – organizes the color map as 16 small maps

SPECIFICATION

C **multimap()**

FORTRAN **subroutine multim**

Pascal **procedure multimap;**

DESCRIPTION

multimap organizes the color map as 16 small maps, each with a maximum of 256 RGB entries. The number of entries in each map is the total number of available colors, which is determined by the number of bitplanes divided by 16. **multimap** does not take effect until **gconfig** is called.

SEE ALSO

gconfig, getcmmode, getmap, onemap, setmap

Programming Guide, Section 6.2, Color Maps

NOTE

This routine is available only in immediate mode.

Do not use under the window manager.

NAME

multmatrix – premultiplies the current transformation matrix

SPECIFICATION

C
```
multmatrix(m)
Matrix m;
```

FORTRAN
```
subroutine multma(m)
real m(4,4)
```

Pascal
```
procedure multmatrix(var m: Matrix);
```

DESCRIPTION

multmatrix premultiplies the current top of the transformation stack by the given matrix (*m*). If *T* is the current matrix, **multmatrix**(M) replaces *T* with *M*T*.

SEE ALSO

loadmatrix, popmatrix, pushmatrix

Programming Guide, Section 4.5, User-Defined Transformations

NAME

newpup – allocates and initializes a structure for a new menu

SPECIFICATION

C **long newpup()**

FORTRAN **integer*4 function newpup()**

Pascal **function newpup: longint;**

DESCRIPTION

newpup allocates and initializes a structure for a new menu; it returns a positive menu identifier. Use **newpup** with **addtop** to create pop-up menus in FORTRAN programs.

SEE ALSO

addtopup, defpup, dopup, freepup

Using mex, Chapter 3, Making Pop-Up Menus

NOTE

This routine is available only in immediate mode under the window manager. It cannot be used remotely.

NAME

newtag – creates a new tag in an object

SPECIFICATION

C
```
newtag(newtag, oldtag, offset)
Tag newtag, oldtag;
long offset;
```

FORTRAN
```
subroutine newtag(newtag, oldtag, offset)
integer*4 newtag, oldtag, offset
```

Pascal
```
procedure newtag(newtag, oldtag: Tag; offset:
        Offset);
```

DESCRIPTION

newtag creates a new tag *offset* positions beyond *oldtag*.

SEE ALSO

maketag

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

noise – filters valuator motion

SPECIFICATION

C
```
noise(v, delta)
Device v;
short delta;
```

FORTRAN
```
subroutine noise(v, delta)
integer*4 v, delta
```

Pascal
```
procedure noise(v: Device; delta: Short);
```

DESCRIPTION

noise determines how often queued valuators make entries in the event queue. Some valuators are noisy. For example, a device that is not moving can still report small fluctuations in value. **noise** can set a lower limit on what constitutes a move. That is, the value of a noisy valuator (*v*) must change by at least *delta* before the motion is considered significant. For example, **noise(v,5)** means that valuator *v* must move at least 5 units before it makes a new queue entry.

SEE ALSO

setvaluator

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

noport – specifies that a program does not require a graphics window

SPECIFICATION

C **noport()**

FORTRAN **subroutine noport**

Pascal **procedure noport;**

DESCRIPTION

noport specifies that a graphics program does not need screen space, and therefore does not need a graphics window. This is useful for programs that only read or write the color map. Call **noport** at the beginning of a graphics program; then call **getport** or **winopen** to do a graphics initialization.

The system ignores **noport** if **winopen** is not called.

SEE ALSO

getport, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

objdelete – deletes routines from an object

SPECIFICATION

C
```
objdelete(tag1, tag2)
Tag tag1, tag2;
```

FORTRAN
```
subroutine objdel(tag1, tag2)
integer*4 tag1, tag2
```

Pascal
```
procedure objdelete(tag1, tag2: Tag);
```

DESCRIPTION

objdelete is an object editing routine. It removes the routines that are between *tag1* and *tag2* from an object. **objdelete** removes any tags defined between *tag1* and *tag2*, although *tag1* and *tag2* remain.

If no object is open for editing when **objdelete** is called, it is ignored.

objdelete leaves the pointer at the end of the object after it executes.

SEE ALSO

editobj, objinsert, objreplace

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

objinsert – inserts routines in an object at a specified location

SPECIFICATION

C
```
objinsert(t)
Tag t;
```

FORTRAN
```
subroutine objins(t)
integer*4 t
```

Pascal
```
procedure objinset(t: Tag);
```

DESCRIPTION

objinsert takes tag *t* as an argument, and positions an editing pointer on the specified routine. Add the desired graphics routines after the tag.

Use **closeobj** or another positioning routine **(objdelete, objinsert,** or **objreplace)** to terminate the insertion.

SEE ALSO

editobj, closeobj, objdelete, objreplace, maketag

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

objreplace – overwrites existing display list routines with new ones

SPECIFICATION

C
```
objreplace(t)
Tag t;
```

FORTRAN
```
subroutine objrep(t)
integer*4 t
```

Pascal
```
procedure objreplace(t: Tag);
```

DESCRIPTION

objreplace combines the functions of **objinsert** and **objdelete.** It takes a single argument, tag *t*. Graphics routines that follow **objreplace** overwrite existing ones until **closeobj, objinsert, objdelete,** or **objreplace** terminates the replacement.

objreplace requires the new routine be the same length as the one it replaces; this makes replacement operations fast. Use **objdelete** and **objinsert** for more general replacement.

Use **objreplace** as a quick method to create a new version of a routine.

SEE ALSO

closeobj, editobj, objdelete, objinsert

Programming Guide, Section 8.3, Object Editing

NOTE

This routine is available only in immediate mode.

NAME

onemap – organizes the color map as one large map

SPECIFICATION

C **onemap()**

FORTRAN **subroutine onemap**

Pascal **procedure onemap;**

DESCRIPTION

onemap organizes the color map as a single map with a maximum of 4096 RGB entries. The number of entries is 2^p, where p is the number of available bitplanes. You must call **gconfig** for **onemap** to take effect. The system is initially in **onemap** mode.

SEE ALSO

gconfig, getcmmode, multimap, setmap

Programming Guide, Section 6.2, Color Maps

NOTE

This routine is available only in immediate mode.

NAME

ortho, ortho2 – define an orthographic projection transformation

SPECIFICATION

C

```
ortho(left, right, bottom, top, near, far)
        Coord left, right, bottom, top, near,

ortho2(left, right, bottom, top)
        Coord left, right, bottom, top;
```

FORTRAN

```
subroutine ortho(left, right, bottom, top, near, far)
real left, right, bottom, top, near, far

subroutine ortho2(left, right, bottom, top)
real left, right, bottom, top
```

Pascal

```
procedure ortho(left, right, bottom, top, near,
        far: Coord);

procedure ortho2(left, right, bottom, top: Coord);
```

DESCRIPTION

ortho specifies a box-shaped enclosure in the eye coordinate system that is mapped to the viewport. *left, right, bottom, top* are the *x* and *y* clipping planes. *near* and *far* are distances along the line of sight from the eye space origin, and can be negative. The *z* clipping planes are at *–near* and *–far*.

ortho2 defines a 2-D clipping rectangle. When you use **ortho2** with 3-D world coordinates, the z values are not transformed. When **ortho2** is specified, objects with z values outside the range $-1 \le z \le 1$ are clipped out.

Both **ortho** and **ortho2** load a matrix onto the transformation stack, overwriting what was there.

SEE ALSO

perspective, window

Programming Guide, Section 4.3, Projection Transformations

NAME

pagecolor – sets the color of the textport background

SPECIFICATION

C
```
pagecolor(c)
Colorindex c;
```

FORTRAN
```
subroutine pageco(c)
integer*4 c
```

Pascal
```
procedure pagecolor(c: longint);
```

DESCRIPTION

pagecolor sets the textport background color.

SEE ALSO

color, textcolor, textport, tpon, tpoff

Graphics Programming, Chapter 14, Textports

NOTE

This routine is available only in immediate mode.

NAME

pagewritemask – sets the writemask for the textport background

SPECIFICATION

C
```
pagewritemask(pmask)
Colorindex pmask;
```

FORTRAN
```
subroutine pagewr(pmask)
integer*4 pmask
```

Pascal
```
procedure pagewritemask(pmask: longint);
```

DESCRIPTION

pagewritemask sets the textport background writemask. It is undefined under the window manager.

SEE ALSO

color, pagecolor, textcolor, textport, textwritemask, tpoff, tpon, writemask

Programming Guide, Chapter 14, Textports.

NOTE

This routine is available only in immediate mode.

NAME

passthrough – passes a single token through the Geometry Pipeline

SPECIFICATION

C
```
passthrough(token)
short token;
```

FORTRAN
```
subroutine passth(token)
integer*4 token
```

Pascal
```
procedure passthrough(token: longint);
```

DESCRIPTION

passthrough passes a single 16-bit integer through the Geometry Pipeline. Use it in feedback mode to parse the returned information.

For example, you can use **passthrough** between every pair of points that is being transformed and clipped by the Geometry Engines. If a point is clipped out, two **passthrough** tokens appear in a row in the output buffer. It is dangerous to use **passthrough** when not in feedback mode; it can send a random routine to the raster subsystem.

SEE ALSO

Programming Guide, Section 10.2, Feedback Mode

NOTE

This routine is available only in feedback mode.

NAME

patch – draws a surface patch

SPECIFICATION

C
```
patch(geomx, geomy, geomz)
Matrix geomx, geomy, geomz;
```

FORTRAN
```
subroutine patch(geomx, geomy, geomz)
real geomx(4,4), geomy(4,4), geomz(4,4)
```

Pascal
```
procedure patch(var geomx, geomy, geomz: Matrix);
```

DESCRIPTION

patch draws a surface patch using the current **patchbasis, patchprecision,** and **patchcurves. rpatch** draws a rational surface patch. The control points *geomx , geomy , geomz* determine the shape of the patch.

SEE ALSO

defbasis, patchbasis, patchcurves, patchprecision, rpatch

Programming Guide, Section 11.3, Drawing Surfaces

NAME

patchbasis – sets current basis matrices

SPECIFICATION

C
```
patchbasis(uid, vid)
long uid, vid;
```

FORTRAN
```
subroutine patchb(uid, vid)
integer*4 uid, vid
```

Pascal
```
procedure patchbasis(uid, vid: longint);
```

DESCRIPTION

patchbasis sets the current basis matrices (defined by **defbasis**) for the *u* and *v* parametric directions of a surface patch. **patch** uses the current *u* and *v* bases when it executes.

SEE ALSO

defbasis, patch, patchcurves, patchprecision

Programming Guide, Section 11.3, Drawing Surfaces

NAME

patchcurves – sets the number of curves that represent a patch

SPECIFICATION

C
```
patchcurves(ucurves, vcurves)
long ucurves, vcurves;
```

FORTRAN
```
subroutine patchc(ucurves, vcurves)
integer*4 ucurves, vcurves
```

Pascal
```
procedure patchcurves(ucurves, vcurves: longint);
```

DESCRIPTION

patchcurves sets the current number of *u* and *v* curves that represent a patch as a wire frame.

SEE ALSO

patch, patchbasis, patchprecision

Programming Guide, Section 11.3, Drawing Surfaces

NAME

patchprecision – sets the precision at which curves are drawn

SPECIFICATION

C
```
patchprecision(usegments, vsegments)
long usegments, vsegments;
```

FORTRAN
```
subroutine patchp(usegments, vsegments)
integer*4 usegments, vsegments
```

Pascal
```
procedure patchprecision(usegments, vsegments:
        longint);
```

DESCRIPTION

patchprecision sets the precision with which the system draws curves that make up a wireframe patch. The *u* and *v* directions for a patch specify the precisions independently. Patch precisions are similar to curve precisions—they specify the minimum number of line segments used to draw a patch.

SEE ALSO

curveprecision, patch, patchbasis, patchcurves

Programming Guide, Section 11.3, Drawing Surfaces

NAME

pclos – polygon close

SPECIFICATION

C **pclos()**

FORTRAN **subroutine pclos**

Pascal **procedure pclos;**

DESCRIPTION

pclos closes a filled polygon. It terminates a sequence of **pmv** and **pdr**, or **rpmv** and **rpdr**. The polygon so defined is filled using the current pattern, color, and writemask. For example, the following sequence draws a square:

```
pmv(0.0, 0.0, 0.0);
pdr(1.0, 0.0, 0.0);
pdr(1.0, 1.0, 0.0);
pdr(0.0, 1.0, 0.0);
pclos();
```

All polygons must be convex.

Be careful not to confuse **plcos** with the UNIX system call *pclose*, which closes a UNIX pipe.

SEE ALSO

pdr, pmv, rpdr, rpmv

Programming Guide, Section 3.6, Polygons

NAME

pdr – polygon draw

SPECIFICATION

C

```
pdr(x, y, z)
Coord x, y, z;

pdri(x, y, z)
Icoord x, y, z;

pdrs(x, y, z)
Scoord x, y, z;

pdr2(x, y)
Coord x, y;

pdr2i(x, y)
Icoord x, y;

pdr2s(x, y)
Scoord x, y;
```

FORTRAN

```
subroutine pdr(x, y, z)
real x, y, z

subroutine pdri(x, y, z)
integer*4 x, y, z

subroutine pdrs(x, y, z)
integer*2 x, y, z

subroutine pdr2(x, y)
real x, y

subroutine pdr2i(x, y)
integer*4 x, y

subroutine pdr2s(x, y)
integer*2 x, y
```

Pascal **procedure pdr(x, y, z: Coord);**
procedure pdri(x, y, z: Icoord);
procedure pdrs(x, y, z: Scoord);
procedure pdr2(x, y: Coord);
procedure pdr21(x, y: Icoord);
procedure pdr2s(x, y: Scoord);

DESCRIPTION

pdr specifies the next point in a filled polygon. You draw a typical polygon with a **pmv**, a sequence of **pdr** and close it with a **pclos**. For example, the following sequence draws a square:

```
pmv(0.0, 0.0, 0.0);
pdr(1.0, 0.0, 0.0);
pdr(1.0, 1.0, 0.0);
pdr(0.0, 1.0, 0.0);
pclos();
```

All polygons must be convex.

SEE ALSO

pclos, pmv, rpdr, rpmv

Programming Guide, Section 3.6, Polygons

NAME

perspective – defines a perspective projection transformation

SPECIFICATION

C
```
perspective(fovy, aspect, near, far)
Angle fovy;
float aspect;
Coord near, far;
```

FORTRAN
```
subroutine perspe(fovy, aspect, near, far)
integer*4 fovy
real aspect, near, far
```

Pascal
```
procedure perspective(foyv: longint; aspect:
        real; near, far: Coord);
```

DESCRIPTION

perspective defines a projection transformation by indicating the field-of-view angle (*fovy*) in the *y* direction of the eye coordinate system; the *aspect* ratio that determines the field of view in the *x* direction; and the distance to the *near* and *far* clipping planes in the *z* direction. The aspect ratio is a ratio of *x* to *y*. In general, the aspect ratio in **perspective** should match the aspect ratio of the associated viewport. For example, *aspect*=2.0 means the viewer's angle of view is twice as wide in *x* as it is in *y*. If the viewport is twice as wide as it is tall, it displays the image without distortion. *near* and *far* are the distances from the viewer to the near and far clipping planes, and are always positive.

perspective loads a matrix onto the transformation stack, overwriting what was there.

fovy is in tenths of degrees, as are all angles. *fovy* must be ≥ 2 or an error results.

SEE ALSO

ortho, window

Programming Guide, Section 4.3, Projection Transformations

NAME

pick – puts the system in picking mode

SPECIFICATION

C
```
pick(buffer, numnames)
short buffer[];
long numnames;
```

FORTRAN
```
subroutine pick(buffer, numnam)
integer*2 buffer(*)
integer*4 numnam
```

Pascal
```
procedure pick(var buffer: Short; numnames:
        longint);
```

DESCRIPTION

pick facilitates the cursor as a pointing object. When you draw an image in picking mode, nothing is drawn. It places a special viewing matrix on the stack, which discards everything in the image that does not intersect a small region around the lower-left corner of the cursor.

The graphical items that intersect the picking region are hits and store the contents of the name stack in *buffer*. *numnames* specifies the maximum number of names the system saves. Picking does not work if you issue a new viewport while in picking mode.

SEE ALSO

clearhitcode, endpick, endselect, gethitcode, gselect, loadname, picksize, pushname, popname,

Programming Guide, Section 9.2, Picking

NOTE

This routine is available only in immediate mode.

BUGS

When using a debugger, do not stop the graphics between calls to **pick** and **endpick** because the graphics are frozen and the results cannot appear on the screen.

NAME

picksize – sets the dimensions of the picking region

SPECIFICATION

C
```
picksize(deltax, deltay)
short deltax, deltay;
```

FORTRAN
```
subroutine picksi(deltax, deltay)
integer*4 deltax, deltay
```

Pascal
```
procedure picksize(deltax, deltay; longint);
```

DESCRIPTION

picksize has two arguments, *deltax* and *deltay*, which define the dimensions of the picking region in pixels. The picking region is rectangular. It is centered at the current cursor position, the origin of the cursor glyph. In picking mode, any objects that intersect the picking region are reported in the event queue.

SEE ALSO

pick

Programming Guide, Section 9.2, Picking

NOTE

This routine is available only in immediate mode.

NAME

pmv – polygon move

SPECIFICATION

C

```
pmv(x, y, z)
Coord x, y, z;

pmvi(x, y, z)
Icoord x, y, z;

pmvs(x, y, z)
Scoord x, y, z;

pmv2(x, y)
Coord x, y;

pmv2i(x, y)
Icoord x, y;

pmv2s(x, y)
Scoord x, y;
```

FORTRAN

```
subroutine pmv(x, y, z)
real x, y, z

subroutine pmvi(x, y, z)
integer*4 x, y, z

subroutine pmvs(x, y, z)
integer*2 x, y, z

subroutine pmv2(x, y)
real x, y

subroutine pmv2i(x, y)
integer*4 x, y

subroutine pmv2s(x, y)
integer*2 x, y
```

Pascal **procedure pmv(x, y, z: Coord);**
procedure pmvi(x, y, z: Icoord);
procedure pmvs(x, y, z: Scoord);
procedure pmv2(x, y: Coord);
procedure pmv2i(x, y: Icoord);
procedure pmv2s(x, y: Scoord);

DESCRIPTION

pmv moves the starting point of a filled polygon. You draw a typical polygon with a **pmv**, a sequence of **pdr**, and close it with a **pclos**. For example, the following sequence draws a square:

```
(0.0, 0.0, 0.0);
pdr(1.0, 0.0, 0.0);
pdr(1.0, 1.0, 0.0);
pdr(0.0, 1.0, 0.0);
pclos();
```

All polygons must be convex.

SEE ALSO

pclos, pdr, rpdr, rpmv

Programming Guide, Section 3.6, Polygons

NAME

pnt – draws a point

SPECIFICATION

C

```
pnt(x, y, z)
Coord x, y, z;

pnti(x, y, z)
Icoord x, y, z;

pnts(x, y, z)
Scoord x, y, z;

pnt2(x, y)
Coord x, y;

pnt2i(x, y)
Icoord x, y;

pnt2s(x, y)
Scoord x, y;
```

FORTRAN

```
subroutine pnt(x, y, z)
real x, y, z

subroutine pnti(x, y, z)
integer*4 x, y, z

subroutine pnts(x, y, z)
integer*2 x, y, z

subroutine pnt2(x, y)
real x, y

subroutine pnt2i(x, y)
integer*4 x, y

subroutine pnt2s(x, y)
integer*2 x, y
```

Pascal **procedure pnt(x, y, z: Coord);**
procedure pnti(x, y, z: Icoord);
procedure pnts(x, y, z: Scoord);
procedure pnt2(x, y: Coord);
procedure pnt2i(x, y: Icoord);
procedure pnt2s(x, y: Scoord);

DESCRIPTION

pnt colors a point in world coordinates. If the point is visible in the current viewport, it is shown as one pixel. The pixel is drawn in the current color (if in depth-cue mode, the depth-cued color is used) using the current writemask. **pnt** updates the current graphics position after it executes.

SEE ALSO

draw, move

Programming Guide, Section 3.3, Points

NAME

polarview – defines the viewer's position in polar coordinates

SPECIFICATION

C
```
polarview(dist, azim, inc, twist)
Coord dist;
Angle azim, inc, twist;
```

FORTRAN
```
subroutine polarv(dist, azim, inc, twist)
real dist
integer*4 azim, inc, twist
```

Pascal
```
procedure polarview(dist: Coord; azim, inc, twist:
        longint);
```

DESCRIPTION

polarview defines the viewer's position in polar coordinates. *dist*, *azim*, and *inc* define a viewpoint. *dist* is the distance from the viewpoint to the world space origin. *azim* is the angle in the *x*-*y* plane, measured from the axis. *inc* is the angle in *y*-*z* plane, measured from the *z* axis. The line of sight extends from the viewpoint through the world space origin. *twist* rotates the viewpoint around the line of sight using the right-hand rule. All angles are specified in tenths of degrees and are integers.

SEE ALSO

lookat

Programming Guide, Section 4.2, Viewing Transformations

NAME

polf – draws a filled polygon

SPECIFICATION

C

```
polf(n, parray)
long n;
Coord parray[][3];

polfi(n, parray)
long n;
Icoord parray[][3];

polfs(n, parray)
long n;
Scoord parray[][3];

polf2(n, parray)
long n;
Coord parray[][2];

polf2i(n, parray)
long n;
Icoord parray[][2];

polf2s(n, parray)
long n;
Scoord parray[][2];
```

FORTRAN

```
subroutine polf(n, parray)
integer*4 n
real parray(3,n)

subroutine polfi(n, parray)
integer*4 n
integer*4 parray(3,n)

subroutine polfs(n, parray)
integer*4 n
integer*2 parray(3,n)
```

```
subroutine polf2(n, parray)
integer*4 n
real parray(2,n)

subroutine polf2i(n, parray)
integer*4 n
integer*4 parray(2,n)

subroutine polf2s(n, parray)
integer*4 n
integer*2 parray(2,n)
```

Pascal

```
procedure polf(n: longint; var parray: Coord);
procedure polfi(n: longint; var parray: Icoord);
procedure polfs(n: longint; var parray: Scoord);
procedure polf2(n: longint; var parray: Coord);
procedure polf2i(n: longint; var parray: Icoord);
procedure polf2s(n: longint; var parray: Scoord);
```

DESCRIPTION

polf fills polygonal areas using the current pattern, color, and writemask. It takes two arguments: an array of points (*parray*) and the number of points in that array (*n*). Polygons are represented as arrays of points. The first and last points connect automatically to close a polygon. The points can be expressed as integers, shorts, or real numbers in 2-D or 3-D space. 2-D polygons are drawn with z=0. After the polygon is filled, the current graphics position is set to the first point in the array.

SEE ALSO

pdr, pmv, poly, rect, rectf, rpdr, rpmv

Programming Guide, Section 3.6, Polygons

NAME

poly – outlines a polygon

SPECIFICATION

C

```
poly(n, parray)
long n;
Coord parray[][3];

polyi(n, parray)
long n;
Icoord parray[][3];

polys(n, parray)
long n;
Scoord parray[][3];

poly2(n, parray)
long n;
Coord parray[][2];

poly2i(n, parray)
long n;
Icoord parray[][2];

poly2s(n, parray)
long n;
Scoord parray[][2];
```

FORTRAN

```
subroutine poly(n, parray)
integer*4 n
real parray(3,n)

subroutine polyi(n, parray)
integer*4 n
integer*4 parray(3,n)

subroutine polys(n, parray)
integer*4 n
integer*2 parray(3,n)
```

```
subroutine poly2(n, parray)
integer*4 n
real parray(2,n)

subroutine poly2i(n, parray)
integer*4 n
integer*4 parray(2,n)

subroutine poly2s(n, parray)
integer*4 n
integer*2 parray(2,n)
```

Pascal

```
procedure poly(n: longint; var parray: Coord);
procedure polyi(n: longint; var parray; Icoord);
procedure polys(n: longint; var parray: Scoord);
procedure poly2(n: longint; var parray: Coord);
procedure poly2i(n: longint; var parray: Icoord);
procedure poly2s(n: longint; var parray: Scoord);
```

DESCRIPTION

poly outlines a polygon. It takes two arguments: an array of points (*parray*) and the number of points in that array (*n*). A polygon is represented as an array of points. The first and last points connect automatically to close the polygon. The points can be expressed as integers, shorts, or real numbers, in 2-D or 3-D space. 2-D polygons are drawn with z=0. The polygon is outlined using the current linestyle, linewidth, color, and writemask. The maximum number of points in a polygon is 384.

SEE ALSO

pclos, pdr, pmv, polf, rect, rectf, rpdr, rpmv

Programming Guide, Section 3.6, Polygons

NAME

popattributes – pops the attribute stack

SPECIFICATION

C **popattributes()**

FORTRAN **subroutine popatt**

Pascal **procedure popattributes;**

DESCRIPTION

popattributes restores the most recently saved values (those that were pushed by **pushattributes)** of the global state attributes:

Attributes	
backbuffer	pattern
color	raster font
frontbuffer	resetlinestyle
linestyle	RGB color
linestyle backup	RGB writemask
linewidth	writemask pattern
lsrepeat	

If you attempt to pop an empty attribute stack, an error message appears.

SEE ALSO

backbuffer, color, defcursor, frontbuffer, linewidth, lsbackup, lsrepeat, pushattributes, RGBcolor, RGBwritemask, setlinestyle, setpattern, writemask

Programming Guide, Section 2.2, Saving Global State Attributes

NAME

popmatrix – pops the transformation matrix stack

SPECIFICATION

C **popmatrix()**

FORTRAN **subroutine popmat**

Pascal **procedure popmatrix;**

DESCRIPTION

popmatrix pops the transformation matrix stack. When **popmatrix** executes, the matrix on top of the stack is lost.

SEE ALSO

loadmatrix, multmatrix, pushmatrix

Programming Guide, Section 4.5, User-Defined Transformations

NAME

popname – pops a name off the name stack

SPECIFICATION

C **popname()**

FORTRAN **subroutine popnam**

Pascal **procedure popname;**

DESCRIPTION

popname removes the top name from the name stack. It is used in picking and selecting.

popname is ignored outside of picking and selecting mode.

SEE ALSO

gselect, loadname, pushname, pick

Programming Guide, Section 9.2, Picking

NAME

popviewport – restores viewport, screenmask, and **setdepth** parameters

SPECIFICATION

C **popviewport()**

FORTRAN **subroutine popvie**

Pascal **procedure popviewport;**

DESCRIPTION

popviewport pops the stack of viewports and screenmasks. When **popviewport** executes, the viewports on top of the stack are lost. **popviewport** also restores the **setdepth** parameters.

SEE ALSO

getscrmask, pushviewport, setdepth, viewport

Programming Guide, Section 4.4, Viewports

NAME

prefposition – specifies the preferred location and size of a graphics window

SPECIFICATION

C
```
prefposition(x1, x2, y1, y2)
long x1, x2, y1, y2;
```

FORTRAN
```
subroutine prefpo(x1, x2, y1, y2)
integer*4 x1, x2, y1, y2
```

Pascal
```
procedure prefposition(x1, x2, y1, y2: longint);
```

DESCRIPTION

prefposition specifies the preferred location and size of a graphics window. You specify the location in screen coordinates (*x1*, *x2*, *y1*, y2).

Call **prefposition** at the beginning of a graphics program that runs under the window manager. Use **prefposition** in conjuction with **winconstraints** to modify the enforced size and location after the window has been created. **prefposition** is ignored if **winopen** is not called, or if the window manager is not running.

SEE ALSO

winconstraints, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

prefsize – specifies the preferred size of a graphics window

SPECIFICATION

C
```
prefsize(x, y)
long x, y;
```

FORTRAN
```
subroutine prefsi(x, y)
integer*4 x, y
```

Pascal
```
procedure prefsize(x, y: longint);
```

DESCRIPTION

prefsize specifies the preferred size of a graphics window as *x* pixels by *y* pixels. Call **prefsize** at the beginning of a graphics program that runs under the window manager.

Use **prefsize** with **winconstraints** to modify the enforced window size after the window has been created. **prefsize** is ignored if **winopen** is not called or if the window manager is not running.

SEE ALSO

winconstraints, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

pupcolor – specifies the current pop-up drawing color

SPECIFICATION

C
```
pupcolor(c)
long c;
```

FORTRAN
```
subroutine pupcol(c)
integer*4 c
```

Pascal
```
procedure pupcolor(c: longint);
```

DESCRIPTION

pupcolor specifies the current color for drawing into the pop-up bitplanes. The colors specified in the file *.mexrc* range from 1 to 3. In *.mexrc* , the commands bindcolor cursor 255 0 0, bindcolor menu 20 20 20, bindcolor menuback 220 220 220, associate pupcolors 1, 2, and 3 to the specified RGB values.

In *gl.h*, **PUP_CURSOR** is the cursor color, **PUP_BLACK** is the menu color, and **PUP_WHITE** is the menu background color.

SEE ALSO

endfullscrn, fullscrn, endpupmode, pupmode

Using mex , Chapter 3, Making Pop-Up Menus

NOTE

This routine is available only in immediate mode under the window manager.

NAME

pupmode – provides access to the pop-up menu bitplanes

SPECIFICATION

C **pupmode()**

FORTRAN **suboutine pupmod**

Pascal **procedure pupmode;**

DESCRIPTION

pupmode enables the two highest-order bitplanes for writing. The window manager uses these bitplanes to display pop-up menus. An application process can access these bitplanes and retain the input focus without conflicting with other processes. Although processes can write in these bitplanes at any time, it is recommended they write to them when they have the input focus. Carefully consider exceptions to this rule.

SEE ALSO

endfullscrn, endpupmode, fullscrn, pupcolor

Using mex , Chapter 3, Making Pop-Up Menus

NOTE

This routine is available only in immediate mode under the window manager.

NAME

pushattributes – saves the global state attributes

SPECIFICATION

C **pushattributes()**

FORTRAN **subroutine pushat**

Pascal **procedure pushattributes;**

DESCRIPTION

pushattributes saves the global state attributes. The system maintains a stack of attributes, and **pushattributes** puts copies of them on the stack. The global state attributes that are saved include:

Attributes	
backbuffer	pattern
color	raster font
frontbuffer	resetlinestyle
linestyle	RGB color
linestyle backup	RGB writemask
linewidth	writemask pattern
lsrepeat	

The attribute stack is 10 levels deep. **pushattributes** is ignored if the stack is full.

SEE ALSO

backbuffer, color, frontbuffer, linewidth, lsbackup, lsrepeat, popattributes, resetls, RGBcolor, RGBmode, RGBwritemask, setlinestyle, setpattern, writemask

Programming Guide, Section 2.2, Saving Global State Attributes

NAME

pushmatrix – pushes down the transformation matrix stack

SPECIFICATION

C **pushmatrix()**

FORTRAN **subroutine pushma**

Pascal **procedure pushmatrix;**

DESCRIPTION

pushmatrix pushes down the transformation matrix stack, duplicating the current matrix. For example, if the stack contains one matrix, *M*, after a call to **pushmatrix**, the matrix contains two copies of *M*. The top copy can be modified.

The transformation matrix stack is 8 levels in hardware and is 32 levels in software.

SEE ALSO

loadmatrix, multmatrix, popmatrix

Programming Guide, Section 4.5, User-Defined Transformations

NAME

pushname – pushes a new name on the name stack

SPECIFICATION

C
```
pushname(name)
short name;
```

FORTRAN
```
subroutine pushna(name)
integer*4 name
```

Pascal
```
procedure pushname(name: longint);
```

DESCRIPTION

pushname pushes the name stack down one level, and puts a new 16-bit name on top. The system stores the contents of the name stack in a buffer for each hit in picking and selecting modes.

pushname is ignored outside of picking and selecting mode.

SEE ALSO

gselect, loadname, pick, popname

Programming Guide, Section 9.2, Picking

NAME

pushviewport – duplicates the current viewport

SPECIFICATION

C **pushviewport()**

FORTRAN **subroutine pushvi**

Pascal **procedure pushviewport;**

DESCRIPTION

The current viewport is the top element in a stack of viewports. **pushviewport** duplicates the current viewport and pushes it on the stack. After **pushviewport** executes, there are two copies of the current viewport in the stack; you can change the top one without losing the previous one. In addition, it saves the screenmask and the **setdepth** parameters.

SEE ALSO

getscrmask, popviewport, setdepth, viewport

Programming Guide, Section 4.4, Viewports

NAME

qdevice – queues a device (keyboard, button, or valuator)

SPECIFICATION

C
```
qdevice(dev)
Device dev;
```

FORTRAN
```
subroutine qdevic(dev)
integer*4 dev
```

Pascal
```
procedure qdevice(dev: Device);
```

DESCRIPTION

qdevice changes the state of the specified device so that events occurring within the device are entered in the event queue. The device can be the keyboard, a button, a valuator, or certain other pseudo-devices. (See Appendix A for device valuators.)

The maximum number of queue entries is 50.

SEE ALSO

noise, unqdevice, tie

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

qenter – creates an event queue entry

SPECIFICATION

C **qenter(qtype, val)**
short qtype, val;

FORTRAN **subroutine qenter(qtype, val)**
B "integer*4 qtype, val"

Pascal **procedure quenter(qtype, val: longint);**

DESCRIPTION

qenter takes two 16-bit integers, *dev* and *value*, and enters them into the event queue. There is no way to distinguish user-defined and system-defined entries unless disjointed sets of device numbers are used.

See Appendix A for a list of system-defined devices.

SEE ALSO

qread, qreset, qtest

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

qread – reads the first entry in the event queue

SPECIFICATION

C
```
long qread(data)
short *data;
```

FORTRAN
```
integer*4 function qread(data)
integer*2 data
```

Pascal
```
function qread(var data: Short); longint;
```

DESCRIPTION

When there is an entry in the queue, **qread** returns the device number of queue entry, writes the data of the entry into *data*, and removes the entry from the queue.

SEE ALSO

qreset, qtest

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

qreset – empties the event queue

SPECIFICATION

C **qreset()**

FORTRAN **subroutine qreset**

Pascal **procedure qreset;**

DESCRIPTION

qreset removes all entries from the event queue and discards them.

SEE ALSO

qenter, qread, qtest

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

qtest – checks the contents of the event queue

SPECIFICATION

C **long qtest()**

FORTRAN **integer*4 function qtest()**

Pascal **function qtest: longint;**

DESCRIPTION

qtest returns zero if the queue is empty. Otherwise, it returns the device number of the first entry. The queue remains unchanged.

SEE ALSO

qenter, qread, qreset

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

rcrv – draws a curve

SPECIFICATION

C
```
rcrv(geom)
Coord geom[4][4];
```

FORTRAN
```
subroutine rcrv(geom)
real geom(4,4)
```

Pascal
```
procedure rcrv(var geom: Coord);
```

DESCRIPTION

rcrv draws a rational cubic spline curve segment using the current curve basis and precision. *geom* specifies the four control points of the curve segment.

SEE ALSO

crv, crvn, curvebasis, curveprecision, defbasis, rcrvn

Programming Guide, Section 11.2, Drawing Curves

NAME

rcrvn – draws a series of curve segments

SPECIFICATION

C
```
rcrvn(n, geom)
long n;
Coord geom[][4];
```

FORTRAN
```
subroutine rcrvn(n, geom)
integer*4 n
real geom(4,n)
```

Pascal
```
procedure rcrvn(n: longint: var geom: Coord);
```

DESCRIPTION

rcrvn draws a series of rational cubic spline curve segments using the current basis and precision. The control points specified in *geom* determine the shapes of the curve segments and are used four at a time. For example, if *n* is 6, three curve segments are drawn, the first using points 0,1,2,3 as control points, and the second and third segments are controlled by points 1,2,3,4 and 2,3,4,5, respectively. If the current basis is a B-spline, Cardinal spline, or basis with similar properties, the curve segments are joined end to end and appear as a single curve.

SEE ALSO

crv, crvn, curvebasis, curveprecision, crvn, defbasis

Programming Guide, Section 11.2, Drawing Curves

NAME

rdr – relative draw

SPECIFICATION

C

```
rdr(dx, dy, dz)
Coord dx, dy, dz;

rdri(dx, dy, dz)
Icoord dx, dy, dz;

rdrs(dx, dy, dz)
Scoord dx, dy, dz;

rdr2(dx, dy)
Coord dx, dy;

rdr2i(dx, dy)
Icoord dx, dy;

rdr2s(dx, dy)
Scoord dx, dy;
```

FORTRAN

```
subroutine rdr(dx, dy, dz)
real dx, dy, dz

subroutine rdri(dx, dy, dz)
integer*4 dx, dy, dz

subroutine rdrs(dx, dy, dz)
integer*2 dx, dy, dz

subroutine rdr2(dx, dy)
real dx, dy

subroutine rdr2i(dx, dy)
integer*4 dx, dy

subroutine rdr2s(dx, dy)
integer*2 dx, dy
```

Pascal **procedure rdr(dx, dy, dz: Coord);**
procedure rdri(dx, dy, dz: Icoord);
procedure rdrs(dx, dy, dz: Scoord;
procedure rdr2(dx, dy: Coord);
procedure rdr2i(dx, dy: Icoord);
procedure rdr2s(dx, dy: Scoord);

DESCRIPTION

rdr is the relative version of **draw**. It connects the current graphics position and a point, at the specified distance, with a line segment using the current linestyle, linewidth, color (if in depth-cue mode, the depth-cued color is used), and writemask. The system updates the current graphics position to the new point.

Do not place routines that invalidate the current graphics position within sequences of relative moves and draws.

SEE ALSO

draw, move, rmv

Programming Guide, Section 3.4, Lines

NAME

readpixels – returns values of specific pixels

SPECIFICATION

C
```
long readpixels(n, colors)
short n;
Colorindex colors[];
```

FORTRAN
```
integer*4 function readpi(n, colors)
integer*4 n
integer*2 colors(n)
```

Pascal
```
function readpixels(n: Short; var colors:
        Colorindex): longint;
```

DESCRIPTION

readpixels attempts to read up to *n* pixel values from the bitplanes. It reads them into the array *colors* starting from the current character position along a single scan line (constant *y*) in the direction of increasing *x*. **readpixels** returns the number of pixels actually read, which is the number requested if the starting point is at least the same number from the edge of the screen. The values of pixels read outside the screen are undefined. **readpixels** updates the current character position to one pixel to the right of the last one read; the current character position is undefined if the new position is outside the viewport.

In double buffer mode, only the back buffer is read. Use **readRGB** to read pixels in RGB mode.

SEE ALSO

readRGB, writepixels

Programming Guide, Section 3.9, Writing and Reading Pixels

NOTE

This routine is available only in immediate mode.

NAME

readRGB – returns values of specific pixels

SPECIFICATION

C
```
long readRGB(n, red, green, blue)
short n;
RGBvalue red[], green[], blue[];
```

FORTRAN
```
integer*4 function readRG(n, red, green, blue)
integer*4 n
character*(*) red, green, blue
```

Pascal
```
function readRGB(n: longint; var red, green, blue:
RGBvalue): longint;
```

DESCRIPTION

readRGB attempts to read up to *n* pixel values from the bitplanes. It reads them into the *red, green, blue* arrays starting from the current character position along a single scan line (constant *y*) in the direction of increasing *x*. **readRGB** returns the number of pixels actually read, which is the number specified in *n* if the starting point is at least the same number from the edge of the screen. The values of pixels read outside the screen are undefined.

readRGB updates the current character position to one pixel to the right of the last one read; the current character position is undefined if the new position is outside the viewport.

readRGB is available only in RGB mode.

SEE ALSO

readpixels, writeRGB

Programming Guide, Section 3.9, Writing and Reading Pixels

NOTE

This routine is available only in immediate mode.

NAME

rect – outlines a rectangular region

SPECIFICATION

C

```
rect(x1, y1, x2, y2)
Coord x1, y1, x2, y2;

recti(x1, y1, x2, y2)
Icoord x1, y1, x2, y2;

rects(x1,y1, x2, y2)
Scoord x1,y1, x2,y2;
```

FORTRAN

```
subroutine rect(x1, y1, x2, y2)
real x1, y1, x2, y2

subroutine recti(x1, y1, x2, y2)
integer*4 x1, y1, x2, y2

subroutine rects(x1, y1, x2,y2)
integer*2 x1, y1, x2, y2
```

Pascal

```
procedure rect(x1, y1, x2, y2: Coord);

procedure recti(x1, y1, x2, y2: Icoord);

procedure rects(x1, y1, x2, y2: Scoord);
```

DESCRIPTION

rect draws a rectangle using the current linestyle, linewidth, color, and writemask. The sides of the rectangle are parallel to the *x* and *y* axes. Since a rectangle is a 2-D shape, **rect** takes only 2-D arguments, and sets the *z* coordinate to zero. The points *(x1, y1)* and *(x2, y2)* are the opposite corners of the rectangle.

SEE ALSO

poly, rectf, rpdr, rpmv

Programming Guide, Section 3.5, Rectangles

NAME

rectcopy – copies a rectangle of pixels on the screen

SPECIFICATION

C
```
rectcopy(x1, y1, x2, y2, newx, newy)
Screencoord x1, y1, x2, y2, newx, newy;
```

FORTRAN
```
subroutine rectco(x1, y1, x2, y2, newx, newy)
integer*4 x1, y1, x2, y2, newx, newy
```

Pascal
```
procedure rectcopy(x1, y1, x2, y2, newx, newy:
        Screencord);
```

DESCRIPTION

rectcopy copies a rectangular array of pixels (*x1, y1, x2, y2*) to another position on the screen. The point (*newx, newy*) defines the lower-left corner of the new window position. The current viewport and screenmask mask the drawing of the copied region.

SEE ALSO

Programming Guide, Section 3.5, Rectangles

NOTE

This routine is available only in immediate mode.

NAME

rectf – fills a rectangular area

SPECIFICATION

C

```
rectf(x1, y1, x2, y2)
Coord x1, y1, x2, y2;

rectfi(x1, y1, x2, y2)
Icoord x1, y1, x2, y2;

rectfs(x1, y1, x2, y2)
Scoord x1, y1, x2, y2;
```

FORTRAN

```
subroutine rectf(x1, y1, x2, y2)
real x1, y1, x2, y2

subroutine rectfi(x1, y1, x2, y2)
integer*4 x1, y1, x2, y2

subroutine rectfs(x1, y1, x2, y2)
integer*2 x1, y1, x2, y2
```

Pascal

```
procedure rectf(x1, y1, x2, y2: Coord);

procedure rectfi(x1, y1, x2, y2: Icoord);

procedure rectfs(x1, y1, x2, y2: Scoord);
```

DESCRIPTION

rectf produces a filled rectangular region, using the current pattern, color, and writemask. The sides of the rectangle are parallel to the *x* and *y* axes of the object coordinate system. Since a rectangle is a 2-D shape, **rectf** takes only 2-D arguments and sets the *z* coordinate to zero. The points *(x1, y1)* and *(x2, y2)* are the opposite corners of the rectangle. The current graphics position is set to *(x1, y1)* after the region is drawn.

In backface mode, you must specify the lower-left and upper-right corners.

SEE ALSO

polf, rect, rdr, rmv

Programming Guide, Section 3.5, Rectangles

NAME

resetls – controls the continuity of linestyles

SPECIFICATION

C
```
resetls(b)
Boolean b;
```

FORTRAN
```
subroutine resetl(b)
logical b
```

Pascal
```
procedure resetls(b: Boolean);
```

DESCRIPTION

resetls affects the reinitialization of the linestyle pattern between segments. It takes one boolean argument. TRUE(1), the default, indicates stippling of each line starts at the beginning of the linestyle pattern. FALSE(0) turns off the mode: the linestyle is not reset between segments, and the stippling of one segment continues from where it left off at the end of the previous segment. Calls to **resetls** initialize the linestyle, no matter what the argument, and invalidate the current graphics position.

Use **resetls** to approximate circles, arcs, and curves with many short lines. If the linestyle is not reset between segments, the pattern of the curve appears smooth and continuous. Do not set the linewidth to 2 unless **resetls** is TRUE(1).

SEE ALSO

deflinestyle, getresetls, lsbackup, setlinestyle

Programming Guide, Section 5.1, Linestyles

NAME

reshapeviewport – sets the viewport to the dimensions of the current graphics window

SPECIFICATION

C **reshapeviewport()**

FORTRAN **subroutine reshap**

Pascal **procedure reshapeviewport;**

DESCRIPTION

reshapeviewport sets the viewport to the dimensions of the current graphics window. **reshapeviewport** is equivalent to:

```
long  xsize, ysize;

getsize(&xsize, &ysize);
viewport(0, xsize, 0, ysize);
```

Use **reshapeviewport** when REDRAW events are received. It is most useful in programs that are independent of the size and shape of the viewport.

SEE ALSO

getorigin, getsize, viewport,

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

RGBcolor – sets the current color in RGB mode

SPECIFICATION

C
```
RGBcolor(red, green, blue)
short red, green, blue;
```

FORTRAN
```
subroutine RGBcol(red, green, blue)
integer*4 red, green, blue
```

Pascal
```
procedure RGBcolor(red, green, blue: longint);
```

DESCRIPTION

RGBcolor sets the current color in RGB mode. *red*, *green*, *blue* are each 8-bit values. The system writes these numbers directly into the bitplanes whenever it draws a pixel. These values control the intensity of red, green, and blue displayed on the screen.

RGBcolor is available only in RGB mode.

SEE ALSO

color, gRGBcolor, RGBwritemask

Programming Guide, Section 6.3, Colors and Writemasks

NAME

RGBcursor – sets the characteristics of the cursor in RGB mode

SPECIFICATION

C
```
RGBcursor(index, red, green, blue, redm, greenm,
        bluem)
short index, red, green, blue, redm, greenm, bluem;
```

FORTRAN
```
subroutine RGBcur(index, red, green, blue, redm,
        greenm, bluem)
integer*4 index, red, green, blue, redm, greenm,
        bluem
```

Pascal
```
procedure RGBcursor(index, red, green, blue, redm,
        greenm, bluem: longint);
```

DESCRIPTION

RGBcursor selects a cursor glyph from a table of 16x16 bit patterns that you define. *index* picks a glyph from the definition table. *red, green, blue* specify the cursor color in RGB mode; *redm, greenm, bluem* define an RGB writemask for the cursor.

RGBcursor is available only in RGB mode.

SEE ALSO

defcursor, RGBmode, RGBwritemask, setcursor

Programming Guide, Section 6.4, Cursors

NOTE

This routine is available only in immediate mode.

NAME

RGBmode – sets a display mode that bypasses the color map

SPECIFICATION

C **RGBmode()**

FORTRAN **subroutine RGBmod**

Pascal **procedure RGBmode;**

DESCRIPTION

There are three display modes: single buffer, double buffer, and RGB. In RGB mode, the IRIS simultaneously writes and displays all bitplanes. The system writes 8-bit values of red, green, and blue into the bitplanes; these values directly control the intensity of the color displayed on the monitor. RGB mode is most useful when the system has 24 bitplanes; otherwise, in systems with fewer bitplanes, the first 16 are shared equally between red and green, and the last 8 define the blue component. A 12-bitplane system provides one-fourth the maximum intensity in red and green, and no blue at all.

You must call **gconfig** for **RGBmode** to execute.

SEE ALSO

doublebuffer, gconfig, getdisplaymode, singlebuffer

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

Do not use this routine while running the window manager.

NAME

RGBwritemask – grants write access to a subset of available bitplanes

SPECIFICATION

C
```
RGBwritemask(red, green, blue)
short red, green, blue;
```

FORTRAN
```
subroutine RGBwri(red, green, blue)
integer*4 red, green, blue
```

Pascal
```
procedure RGBwritemask(red, green, blue: longint);
```

DESCRIPTION

RGBwritemask shields bitplanes that are not used for ordinary drawing routines in RGB mode. *red, green, blue* are masks for each of the three sets of bitplanes. Wherever the digit 1 is in the writemask, the system writes the corresponding bits in the RGB color into the bitplanes. Zeros in the writemask mark bitplanes as read-only. These bitplanes are not changed, regardless of the bits in the RGB color.

SEE ALSO

gRGBmask, RGBcolor, writemask

Programming Guide, Section 6.3, Colors and Writemasks

NAME

ringbell – rings the keyboard bell

SPECIFICATION

C **ringbell()**

FORTRAN **subroutine ringbe**

Pascal **procedure ringbell;**

DESCRIPTION

ringbell rings the keyboard bell.

SEE ALSO

clkoff, clkon, lampoff, lampon, setbell

Programming Guide, Section 7.5, Controlling Peripheral Input/Output Devices

NOTE

This routine is available only in immediate mode.

NAME

rmv – relative move

SPECIFICATION

C

```
rmv(dx, dy, dz)
Coord dx, dy, dz;

rmvi(dx, dy, dz)
Icoord dx, dy, dz;

rmvs(dx, dy, dz)
Scoord dx, dy, dz;

rmv2(dx, dy)
Coord dx, dy;

rmv2i(dx, dy)
Icoord dx, dy;

rmv2s(dx, dy)
Scoord dx, dy;
```

FORTRAN

```
subroutine rmv(dx, dy, dz)
real dx, dy, dz

subroutine rmvi(dx, dy, dz)
integer*4 dx, dy, dz

subroutine rmvs(dx, dy, dz)
integer*2 dx, dy, dz

subroutine rmv2(dx, dy)
real dx, dy

subroutine rmv2i(dx, dy)
integer*4 dx, dy

subroutine rmv2s(dx, dy)
integer*2 dx, dy
```

Pascal **procedure rmv(dx, dy, dz: Coord);**
procedure rmvi(dx, dy, dz: Icoord);
procedure rmvs(dx, dy, dz: Scoord);
procedure rmv2(dx, dy: Coord);
procedure rmv2i(dx, dy: Icoord);
procedure rmv2s(dx, dy: Scoord);

DESCRIPTION

rmv is the relative version of **move**. It moves (without drawing) the graphics position the specified amount relative to its current value. The routine has six forms: 3-D floating point, 3-D integer, 2-D floating point, 2-D integer, 3-D short integer, and 2-D short integer. **rmv2(x,y)** is equivalent to **rmv(x, y, 0.0)**.

SEE ALSO

draw, move, rdr

Programming Guide, Section 3.4, Lines

NAME

rot – rotates graphical primitives (floating point version)

SPECIFICATION

C
```
rot(a, axis)
float a;
char axis;
```

FORTRAN
```
subroutine rot(a, axis)
real a
character axis
```

Pascal
```
procedure rot(a: real; axis: longint);
```

DESCRIPTION

rot specifies an angle (*a*) and an axis of rotation (*axis*). The floating point angle is given in degrees according to the right-hand rule. The axis of rotation is defined by a character, either 'x', 'y', or 'z' (the character can be upper- or lowercase).

rot is a modeling routine; it changes the current transformation matrix. All objects drawn after **rot** executes are rotated. Use **pushmatrix** and **popmatrix** to preserve and restore an unrotated world space.

SEE ALSO

popmatrix, pushmatrix, rotate, scale, translate

Programming Guide, Section 4.1 Modeling Transformations

NAME

rotate – rotates graphical primitives

SPECIFICATION

C
```
rotate(a, axis)
Angle a;
char axis;
```

FORTRAN
```
subroutine rotate(a, axis)
integer*4 a
character axis
```

Pascal
```
procedure rotate(a: longint; axis: longint);
```

DESCRIPTION

rotate specifies an angle (*a*) and an axis of rotation (*axis*). The angle (*a*) is given in tenths of degrees according to the right-hand rule. The axis of rotation (*axis*) is defined by a character, either 'x', 'y', or 'z'. (The character can be upper- or lowercase.)

rotate is a modeling routine; it changes the current transformation matrix. All objects drawn after **rotate** executes are rotated. Use **pushmatrix** and **popmatrix** to preserve and restore an unrotated world space.

SEE ALSO

popmatrix, pushmatrix, rot, scale, translate

Programming Guide, Section 4.1, Modeling Transformations

NAME

rpatch – draws a rational surface patch

SPECIFICATION

C
```
rpatch(geomx, geomy, geomz, geomw)
Matrix geomx, geomy, geomz, geomw;
```

FORTRAN
```
subroutine rpatch(geomx, geomy, geomz, geomw)
real geomx(4,4), geomy(4,4), geomz(4,4), geomw(4,4)
```

Pascal
```
procedure rpatch(var geomx, geomy, geomz,
        geomw: Matrix);
```

DESCRIPTION

rpatch draws a rational surface patch using the current **patchbasis**, **patchprecision**, and **patchcurves**. The control points *geomx* , *geomy* , *geomz* determine the shape of the patch. *geomw* specifies the rational component of the patch to **rpatch**.

SEE ALSO

defbasis, patch, patchbasis, patchcurves, patchprecision,

Programming Guide, Section 11.3, Drawing Surfaces

NAME

rpdr – relative polygon draw

SPECIFICATION

C

```
rpdr(dx, dy, dz)
Coord dx, dy, dz;

rpdri(dx, dy, dz)
Icoord dx, dy, dz;

rpdrs(dx, dy, dz)
Scoord dx, dy, dz;

rpdr2(dx, dy)
Coord dx, dy;

rpdr2i(dx, dy)
Icoord dx, dy;

rpdr2s(dx, dy)
Scoord dx, dy;
```

FORTRAN

```
subroutine rpdr(dx, dy, dz)
real dx, dy, dz

subroutine rpdri(dx, dy, dz)
integer*4 dx, dy, dz

subroutine rpdrs(dx, dy, dz)
integer*2 dx, dy, dz

subroutine rpdr2(dx, dy)
real dx, dy

subroutine rpdr2i(dx, dy)
integer*4 dx, dy

subroutine rpdr2s(dx, dy)
integer*2 dx, dy
```

Pascal **procedure rpdr(dx, dy, dz: Coord);**
procedure rpdri(dx, dy, dz: Icoord);
procedure rpdrs(dx, dy, dz: Scoord);
procedure rpdr2(dx, dy: Coord);
procedure rpdr2i(dx, dy: Icoord);
procedure rpdr2s(dx, dy: Scoord);

DESCRIPTION

rpdr is the relative version of **pdr**. It specifies the next point in a filled polygon, using the previous point (the current graphics position) as the origin. **rpdr** updates the current graphics position.

All polygons must be convex.

SEE ALSO

pclos, pdr, pmv, rpmv

Programming Guide, Section 3.6, Polygons

NAME

rpmv – relative polygon move

SPECIFICATION

C

```
rpmv(dx, dy, dz)
Coord dx, dy, dz;

rpmvi(dx, dy, dz)
Icoord dx, dy, dz;

rpmvs(dx, dy, dz)
Scoord dx, dy, dz;

rpmv2(dx, dy)
Coord dx, dy;

rpmv2i(dx, dy)
Icoord dx, dy;

rpmv2s(dx, dy)
Scoord dx, dy;
```

FORTRAN

```
subroutine rpmv(dx, dy, dz)
real dx, dy, dz

subroutine rpmvi(dx, dy, dz)
integer*4 dx, dy, dz

subroutine rpmvs(dx, dy, dz)
integer*2 dx, dy, dz

subroutine rpmv2(dx, dy)
real dx, dy

subroutine rpmv2i(dx, dy)
integer*4 dx, dy

subroutine rpmv2s(dx, dy)
integer*2 dx, dy
```

Pascal **procedure rpmv(dx, dy, dz: Coord)**
procedure rpmvi(dx, dy, dz: Icoord);
procedure rpmvs(dx, dy, dz: Scoord);
procedure rpmv2(dx, dy: Coord);
procedure rpmv2i(dx, dy: Icoord);
procedure rpmv2s(dx, dy: Scoord);

DESCRIPTION

rpmv is the relative version of **pmv**. It specifies a relative move to the starting point of a filled polygon, using the current graphics position as the origin. **rpmv** updates the current graphics position to the new point.

All polygons must be complex.

SEE ALSO

pclos, pdr, pmv, rpdr

Programming Guide, Section 3.6, Polygons

NAME

scale – scales and mirrors objects

SPECIFICATION

C
```
scale(x, y, z)
float x, y, z;
```

FORTRAN
```
subroutine scale(x, y, z)
real x, y, z
```

Pascal
```
procedure scale(x, y, z: real);
```

DESCRIPTION

scale shrinks, expands, and mirrors objects. *x*, *y*, *z* specify scaling in each of the three coordinate directions. Values with a magnitude greater than 1 expand the object; values with a magnitude less than 1 shrink it. Negative values mirror the object.

scale is a modeling routine; it changes the current transformation matrix. All objects drawn after **scale** executes are affected.

Use **pushmatrix** and **popmatrix** to limit the scope of **scale**.

SEE ALSO

popmatrix, pushmatrix, rot, rotate, translate

Programming Guide, Section 4.1, Modeling Transformations

NAME

screenspace – interprets graphics positions as absolute screen coordinates

SPECIFICATION

C **screenspace()**

FORTRAN **subroutine screen**

Pascal **procedure screenspace;**

DESCRIPTION

screenspace makes a program interpret graphics positions as absolute screen coordinates. This allows pixels and locations outside a program's graphics window to be read. **screenspace** is equivalent to:

```
int xmin, ymin;

getorigin(&xmin, &ymin);
viewport(-xmin, XMAXSCREEN-xmin, -ymin, YMAXSCREEN-ymin);
ortho2(-0.5, 1023.5, -0.5, 767.5);
```

SEE ALSO

getorigin, ortho2, viewport

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

scrmask – defines a clipping mask for fine character clipping

SPECIFICATION

C
```
scrmask(left, right, bottom, top)
Screencoord left, right, bottom, top;
```

FORTRAN
```
subroutine scrmas(left, right, bottom, top)
integer*4 left, right, bottom, top
```

Pascal
```
procedure scrmask(left, right, bottom,
        top: Screencoord);
```

DESCRIPTION

scrmask provides fine character clipping. **viewport** sets the same area for both the viewport and the screenmask, which *left, right, bottom, top* define. **scrmask** sets only the screenmask, which must be placed entirely within the viewport.

Strings that begin outside the viewport are clipped out; this is called gross clipping. Strings that begin inside the viewport but outside the screenmask are clipped to the pixel boundaries of the screenmask; this is called fine clipping. All drawing routines are also clipped to the viewport, but **scrmask** is only useful for characters; gross clipping is sufficient for all other primitives.

SEE ALSO

getscrmask, viewport

Programming Guide, Section 4.4, Viewports

NAME

setbell – sets the duration of the keyboard bell

SPECIFICATION

C
```
setbell(mode)
char mode;
```

FORTRAN
```
subroutine setbel(mode)
integer*4 mode
```

Pascal
```
procedure setbell(mode: Byte);
```

DESCRIPTION

setbell sets the duration of the beep of the keyboard bell.

Mode	Meaning
0	off
1	short beep
2	long beep

SEE ALSO

clkoff, clkon, lampoff, lampon, ringbell

Programming Guide, Section 7.5, Controlling Peripheral Input/Output Devices

NOTE

This routine is available only in immediate mode.

NAME

setcursor – sets the cursor characteristics

SPECIFICATION

C
```
setcursor(index, color, wtm)
short index;
Colorindex color, wtm;
```

FORTRAN
```
subroutine setcur(index, color, wtm)
integer*4 index, color, wtm
```

Pascal
```
procedure setcursor(index: longint; color, wtm:
        longint);
```

DESCRIPTION

setcursor selects a cursor glyph from among those defined with **defcursor.** *index* picks a glyph from the definition table. *color* and *wtm* set a color and writemask for the cursor. The default cursor is zero; it is displayed with the color 1 drawn in the first available bitplane, and is automatically updated on each vertical retrace.

Under the window manager, *color* and *wtm* are ignored.

SEE ALSO

attachcursor, curorigin, curstype, defcursor, getcursor, mapcolor, RGBcursor

Programming Guide, Section 6.4, Cursors

NOTE

This routine is available only in immediate mode.

NAME

setdblights – sets the lights on the dial and button box

SPECIFICATION

C
```
setdblights(mask)
long mask;
```

FORTRAN
```
subroutine setdbl(mask)
integer*4 mask
```

Pascal
```
procedure setdblights(mask: longint);
```

DESCRIPTION

setdblights turns on a combination of the lights on the dial and button box. Each bit in the mask corresponds to a light. For example, to turn on lights 4, 7, and 22 (and leave all the others off), set the mask to (1<<4) | (1<<7) | (1<<22) = 0x400090.

SEE ALSO

dbtext

Programming Guide, Section 7.5, Controlling Peripheral Input/Output Devices

NOTE

This routine is available only in immediate mode.

NAME

setdepth – sets up a 3-D viewport

SPECIFICATION

C
```
setdepth(near, far)
Screencoord near, far;
```

FORTRAN
```
subroutine setdep(near, far)
integer*4 near, far
```

Pascal
```
procedure setdepth(near, far: Screencoord);
```

DESCRIPTION

viewport specifies a mapping from the left, right, bottom, and top clipping planes in world coordinate values to screen coordinate values. **setdepth** completes this mapping for homogeneous world coordinates. The two arguments map the near and far clipping planes to the desired screen coordinate values. The default is **setdepth(0, 1023)**. The legal values range from 32767 (0x7FFF) to -32768 (0x8000). When used for depth-cueing or z-buffering, the range should be restricted to (0x3FFF) to (0xC000).

The depth is the distance from your eye. **setdepth** is used in z-buffering, depth-cueing, and certain feedback applications.

SEE ALSO

depthcue, endfeedback, feedback, zbuffer

Programming Guide, Section 12.1, Z-Buffer Mode

NAME

setfastcom – sends data in 8 bits per byte.

SPECIFICATION

C **Boolean setfastcom()**

FORTRAN **logical function setfas**

Pascal **function setfastcom: Boolean;**

DESCRIPTION

setslowcom sends data in 8 bits per byte, as required by certain connections, e.g., some ethernet protocalls. You use **setfastcom** on an IRIS terminal. It has no effect on a workstation.

setfastcom is listed here for compatibility with the remote Graphics Library.

SEE ALSO

gflush, setslowcom

Programming Guide, Section 2.1, Initialization

NOTES

setfastcom has no effect if it is run locally.

NAME

setlinestyle – selects a linestyle pattern

SPECIFICATION

C
```
setlinestyle(index)
short index;
```

FORTRAN
```
subroutine setlin(index)
integer*4 index
```

Pascal
```
procedure setlinestyle(index: longint);
```

DESCRIPTION

setlinestyle selects a linestyle pattern. *index* is an index into the linestyle table built by **deflinestyle**. There is always a current linestyle; it draws lines and curves, and outlines rectangles, polygons, circles, and arcs. The default linestyle is 0, which is a solid line. It cannot be redefined.

SEE ALSO

deflinestyle, getlstyle, linewidth, lsbackup, resetls

Programming Guide, Section 5.1, Linestyles

NAME

setmap – selects one of the 16 small color maps

SPECIFICATION

C
```
setmap(mapnum)
short mapnum;
```

FORTRAN
```
subroutine setmap(mapnum)
integer*4 mapnum
```

Pascal
```
procedure setmap(mapnum: Short);
```

DESCRIPTION

setmap selects one of the 16 small maps, numbered 0 through 15 in multimap mode. **setmap** is ignored in onemap mode.

SEE ALSO

getmap, multimap, onemap

Programming Guide, Section 6.2, Color Maps

NOTE

This routine is available only in immediate mode.

NAME

setmonitor – sets the monitor type

SPECIFICATION

C
```
setmonitor(type)
short type;
```

FORTRAN
```
subroutine setmon(type)
integer*4 type
```

Pascal
```
procedure setmonitor(type: Short);
```

DESCRIPTION

setmonitor sets the monitor to 30Hz interlaced, 50Hz noninterlaced, 60Hz noninterlaced, NTSC, or PAL depending on whether *type* is HZ30, HZ60, NTSC, or PAL, respectively. Those constants are defined in the file *get.h*.

TYPE	Monitor Type
HZ30	30Hz interlaced
HZ50	50Hz noninterlaced
HZ60	60Hz noninterlaced
NTSC	NTSC
PAL	PAL

SEE ALSO

getmonitor, getothermonitor

Programming Guide, Section 2.1, Initialization

NOTE

This routine is available only in immediate mode.

NAME

setpattern – selects a pattern for filling polygons, rectangles, and curves

SPECIFICATION

C
```
setpattern(index)
short index;
```

FORTRAN
```
subroutine setpat(index)
integer*4 index
```

Pascal
```
procedure setpattern(index: longint);
```

DESCRIPTION

setpattern selects a pattern from among those that **defpattern** defines. The default pattern is pattern 0, which is solid. If you specify an undefined pattern, the default pattern is selected.

SEE ALSO

color, defpattern, getpattern, writemask

Programming Guide, Section 5.2, Patterns

NAME

setshade – sets the current polygon shade

SPECIFICATION

C
```
setshade(shade)
Colorindex shade;
```

FORTRAN
```
subroutine setsha(shade)
integer*4 shade
```

Pascal
```
procedure setshade(shade: longint);
```

DESCRIPTION

setshade sets the current shade value. *shade* is a color index that is associated with the specified vertices that immediately follow **setshade.**

setshade values shade polygons closed with **spclos.** If you use **setshade** in the definition of a polygon that is closed with **pclos**, the results are undefined. Shading works only when the solid pattern is set.

SEE ALSO

spclos, splf

Programming Guide, Section 13.1, Shading

NAME

setslowcom – sends data in 6 bits per byte.

SPECIFICATION

C **Boolean setslowcom()**

FORTRAN **logical function setslo**

Pascal **function setslowcom: Boolean;**

DESCRIPTION

setslowcom sends data in 6 bits per byte, as certain connections require, e.g., RS232 connection. You use **setslowcom** on an IRIS terminal. It has no effect on a workstation.

setslowcom is listed here for compatibility with the remote Graphics Library.

SEE ALSO

gflush, setfastcom

Programming Guide, Section 2.1, Initialization

NOTES

setslowcom has not effect if it is run locally.

NAME

setvaluator – assigns an initial value to a valuator

SPECIFICATION

C
```
setvaluator(v, init, min, max)
Device v;
short init, min, max;
```

FORTRAN
```
subroutine setval(v, init, min, max)
integer*4 v, init, min, max
```

Pascal
```
procedure setvaluator(v: Device; init, min, max:
        Short);
```

DESCRIPTION

setvaluator assigns an initial value *init* to a valuator. *min* and *max* are the lower and upper bounds for the values the device can assume.

Some devices, such as tablets, report values fixed to a grid. In this case, the device defines an initial position and *init* is ignored.

SEE ALSO

getvaluator

Programming Guide, Section 7.2, Initializing a Device

NOTE

This routine is available only in immediate mode.

NAME

shaderange – sets range of color indices used in depth-cueing

SPECIFICATION

C
```
shaderange(lowindex, highindex, z1, z2)
Colorindex lowindex, highindex;
Screencoord z1, z2;
```

FORTRAN
```
subroutine shader(lowindex, highindex, z1, z2)
integer*2 lowindex, highindex, z1, z2
```

Pascal
```
procedure shaderange(lowindex, highindex: longint;
        z1, z2: longint);
```

DESCRIPTION

shaderange sets the range of color indices used in drawing depth-cued lines and points. The range [z1, z2] is mapped linearly into the color index range. z values less than *z1* map to *highindex*; z values greater than *z2* map to *lowindex*.

SEE ALSO

depthcue

Programming Guide, Section 13.1, Shading

NAME

singlebuffer – writes and displays all bitplanes

SPECIFICATION

C **singlebuffer()**

FORTRAN **subroutine single**

Pascal **procedure singlebuffer;**

DESCRIPTION

singlebuffer invokes single buffer mode. In single buffer mode, the system simultaneously updates and displays the image data in the active bitplanes. Consequently, incomplete or changing pictures can appear on the screen. **singlebuffer** does not take effect until **gconfig** is called.

SEE ALSO

doublebuffer, gconfig, getdisplaymode, gsync, RGBmode

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

NAME

spclos – draws the current open, shaded polygon

SPECIFICATION

C **spclos()**

FORTRAN **subroutine spclos**

Pascal **procedure spclos;**

DESCRIPTION

spclos closes and shades the polygon. It is like **pclos**. When **spclos** ends a polygon, the system draws it using the intensities that **setshade** specifies. If **setshade** is used in the definition of a polygon, it must be closed with **spclos**. If it is closed with **pclos**, the results are undefined.

SEE ALSO

pclos, pdr, pmv, setshade, splf

Programming Guide, Section 13.1, Shading

NAME

splf – draws a shaded filled polygon

SPECIFICATION

C

```
splf(n, parray, iarray)
long n;
Coord parray[][3];
Colorindex iarray[];

splfi(n, parray, iarray)
long n;
Icoord parray[][3];
Colorindex iarray[];

splfs(n, parray, iarray)
long n;
Scoord parray[][3];
Colorindex iarray[];

splf2(n, parray, iarray)
long n;
Coord parray[][2];
Colorindex iarray[];

splf2i(n, parray, iarray)
long n;
Icoord parray[][2];
Colorindex iarray[];

splf2s(n, parray, iarray)
long n;
Scoord parray[][2];
Colorindex iarray[];
```

FORTRAN

```
subroutine splf(n, parray, iarray)
integer*4 n
real parray(3,n)
integer*2 iarray(n)
```

```
subroutine splfi(n, parray, iarray)
integer*4 n
integer*4 parray(3,n)
integer*2 iarray(n)

subroutine splfs(n, parray, iarray)
integer*4 n
integer*2 parray(3,n)
integer*2 iarray(n)

subroutine splf2(n, parray, iarray)
integer*4 n
real parray(2,n)
integer*2 iarray(n)

subroutine splf2i(n, parray, iarray)
integer*4 n
integer*4 parray(2,n)
integer*2 iarray(n)

subroutine splf2s(n, parray, iarray)
integer*4 n
integer*2 parray(2,n)
integer*2 iarray(n)
```

Pascal

```
procedure splf(n: longint; var parray; Coord;
        var iarray: Colorindex);

procedure splfi(n: longint; var parray; Icoord;
        var iarray: Colorindex);

procedure splfs(n: longint; var parray: Scoord;
        var iarray: Colorindex);

procedure splf2(n: longint; var parray: Coord;
        var iarray: Colorindex);

procedure splf2i(n: longint; var parray: Icoord;
        var iarray: Colorindex;

procedure splf2s(n: longint; var parray: Scoord;
        var iarray: Colorindex);
```

splf

DESCRIPTION

splf draws Gouraud-shaded polygons using the current pattern and writemask. It takes three arguments: *parray*, an array of points; *array*, an array of the intensities at these points; *n*, the number of points in each array. Polygons are represented as arrays of points. The first and last points automatically connect to close a polygon. The points can be expressed as integers, shorts, or real numbers, in 2-D or 3-D space. 2-D polygons are drawn with $z = 0$. After the polygon is drawn, the current graphics position is set to the first point in the array.

All polygons must be convex.

SEE ALSO

pdr, pmv, poly, rect, rectf, rpdr, rpmv

Programming Guide, Section 13.1, Shading

NAME

stepunit – specifies that a graphics window change size in discrete steps

SPECIFICATION

C
```
stepunit(xunit, yunit)
long xunit, yunit;
```

FORTRAN
```
subroutine stepun(xunit, yunit)
integer*4 xunit, yunit
```

Pascal
```
procedure stepunit(xunit, yunit: longint);
```

DESCRIPTION

stepunit specifies that the size of a graphics window change in discrete steps of *xunit* in the *x* direction and *yunit* in the *y* direction. Call **stepunit** at the beginning of a graphics program; it takes effect when **winopen** is called. **stepunit** resizes graphics windows in units of a standard size (in pixels). When **stepunit** is called, the dimensions of the graphics window are:

width = xunit*n
height = yunit*m

If **winopen** is not called, or if the system is not running the window manager, **stepunit** is ignored.

SEE ALSO

fudge, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

strwidth – returns the width of the specified text string

SPECIFICATION

C
```
long strwidth(str)
String str;
```

FORTRAN
```
integer*4 function strwid(str, length)
character*(*) str
integer*4 length
```

Pascal
```
function strwidth(str: pstring128): longint;
```

DESCRIPTION

strwidth returns the width of a text string in pixels, using the character spacing parameters of the current raster font. **strwidth** is useful when you do a simple mapping from screen space to world space.

Undefined characters have zero width.

In FORTRAN, **strwid** has two arguments: *str* is the name of the string; *length* is the number of characters in that string.

SEE ALSO

getdescender, getheight, getlwidth, mapw, mapw2

Programming Guide, Section 5.3, Fonts

NOTE

This routine is available only in immediate mode.

NAME

swapbuffers – exchanges the front and back buffers

SPECIFICATION

C **swapbuffers()**

FORTRAN **subroutine swapbu**

Pascal **procedure swapbuffers;**

DESCRIPTION

swapbuffers exchanges the front and back buffers in double buffer mode during the next vertical retrace. Once an image is fully drawn in the back buffer, **swapbuffers** displays it. **swapbuffers** is ignored in single buffer mode or RGB mode.

SEE ALSO

backbuffer, doublebuffer, frontbuffer, gsync, swapinterval

Programming Guide, Section 6.1, Display Modes

NAME

swapinterval – defines a minimum time between buffer swaps

SPECIFICATION

C
```
swapinterval(i)
short i;
```

FORTRAN
```
subroutine swapin(i)
integer*4 i
```

Pascal
```
procedure swapinterval(i: Short);
```

DESCRIPTION

swapinterval defines a minimum time between buffer swaps. For example, a swap interval of 5 refreshes the screen at least five times between execution of successive **swapbuffers**. **swapinterval** changes frames at a steady rate if a new image can be created within one swap interval. The default interval is 1. **swapinterval** is valid only in double buffer mode. It is ignored in single buffer mode or RGB mode.

SEE ALSO

doublebuffer, gsync, swapbuffers

Programming Guide, Section 6.1, Display Modes

NOTE

This routine is available only in immediate mode.

NAME

textcolor – sets the color of text drawn in the textport

SPECIFICATION

C
```
textcolor(tcolor)
Colorindex tcolor;
```

FORTRAN
```
subroutine textco(tcolor)
integer*4 tcolor
```

Pascal
```
procedure textcolor(tcolor: longint);
```

DESCRIPTION

textcolor sets the color of the text drawn in the textport. *tcolor* determines the color of the text **charstr** draws. *tcolor* is the index of the desired color.

SEE ALSO

color, pagecolor, pagewritemask, textport, textwritemask, tpoff, tpon

Programming Guide, Chapter 14, Textports

NOTE

This routine is available only in immediate mode.

NAME

textinit – initializes the console textport

SPECIFICATION

C **textinit()**

FORTRAN **subroutine textin**

Pascal **procedure textinit;**

DESCRIPTION

textinit initializes the console textport to default size, location, textcolor, and pagecolor.

SEE ALSO

pagecolor, textcolor, textport, textwritemask

Programming Guide, Chapter 14, Textports

NOTE

This routine is available only in immediate mode.

NAME

textport – allocates an area of the screen for the textport

SPECIFICATION

C
```
textport(left, right, bottom, top)
Screencoord left, right, bottom, top;
```

FORTRAN
```
subroutine textpo(left, right, bottom, top)
integer*4 left, right, bottom, top
```

Pascal
```
procedure textport(lesft, right, bottom, top: longint);
```

DESCRIPTION

textport allocates an area on the screen for the textport. *left, right, bottom, top* specify the textport in screen coordinates.

textport does not work in RGB mode.

SEE ALSO

gettp, pagecolor, pagewritemask, textcolor, textwritemask, tpoff, tpon, viewport

Programming Guide, Chapter 14, Textports

NOTE

This routine is available only in immediate mode.

NAME

textwritemask – grants write permission for text drawn in the textport

SPECIFICATION

C
```
textwritemask(tmask)
Colorindex tmask;
```

FORTRAN
```
subroutine textwr(tmask)
integer*4 tmask
```

Pascal
```
procedure textwritemask(tmask: longint);
```

DESCRIPTION

textwritemask grants write permission for text drawn in the textport. It does not affect text drawn using **charstr.**

textwritemask is undefined under the window manager.

SEE ALSO

pagecolor, pagewritemask, textcolor, textport, tpoff, tpon, writemask

Programming Guide, Chapter 14, Textports

NOTE

This routine is available only in immediate mode.

NAME

tie – ties two valuators to a button

SPECIFICATION

C
```
tie(b, v1, v2)
Device b, v1, v2;
```

FORTRAN
```
subroutine tie(b, v1, v2)
integer*4 b, v1, v2
```

Pascal
```
procedure tie(b, v1, v2: Device);
```

DESCRIPTION

tie requires a button *b* and two valuators *v1* and *v2*. When a queued button changes state, three entries are made in the queue: one records the current state of the button and two record the current positions of each valuator. You can tie one valuator to a button by making *v2* = 0. You can untie a button by making both *v1* and *v2*= 0. *v1* appears before *v2* in the event queue; *b* precedes both *v1* and *v2*.

SEE ALSO

getbutton

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

tpoff – turns off the textport

SPECIFICATION

C **tpoff()**

FORTRAN **subroutine tpoff**

Pascal **procedure tpoff;**

DESCRIPTION

tpoff turns off the textport. When the textport is off, characters are not written to the textport and the textport does not appear on the screen. The texport automatically turns on when you exit a program.

SEE ALSO

textport, tpon

Programming Guide, Chapter 14, Textports

NOTE

This routine is available only in immediate mode.

NAME

tpon – turns on the textport

SPECIFICATION

C **tpon()**

FORTRAN **subroutine tpon**

Pascal **procedure tpon;**

DESCRIPTION

tpon turns on the textport.

SEE ALSO

tpoff, textport

Programming Guide, Chapter 14, Textports

NOTE

This routine is available only in immediate mode.

NAME

translate – translates graphical primitives

SPECIFICATION

C
```
translate(x, y, z)
Coord x, y, z;
```

FORTRAN
```
subroutine transl(x, y, z)
real x, y, z
```

Pascal
```
procedure translate(x, y, z: Coord);
```

DESCRIPTION

translate moves the object space origin to a point specified in the current object coordinate system. **translate** is a modeling routine which changes the current transformation matrix. All objects drawn after **translate** executes are translated. Use **pushmatrix** and **popmatrix** to limit the scope of the translation.

SEE ALSO

popmatrix, pushmatrix, rot, rotate, scale

Programming Guide, Section 4.1, Modeling Transformations

NAME

unqdevice – disables the specified device from making entries in the event queue

SPECIFICATION

C
```
unqdevice(dev)
Device dev;
```

FORTRAN
```
subroutine unqdev(dev)
integer*4 dev
```

Pascal
```
procedure unqdevice(dev: Device);
```

DESCRIPTION

unqdevice removes the specified device from the list of devices whose changes are recorded in the event queue. If a device has recorded events that have not been read, they remain in the queue.

Use **qreset** to flush the event queue.

SEE ALSO

qdevice, qreset

Programming Guide, Section 7.4, The Event Queue

NOTE

This routine is available only in immediate mode.

NAME

viewport – allocates an area of the window for an image

SPECIFICATION

C
```
viewport(left, right, bottom, top)
Screencoord left, right, bottom, top;
```

FORTRAN
```
subroutine viewpo(left, right, bottom, top)
integer*4 left, right, bottom, top
```

Pascal
```
procedure viewport(left, right bottom,
        top: Screencoord);
```

DESCRIPTION

viewport specifies, in pixels, the area of the window that displays an image. Specifying the viewport is the first step in mapping world coordinates to screen coordinates. The portion of world space that **window**, **ortho**, or **perspective** describes is mapped into the viewport. *left*, *right*, *bottom*, *top* coordinates define a rectangular area on the screen.

viewport also loads the screenmask.

SEE ALSO

scrmask, getviewport, popviewport, pushviewport

Programming Guide, Section 4.4, Viewports

NAME

winat – returns the indentifier of the window beneath the cursor

SPECIFICATION

C **long winat();**

FORTRAN **integer*4 function winat()**

Pascal **function winat(): longint;**

DESCRIPTION

winat returns the graphics window identifier (gid) of the window beneath the current position of the cursor.

SEE ALSO

winget

Using mex, Chapter 2, Programming with mex

NAME

winattach – attaches the input focus to the current graphics window and call process

SPECIFICATION

C	**long winattach()**
FORTRAN	**integer*4 function winatt()**
Pascal	**function winattach: longint;**

DESCRIPTION

winattach attaches the input focus to the current graphics window and calling process. **winattach** returns the identifier of the window that lost the input focus. In a multiple window environment, input focus refers to the process that receives events from various input devices, such as the keyboard, mouse, dials, and buttons.

SEE ALSO

qread, qtest, qdevice, winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

winclose – closes the identified graphics window

SPECIFICATION

C
```
winclose(gid)
long gid;
```

FORTRAN
```
subroutine winclo(gid)
integer*4 gid
```

Pascal
```
procedure winclose(gid: longint);
```

DESCRIPTION

winclose closes the graphics window associated with identifier *gid* .

SEE ALSO

winopen

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode under the window manager.

NAME

winconstraints – changes the constraints of the current graphics window

SPECIFICATION

C **winconstraints()**

FORTRAN **subroutine wincon**

Pascal **procedure winconstraints();**

DESCRIPTION

winconstraints binds the currently specified constraints to the current graphics window. These constraints are **minsize, maxsize, keepaspect, prefsize** and **prefposition.** These constraints are reset to none after they are bound to the graphics window.

SEE ALSO

getport, minsize, maxsize, keepaspect, prefsize, prefposition,

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

window – defines a perspective projection transformation

SPECIFICATION

C
```
window(left, right, bottom, top, near, far)
Coord left, right, bottom, top, near, far;
```

FORTRAN
```
subroutine window(left, right, bottom, top, near, far)
real left, right, bottom, top, near, far
```

Pascal
```
procedure window(left, right, bottom, top, near,
        far: Coord);
```

DESCRIPTION

window specifies the position and size of a rectangular viewing frustum in terms of the boundaries of the rectangular region (*left, right, bottom, top*); the distance closest to the eye in the near clipping plane (*near*); and the distance to the far clipping plane (*far*). The system projects the image with perspective onto the screen area that **viewport** defines.

window loads a matrix onto the transformation stack; it overwrites whatever was on the stack.

SEE ALSO

ortho, perspective, viewport

Programming Guide, Section 4.3, Projection Transformations

NAME

winget – returns the identifier of the current graphics window

SPECIFICATION

C **long winget()**

FORTRAN **integer*4 function winget()**

Pascal **function winget: longint;**

DESCRIPTION

winget returns the identifier of the current graphics window.

SEE ALSO

winset

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

winmove – moves the lower-left corner of the current graphics window

SPECIFICATION

C
```
winmove(orgx, orgy)
long orgx, orgy;
```

FORTRAN
```
subroutine winmov
integer*4 orgx, orgy
```

Pascal
```
procedure winmove(orgx, orgy: longint);
```

DESCRIPTION

winmove moves the origin of the current graphics window to the screen coordinates *orgx, orgy*. The origin of the current graphics window is the lower-left corner.

winmove does not change the size and shape of the window.

SEE ALSO

winposition

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode under the window manager.

NAME

winopen – creates a graphics window

SPECIFICATION

C
```
long winopen(name)
char name[];
```

FORTRAN
```
integer*4 function winope(name,length)
character*(*) name
integer*4 length
```

Pascal
```
function winopen(name: pstring128): longint;
```

DESCRIPTION

winopen creates a graphics window. Before calling **winopen**, the characteristics of the graphics window must be specified by **minsize, maxsize, keepaspect, prefsize, prefposition, stepunit, fudge**, and **foreground**. This will initialize graphics the first time it is called in a program.

winopen returns a small integer value (*gid*) identifying the graphics window, or –1 if no additional graphics windows are available. The new window inherits the state of the current graphics window, and replaces it as the current window. If no window characteristics are specified or if the description is incomplete, the window manager prompts you for the missing information. Use the cursor to show the size and location of the graphics window.

name specifies the window *.deskconfig* uses to refer to the window.

In FORTRAN, there is an extra argument, *length*, which is the number of characters in the name string.

winopen queues the pseudo devices INPUTCHANGE and REDRAW.

SEE ALSO

foreground, fudge, keepaspect, maxsize, minsize, noport, prefposition, prefsize, stepunit, winclose

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode under the window manager.

NAME

winpop – moves the current graphics window in front of all other windows

SPECIFICATION

C **winpop()**

FORTRAN **subroutine winpop**

Pascal **procedure winpop;**

DESCRIPTION

Multiple windows appear as a stack of windows that can obscure each other. **winpop** takes the current graphics window from the stack of windows and places it on top.

SEE ALSO

winpush

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

winposition – changes the size and position of the current graphics window

SPECIFICATION

C
```
winposition(x1, x2, y1, y2)
long x1, x2, y1, y2;
```

FORTRAN
```
subroutine winpos(x1, x2, y1, y2)
integer*4 x1, x2, y1, y2;
```

Pascal
```
procedure winposition(x1, x2, y1, y2: Short);
```

DESCRIPTION

winposition moves and reshapes the current graphics window to match the screen coordinates *x1, x2, y1, y2*. This differs from **prefposition** because the reshaped window is not fixed in size and shape, and can be reshaped interactively under the window manager.

SEE ALSO

prefposition, prefsize, winmove

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode under the window manager.

NAME

winpush – places the current graphics window behind all other windows

SPECIFICATION

C **winpush()**

FORTRAN **subroutine winpus**

Pascal **procedure winpush;**

DESCRIPTION

winpush places the current graphics window behind all other windows. Multiple windows appear as a stack of windows that can obscure one another. **winpush** takes the current graphics window from the stack and places it at the bottom.

SEE ALSO

winpop

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

winset – sets the current graphics window

SPECIFICATION

C
```
winset(gid)
long gid;
```

FORTRAN
```
subroutine winset(gid)
integer*4 gid
```

Pascal
```
function winset(gid: longint): longint;
```

DESCRIPTION

winset makes the graphics window associated with identifier *gid* the current window. The system directs all graphics output routines to the current graphics window.

SEE ALSO

winget

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode.

NAME

wintitle – adds a title bar to the current graphics window

SPECIFICATION

C
```
wintitle(name)
char name[];
```

FORTRAN
```
subroutine wintit(name, length)
character*(*) name
integer*4 length
```

Pascal
```
procedure wintitle(name: pstring128);
```

DESCRIPTION

wintitle adds a title bar to the current graphics window. The title bar is drawn using color *menu* and *menuback*, the default that the configuration file *.mexrc* assigns. You can override these defaults by defining a modified *.mexrc* in your home directory.

wintitle("") removes the title bar.

In FORTRAN, there is an extra argument, *length*, which is the number of characters in the name string.

SEE ALSO

pupcolor

Using mex, Chapter 2, Programming with mex

NOTE

This routine is available only in immediate mode under the window manager.

NAME

writemask – grants write permission to available bitplanes

SPECIFICATION

C
```
writemask(wtm)
Colorindex wtm;
```

FORTRAN
```
subroutine writem(wtm)
integer*4 wtm
```

Pascal
```
procedure writemask(wtm: Colorindex);
```

DESCRIPTION

writemask protects bitplanes that are reserved for special uses from ordinary drawing routines. *wtm* is a mask with 1 bit per available bitplane. Wherever there are ones in the writemask, the corresponding bits in the color index are written into the bitplanes. Zeros in the writemask mark bitplanes as read-only. These bitplanes will not be changed, regardless of the bits in the color.

Use **RGBwritemask** in RGB mode.

SEE ALSO

color, RGBwritemask

Programming Guide, Section 6.3, Colors and Writemasks

NAME

writepixels – paints a row of pixels on the screen

SPECIFICATION

C
```
writepixels(n, colors)
short n;
Colorindex colors[];
```

FORTRAN
```
subroutine writep(n, colors)
integer*4 n
integer*2 colors(n)
```

Pascal
```
procedure writepixels(n: Short; var colors:
        Colorindex);
```

DESCRIPTION

writepixels paints a row of pixels on the screen; *n* specifies the number of pixels to paint and *color* is an array of color indices. The starting location is the current character position. The system updates the current character position to the pixel that follows the last painted pixel. The current character position becomes undefined if the updated pixel position is greater than XMAXSCREEN.

The system paints pixels from left to right, and clips to the current screenmask.

writepixels does not automatically wrap from one line to the next. It can be used in single buffer and double buffer modes.

Use **writeRGB** in RGB mode.

SEE ALSO

color, readpixels, scrmask, writeRGB

Programming Guide, Section 3.9, Writing and Reading Pixels

NOTE

This routine is available only in immediate mode.

NAME

writeRGB – paints a row of pixels on the screen

SPECIFICATION

C
```
writeRGB(n, red, green, blue)
short n;
RGBvalue red[], green[], blue[];
```

FORTRAN
```
subroutine writeR(n, red, green, blue)
integer*4 n
character*(*) red, green, blue
```

Pascal
```
procedure writeRGB(n: Short; var red, green,
        blue: RGBvalue);
```

DESCRIPTION

writeRGB paints a row of pixels on the screen in RGB mode. *n* specifies the number of pixels to paint; *red, green, blue* specify arrays of colors for each pixel. The starting location is the current character position. The system updates the current character position to the pixel that follows the last painted pixel. The current character position becomes undefined if the updated pixel position is greater than XMAXSCREEN.

Pixels are painted from left to right, and are clipped to the current screenmask.

writeRGB does not automatically wrap from one line to the next. It supplies a 24-bit RGB value (8 bits for each color) for each pixel. This value is written directly into the bitplanes.

SEE ALSO

readRGB, RGBcolor, RGBwritemask, scrmask, writepixels

Programming Guide, Section 3.9, Writing and Reading Pixels

NOTE

This routine is available only in immediate mode and RGB mode.

NAME

xfpt – transforms points

SPECIFICATION

C

```
xfpt(x, y, z)
Coord x, y, z;

xfpti(x, y, z)
Icoord x, y, z;

xfpts(x, y, z)
Scoord x, y, z;

xfpt2(x, y)
Coord x, y;

xfpt2i(x, y)
Icoord x, y;

xfpt2s(x, y)
Scoord x, y;

xfpt4(x, y, z, w)
Coord x, y, z, w;

xfpt4i(x, y, z, w)
Icoord x, y, z, w;

xfpt4s(x, y, z, w)
Scoord x, y, z, w;
```

FORTRAN

```
subroutine xfpt(x, y, z)
real x, y, z

subroutine xfpti(x, y, z)
integer*4 x, y, z

subroutine xfpts(x, y, z)
integer*2 x, y, z

subroutine xfpt2(x, y)
real x, y
```

```
subroutine xfpt2i(x, y)
integer*4 x, y

subroutine xfpt2s(x, y)
integer*2 x, y

subroutine xfpt4(x, y, z, w)
real x, y, z, w

subroutine xfpt4i(x, y, z, w)
integer*4 x, y, z, w

subroutine xfpt4s(x, y, z, w)
integer*2 x, y, z, w
```

Pascal

```
procedure xfpt(x, y, z: Coord);
procedure xfpti(x, y, z: Icoord);
procedure xfpts(x, y, z: Scoord);
procedure xfpt2(x, y: Coord);
procedure xfpt2i(x, y: Icoord);
procedure xfpt2s(x, y: Scoord);
procedure xfpt4(x, y, z, w: Coord);
procedure xfpt4i(x, y, z, w: Icoord);
procedure xfpt4s(x, y, z, w: Scoord);
```

DESCRIPTION

xfpt multiplies the specified point (*x*, *y*, *z*) by the top matrix on the matrix stack and turns off the clippers and scalers in the Geometry Pipeline. In feedback mode, the feedback buffer saves the 4-D result of the multiplication. In nonfeedback mode, the routine is ignored.

SEE ALSO

Programming Guide, Section 10.2, Feedback Mode

NOTE

This routine is available only in immediate mode.

NAME

zbuffer – starts or ends z-buffer mode

SPECIFICATION

C
```
zbuffer(bool)
Boolean bool;
```

FORTRAN
```
subroutine zbuffe(bool)
logical bool
```

Pascal
```
procedure zbuffer(bool: Boolean);
```

DESCRIPTION

zbuffer starts *bool* = TRUE or ends *bool* = FALSE z-buffer mode. In z-buffer mode, each pixel has an associated z value. To draw a pixel, the system compares the new z value with the z value already associated with it. If the new z value is less than or equal to the existing value (i.e., closer to the viewer), then a new color and z value are stored in the bitplanes; otherwise, the color and z value for the pixel are left unchanged.

All obscured surfaces in objects drawn in this mode are not displayed. z values range from 0xC000 to 0x3FFF on a 32-bitplane system, and 0x0 to 0xFFF on a 28-bitplane system. **setdepth** sets this range.

z-buffering does not work for lines greater than 1.

z-buffering is not effective on systems with less than 28 bitplanes. Because memory bandwidth is reduced on 60Hz monitors, you can improve performance by using **blankscreen** to blank the screen during z-buffer drawing.

SEE ALSO

blankscreen, getzbuffer, setdepth, zclear

Programming Guide, Section 12.1, Z-Buffer Mode

BUGS

A bug exists in the line and point code. Lines and points are not drawn if the z values are exactly the same as the existing value.

NAME

zclear – initializes the z-buffer

SPECIFICATION

C **zclear()**

FORTRAN **subroutine zclear**

Pascal **procedure zclear;**

DESCRIPTION

zclear loads the z-buffer with the largest possible positive integer. This is done most before a z-buffer picture is drawn to clear the buffer so that primitives drawn in z-buffer mode are only affected by each other.

SEE ALSO

zbuffer

Programming Guide, Section 12.1, Z-Buffer Mode

APPENDICES

A: Type Definitions for C and FORTRAN A-1
A.1 C Definitions A-1
A.2 FORTRAN Definitions A-19

B: Geometry Engine Computations B-1

C: Transformation Matrices C-1
C.1 Translation C-1
C.2 Scaling and Mirroring C-1
C.3 Rotation C-2
C.4 Viewing Transformations C-2
C.5 Perspective Transformations C-3
C.6 Orthographic Transformations C-4

D: Feedback Parser D-1

E: Window Manager Programs E-1
E.1 Color Tools E-1
E.2 Image Tools E-4
E.3 General Desktop Tools E-5
E.4 Utilities E-6
E.5 Device Usage Examples E-7
E.6 Fantasy Demonstrations E-8

F: IRIS Programming Tutorial Manual Pages F-1

G: Fast Immediate Mode and User-Defined Display Lists G-1
G.1 Introduction and Overview G-1
G.2 User-Defined Display Lists G-7
G.3 Example -- Defining Your Own Display List G-10
G.4 Counting Instructions G-15
G.5 Conclusions G-16

H: Using the Image Library H-1
H.1 Image Files H-1
H.2 Opening and Closing an Image File H-1
H.2.1 iopen - Open an Image File H-2
H.2.2 iclose - Close an Image File H-3
H.3 Reading from and Writing to Image Files H-3
H.3.1 putrow - Write a Row of Pixels from Buffer to Image File H-3
H.3.2 getrow - Read a Row of Pixels from Image File to Buffer H-4
H.4 Miscellaneous Functions H-4
H.4.1 isetname - Name an Image File H-4
H.4.2 isetcolormap - Interpret Pixel Values H-5
H.4.3 scrsave - Save Rectangular Region of Screen to Image File H-5
H.5 An Example H-6

Appendix A: Type Definitions for C and FORTRAN

A.1 C Definitions

This is a listing of *gl.h*, which should be included in each IRIS program. It contains the type definitions, useful constants, and external definitions for all commands.

```
/* graphics library header file */

/* maximum X and Y screen coordinates */

#define XMAXSCREEN        1023
#define YMAXSCREEN        767

/* various hardware/software limits*/

#define ATTRIBSTACKDEPTH 10
#define VPSTACKDEPTH              8
#define MATRIXSTACKDEPTH 32
#define NAMESTACKDEPTH            1025
#define STARTTAG                  -2
#define ENDTAG                    -3
#define MAXFONTNC                 256 /* max size of font offset array */
#define MINFONTNC                 128 /* for compatibility with old fonts */

/* names for colors in color map loaded by ginit() */

#define BLACK                     0
#define RED                       1
#define GREEN                     2
#define YELLOW                    3
```

```
#define BLUE                    4
#define MAGENTA                 5
#define CYAN                    6
#define WHITE                   7

/* popup colors */

#define PUP_CLEAR       0
#define PUP_CURSOR      1
#define PUP_BLACK       2
#define PUP_WHITE       3

#ifndef FALSE
#define FALSE           0
#endif
#ifndef TRUE
#define TRUE                    (!FALSE)
#endif

/* typedefs */

typedef unsigned char Byte;
typedef long Boolean;
typedef char *String;

typedef short Angle;
typedef short Screencoord;
typedef short Scoord;
typedef long Icoord;
typedef float Coord;
typedef float Matrix[4][4];

typedef unsigned short Colorindex;
typedef unsigned char RGBvalue;

typedef unsigned short Device;

#define PATTERN_16 16
#define PATTERN_32 32
#define PATTERN_64 64

#define PATTERN_16_SIZE  16
#define PATTERN_32_SIZE  64
#define PATTERN_64_SIZE  256

typedef unsigned short Pattern16[PATTERN_16_SIZE];
typedef unsigned short Pattern32[PATTERN_32_SIZE];
typedef unsigned short Pattern64[PATTERN_64_SIZE];

typedef unsigned short Linestyle;
typedef unsigned short Cursor[16];

typedef struct {
```

```
                        unsigned short offset;
                        Byte w,h;
                        char xoff,yoff;
                        short width;
} Fontchar;

typedef long Object;
typedef long Tag;
typedef long Offset;

extern void             addtopup();
extern void             arc();
extern void             arcf();
extern void             arcfi();
extern void             arci();
extern void             arcfs();
extern void             arcs();
extern void             attachcursor();
extern void             backbuffer();
extern void             backface();
extern void             bbox2();
extern void             bbox2i();
extern void             bbox2s();
extern void             blankscreen();
extern void             blanktime();
extern void             blink();
extern long             blkqread();
extern void             callfunc();
extern void             callobj();
extern void             capture();
extern void             charstr();
extern void             circ();
extern void             circf();
extern void             circfi();
extern void             circi();
extern void             circfs();
extern void             circs();
extern void             clear();
extern void             clearhitcode();
extern void             clkoff();
extern void             clkon();
extern void             closeobj();
extern void             cmov();
extern void             cmov2();
extern void             cmov2i();
extern void             cmovi();
extern void             cmov2s();
extern void             cmovs();
extern void             color();
extern void             compactify();
extern void             crv();
extern void             crvn();
extern void             curorigin();
```

```
extern void             cursoff();
extern void             curson();
extern void             curvebasis();
extern void             curveit();
extern void             curveprecision();
extern void             cyclemap();
extern void             dbtext();
extern void             defbasis();
extern void             defcursor();
extern void             deflinestyle();
extern void             defpattern();
extern long             defpup();
extern void             defrasterfont();
extern void             delobj();
extern void             deltag();
extern void             depthcue();
extern void             devport();
extern long             dopup();
extern long             dopupbut();
extern void             doublebuffer();
extern void             draw();
extern void             draw2();
extern void             draw2i();
extern void             draw2s();
extern void             drawi();
extern void             draws();
extern void             editobj();
extern long             endfeedback();
extern void             endfullscrn();
extern long             endpick();
extern void             endpupmode();
extern long             endselect();
extern void             feedback();
extern void             finish();
extern void             font();
extern void             foreground();
extern void             freepup();
extern void             frontbuffer();
extern void             fudge();
extern void             fullscrn();
extern void             gbegin();
extern void             gconfig();
extern Object           genobj();
extern Tag              gentag();
extern long             getbuffer();
extern Boolean          getbutton();
extern Boolean          getcmmode();
extern long             getcolor();
extern void             getcpos();
extern void             getcursor();
extern Boolean          getdcm();
extern void             getdepth();
extern void             getdev();
```

```
extern long             getdisplaymode();
extern long             getfont();
extern void             getgpos();
extern long             getheight();
extern long             gethitcode();
extern Boolean          getlsbackup();
extern long             getlsrepeat();
extern long             getlstyle();
extern long             getlwidth();
extern long             getmap();
extern void             getmatrix();
extern void             getmcolor();
extern long             getmem();
extern long             getmonitor();
extern void             getorigin();
extern Object           getopenobj();
extern long             getothermonitor();
extern long             getpattern();
extern long             getplanes();
extern void             getport();
extern Boolean          getresetls();
extern void             getscrmask();
extern long             getshade();
extern void             getsize();
extern void             gettp();
extern long             getvaluator();
extern void             getviewport();
extern long             getwritemask();
extern Boolean          getzbuffer();
extern void             gewrite();
extern void             gexit();
extern void             gflush();
extern void             ginit();
extern void             gselect();
extern void             greset();
extern void             gsync();
extern void             gRGBcolor();
extern void             gRGBcursor();
extern void             gRGBmask();
extern void             imakebackground();
extern void             initnames();
extern Boolean          ismex();
extern Boolean          isobj();
extern Boolean          isqueued();
extern Boolean          istag();
extern void             keepaspect();
extern void             lampoff();
extern void             lampon();
extern void             linewidth();
extern void             loadmatrix();
extern void             loadname();
extern void             lookat();
extern void             lsbackup();
```

```
extern void             lsrepeat();
extern void             makeobj();
extern void             maketag();
extern void             mapcolor();
extern void             mapw();
extern void             mapw2();
extern void             maxsize();
extern void             minsize();
extern void             move();
extern void             move2();
extern void             move2i();
extern void             move2s();
extern void             movei();
extern void             moves();
extern void             multimap();
extern void             multmatrix();
extern long             newpup();
extern void             newtag();
extern void             noise();
extern void             noport();
extern void             objdelete();
extern void             objinsert();
extern void             objreplace();
extern void             onemap();
extern void             ortho();
extern void             ortho2();
extern void             pagecolor();
extern void             pagewritemask();
extern void             passthrough();
extern void             patch();
extern void             patchbasis();
extern void             patchcurves();
extern void             patchprecision();
extern void             pclos();
extern void             pdr();
extern void             pdr2();
extern void             pdr2i();
extern void             pdr2s();
extern void             pdri();
extern void             pdrs();
extern void             perspective();
extern void             pick();
extern void             picksize();
extern void             pmv();
extern void             pmv2();
extern void             pmv2i();
extern void             pmv2s();
extern void             pmvi();
extern void             pmvs();
extern void             pnt();
extern void             pnt2();
extern void             pnt2i();
extern void             pnt2s();
```

```
extern void             pnti();
extern void             pnts();
extern void             polarview();
extern void             polf();
extern void             polf2();
extern void             polf2i();
extern void             polf2s();
extern void             polfi();
extern void             polfs();
extern void             poly();
extern void             poly2();
extern void             poly2i();
extern void             poly2s();
extern void             polyi();
extern void             polys();
extern void             popattributes();
extern void             popmatrix();
extern void             popname();
extern void             popviewport();
extern void             prefposition();
extern void             prefsize();
extern void             pupcolor();
extern void             pupmode();
extern void             pushattributes();
extern void             pushmatrix();
extern void             pushname();
extern void             pushviewport();
extern void             qdevice();
extern void             qenter();
extern long             qread();
extern void             qreset();
extern long             qtest();
extern void             rcapture();
extern void             rcrv();
extern void             rcrvn();
extern void             rdr();
extern void             rdr2();
extern void             rdr2i();
extern void             rdr2s();
extern void             rdri();
extern void             rdrs();
extern long             readpixels();
extern long             readRGB();
extern void             rect();
extern void             rectcopy();
extern void             rectf();
extern void             rectfi();
extern void             rectfs();
extern void             recti();
extern void             rects();
extern void             resetls();
extern void             reshapeviewport();
extern void             RGBcolor();
```

```
extern void             RGBcursor();
extern void             RGBmode();
extern void             RGBwritemask();
extern void             ringbell();
extern void             rmv();
extern void             rmv2();
extern void             rmv2i();
extern void             rmv2s();
extern void             rmvi();
extern void             rmvs();
extern void             rot();
extern void             rotate();
extern void             rpatch();
extern void             rpdr();
extern void             rpdr2();
extern void             rpdr2i();
extern void             rpdr2s();
extern void             rpdri();
extern void             rpdrs();
extern void             rpmv();
extern void             rpmv2();
extern void             rpmv2i();
extern void             rpmv2s();
extern void             rpmvi();
extern void             rpmvs();
extern void             scale();
extern void             screenspace();
extern void             scrmask();
extern void             select();
extern void             setbell();
extern void             setbutton();
extern void             setcursor();
extern void             setdblights();
extern void             setdepth();
extern Boolean          setfastcom();
extern void             setlinestyle();
extern void             setmap();
extern void             setmonitor();
extern void             setpattern();
extern void             setshade();
extern Boolean          setslowcom();
extern void             setvaluator();
extern void             shaderange();
extern void             singlebuffer();
extern void             spclos();
extern void             splf();
extern void             splf2();
extern void             splf2i();
extern void             splf2s();
extern void             splfi();
extern void             splfs();
extern void             stepunit();
extern long             strwidth();
```

```
extern void             swapbuffers();
extern void             swapinterval();
extern void             textcolor();
extern void             textinit();
extern void             textport();
extern void             textwritemask();
extern void             tie();
extern void             tpoff();
extern void             tpon();
extern void             translate();
extern void             unqdevice();
extern void             viewport();
extern long             winattach();
extern void             winclose();
extern void             winconstraints();
extern void             window();
extern long             winget();
extern void             winmove();
extern long             winopen();
extern void             winpop();
extern void             winposition();
extern void             winpush();
extern void             winreshape();
extern long             winset();
extern void             wintitle();
extern void             writemask();
extern void             writepixels();
extern void             writeRGB();
extern void             xfpt();
extern void             xfpt2();
extern void             xfpt2i();
extern void             xfpt2s();
extern void             xfpt4();
extern void             xfpt4i();
extern void             xfpt4s();
extern void             xfpti();
extern void             xfpts();
extern void             zbuffer();
extern void             zclear();
```

Another header file, *device.h*, defines symbolic names for many of the device numbers. You are not required to use it. In fact, you can define your own names for the subset of devices actually used in a particular application program.

```
/* macros to test valuator and button numbers */

#define ISBUTTON( b )       ( ((b)>=BUTOFFSET) &&
                            ((b)<(BUTCOUNT+BUTOFFSET)) )
```

```
#define ISVALUATOR( v )      ( ((v)>=VALOFFSET) &&
                           ((v)<(VALCOUNT+VALOFFSET)) )
#define ISTIMER( t )         ( ((t)>=TIMOFFSET) &&
                           ((t)<(TIMCOUNT+TIMOFFSET)) )
#define ISDIAL( t )          ( ((t)>=DIAL0    ) &&
                           ((t)<=DIAL8      ) )
#define ISLPEN( t )          ( ((t)==LPENX) || ((t)==LPENY) )
#define ISLPENBUT( t )       ( (t)==LPENBUT )
#define ISBPADBUT( t )       ( ((t)>=BPAD0) && ((t)<=BPAD3) )
#define ISSW( t )             ( ((t)>=SW0) && ((t)<=SW31) )
#define ISINPUT( t )         ( ((t)>=INOFFSET) &&
                           ((t)<(INCOUNT+INOFFSET)) )
#define ISOUTPUT( t )        ( ((t)>=OUTOFFSET) &&
                           ((t)<(OUTCOUNT+OUTOFFSET)) )
#define ISSTDKEYBD( t )      ( ((t)>=BUT0) &&
                           ((t)<=MAXKBDBUT) )
#define ISXKEYBD( t )        ( ((t)>=XKBDOFFSET) &&
        ((t)<(XKBDCOUNT+XKBDOFFSET)))
#define ISKEYBD( t )         ( ISSTDKEYBD(t) || ISXKEYBD(t) )

/* include file with device definitions */

#define NULLDEV          0
#define BUTOFFSET        1
#define VALOFFSET        256
#define TIMOFFSET        515
#define INOFFSET         1024
#define OUTOFFSET        1033
#define XKBDOFFSET       143

#define BUTCOUNT         171
#define VALCOUNT         20
#define TIMCOUNT         4
#define INCOUNT          8
#define OUTCOUNT         8
#define XKBDCOUNT        27

/* buttons */

#define BUT0                    1      /* 0+BUTOFFSET */
#define BUT1                    2      /* 1+BUTOFFSET */
#define BUT2                    3      /* 2+BUTOFFSET */
#define BUT3                    4      /* 3+BUTOFFSET */
#define BUT4                    5      /* 4+BUTOFFSET */
#define BUT5                    6      /* 5+BUTOFFSET */
#define BUT6                    7      /* 6+BUTOFFSET */
#define BUT7                    8      /* 7+BUTOFFSET */
#define BUT8                    9      /* 8+BUTOFFSET */
#define BUT9                    10     /* 9+BUTOFFSET */
#define BUT10                   11     /* 10+BUTOFFSET, A */
#define BUT11                   12     /* 11+BUTOFFSET, B */
#define BUT12                   13     /* 12+BUTOFFSET, C */
#define BUT13                   14     /* 13+BUTOFFSET, D */
```

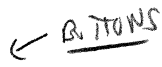
```
#define BUT14                15     /* 14+BUTOFFSET, E */
#define BUT15                16     /* 15+BUTOFFSET, F */
#define BUT16                17     /* 16+BUTOFFSET, 10 */
#define BUT17                18     /* 17+BUTOFFSET, 11 */
#define BUT18                19     /* 18+BUTOFFSET, 12 */
#define BUT19                20     /* 19+BUTOFFSET, 13 */
#define BUT20                21     /* 20+BUTOFFSET, 14 */
#define BUT21                22     /* 21+BUTOFFSET, 15 */
#define BUT22                23     /* 22+BUTOFFSET, 16 */
#define BUT23                24     /* 23+BUTOFFSET, 17 */
#define BUT24                25     /* 24+BUTOFFSET, 18 */
#define BUT25                26     /* 25+BUTOFFSET, 19 */
#define BUT26                27     /* 26+BUTOFFSET, 1A */
#define BUT27                28     /* 27+BUTOFFSET, 1B */
#define BUT28                29     /* 28+BUTOFFSET, 1C */
#define BUT29                30     /* 29+BUTOFFSET, 1D */
#define BUT30                31     /* 30+BUTOFFSET, 1E */
#define BUT31                32     /* 31+BUTOFFSET, 1F */
#define BUT32                33     /* 32+BUTOFFSET, 20 */
#define BUT33                34     /* 33+BUTOFFSET, 21 */
#define BUT34                35     /* 34+BUTOFFSET, 22 */
#define BUT35                36     /* 35+BUTOFFSET, 23 */
#define BUT36                37     /* 36+BUTOFFSET, 24 */
#define BUT37                38     /* 37+BUTOFFSET, 25 */
#define BUT38                39     /* 38+BUTOFFSET, 26 */
#define BUT39                40     /* 39+BUTOFFSET, 27 */
#define BUT40                41     /* 40+BUTOFFSET, 28 */
#define BUT41                42     /* 41+BUTOFFSET, 29 */
#define BUT42                43     /* 42+BUTOFFSET, 2A */
#define BUT43                44     /* 43+BUTOFFSET, 2B */
#define BUT44                45     /* 44+BUTOFFSET, 2C */
#define BUT45                46     /* 45+BUTOFFSET, 2D */
#define BUT46                47     /* 46+BUTOFFSET, 2E */
#define BUT47                48     /* 47+BUTOFFSET, 2F */
#define BUT48                49     /* 48+BUTOFFSET, 30 */
#define BUT49                50     /* 49+BUTOFFSET, 31 */
#define BUT50                51     /* 50+BUTOFFSET, 32 */
#define BUT51                52     /* 51+BUTOFFSET, 33 */
#define BUT52                53     /* 52+BUTOFFSET, 34 */
#define BUT53                54     /* 53+BUTOFFSET, 35 */
#define BUT54                55     /* 54+BUTOFFSET, 36 */
#define BUT55                56     /* 55+BUTOFFSET, 37 */
#define BUT56                57     /* 56+BUTOFFSET, 38 */
#define BUT57                58     /* 57+BUTOFFSET, 39 */
#define BUT58                59     /* 58+BUTOFFSET, 3A */
#define BUT59                60     /* 59+BUTOFFSET, 3B */
#define BUT60                61     /* 60+BUTOFFSET, 3C */
#define BUT61                62     /* 61+BUTOFFSET, 3D */
#define BUT62                63     /* 62+BUTOFFSET, 3E */
#define BUT63                64     /* 63+BUTOFFSET, 3F */
#define BUT64                65     /* 64+BUTOFFSET, 40 */
#define BUT65                66     /* 65+BUTOFFSET, 41 */
#define BUT66                67     /* 66+BUTOFFSET, 42 */
```

```
#define BUT67                   68      /* 67+BUTOFFSET, 43 */
#define BUT68                   69      /* 68+BUTOFFSET, 44 */
#define BUT69                   70      /* 69+BUTOFFSET, 45 */
#define BUT70                   71      /* 70+BUTOFFSET, 46 */
#define BUT71                   72      /* 71+BUTOFFSET, 47 */
#define BUT72                   73      /* 72+BUTOFFSET, 48 */
#define BUT73                   74      /* 73+BUTOFFSET, 49 */
#define BUT74                   75      /* 74+BUTOFFSET, 4A */
#define BUT75                   76      /* 75+BUTOFFSET, 4B */
#define BUT76                   77      /* 76+BUTOFFSET, 4C */
#define BUT77                   78      /* 77+BUTOFFSET, 4D */
#define BUT78                   79      /* 78+BUTOFFSET, 4E */
#define BUT79                   80      /* 79+BUTOFFSET, 4F */
#define BUT80                   81      /* 80+BUTOFFSET, 50 */
#define BUT81                   82      /* 81+BUTOFFSET, 51 */
#define BUT82                   83      /* 82+BUTOFFSET, 52 */
#define MAXKBDBUT               83      /* BUT82 */
#define BUT100                  101     /* 100+BUTOFFSET, Mouse button 1 */
#define BUT101                  102     /* 101+BUTOFFSET, Mouse button 2 */
#define BUT102                  103     /* 102+BUTOFFSET, Mouse button 3 */
#define BUT110                  111     /* 110+BUTOFFSET */
#define BUT111                  112     /* 111+BUTOFFSET */
#define BUT112                  113     /* 112+BUTOFFSET */
#define BUT113                  114     /* 113+BUTOFFSET */
#define BUT114                  115     /* 114+BUTOFFSET */
#define BUT115                  116     /* 115+BUTOFFSET */
#define BUT116                  117     /* 116+BUTOFFSET */
#define BUT117                  118     /* 117+BUTOFFSET */
#define BUT118                  119     /* 118+BUTOFFSET */
#define BUT119                  120     /* 119+BUTOFFSET */
#define BUT120                  121     /* 120+BUTOFFSET */
#define BUT121                  122     /* 121+BUTOFFSET */
#define BUT122                  123     /* 122+BUTOFFSET */
#define BUT123                  124     /* 123+BUTOFFSET */
#define BUT124                  125     /* 124+BUTOFFSET */
#define BUT125                  126     /* 125+BUTOFFSET */
#define BUT126                  127     /* 126+BUTOFFSET */
#define BUT127                  128     /* 127+BUTOFFSET */
#define BUT128                  129     /* 128+BUTOFFSET */
#define BUT129                  130     /* 129+BUTOFFSET */
#define BUT130                  131     /* 130+BUTOFFSET */
#define BUT131                  132     /* 131+BUTOFFSET */
#define BUT132                  133     /* 132+BUTOFFSET */
#define BUT133                  134     /* 133+BUTOFFSET */
#define BUT134                  135     /* 134+BUTOFFSET */
#define BUT135                  136     /* 135+BUTOFFSET */
#define BUT136                  137     /* 136+BUTOFFSET */
#define BUT137                  138     /* 137+BUTOFFSET */
#define BUT138                  139     /* 138+BUTOFFSET */
#define BUT139                  140     /* 139+BUTOFFSET */
#define BUT140                  141     /* 140+BUTOFFSET */
#define BUT141                  142     /* 141+BUTOFFSET */
#define BUT142                  143     /* 142+BUTOFFSET */
```

```
#define BUT143          144     /* 143+BUTOFFSET */
#define BUT144          145     /* 144+BUTOFFSET */
#define BUT145          146     /* 145+BUTOFFSET */
#define BUT146          147     /* 146+BUTOFFSET */
#define BUT147          148     /* 147+BUTOFFSET */
#define BUT148          149     /* 148+BUTOFFSET */
#define BUT149          150     /* 149+BUTOFFSET */
#define BUT150          151     /* 150+BUTOFFSET */
#define BUT151          152     /* 151+BUTOFFSET */
#define BUT152          153     /* 152+BUTOFFSET */
#define BUT153          154     /* 153+BUTOFFSET */
#define BUT154          155     /* 154+BUTOFFSET */
#define BUT155          156     /* 155+BUTOFFSET */
#define BUT156          157     /* 156+BUTOFFSET */
#define BUT157          158     /* 157+BUTOFFSET */
#define BUT158          159     /* 158+BUTOFFSET */
#define BUT159          160     /* 159+BUTOFFSET */
#define BUT160          161     /* 160+BUTOFFSET */
#define BUT161          162     /* 161+BUTOFFSET */
#define BUT162          163     /* 162+BUTOFFSET */
#define BUT163          164     /* 163+BUTOFFSET */
#define BUT164          165     /* 164+BUTOFFSET */
#define BUT165          166     /* 165+BUTOFFSET */
#define BUT166          167     /* 166+BUTOFFSET */
#define BUT167          168     /* 167+BUTOFFSET */
#define BUT168          169     /* 168+BUTOFFSET */

#define MOUSE1          101     /* BUT100 */
#define MOUSE2          102     /* BUT101 */
#define MOUSE3          103     /* BUT102 */
#define LEFTMOUSE       103     /* BUT102 */
#define MIDDLEMOUSE     102     /* BUT101 */
#define RIGHTMOUSE      101     /* BUT100 */
#define LPENBUT         104     /* LIGHT PEN BUTTON */
#define BPAD0           105     /* BITPAD BUTTON 0 */
#define BPAD1           106     /* BITPAD BUTTON 1 */
#define BPAD2           107     /* BITPAD BUTTON 2 */
#define BPAD3           108     /* BITPAD BUTTON 3 */
#define LPENVALID       109     /* LIGHT PEN VALID */

#define SWBASE          111     /* BUT110 */
#define SW0             111     /* SWBASE */
#define SW1             112     /* SWBASE+1 */
#define SW2             113     /* SWBASE+2 */
#define SW3             114     /* SWBASE+3 */
#define SW4             115     /* SWBASE+4 */
#define SW5             116     /* SWBASE+5 */
#define SW6             117     /* SWBASE+6 */
#define SW7             118     /* SWBASE+7 */
#define SW8             119     /* SWBASE+8 */
#define SW9             120     /* SWBASE+9 */
#define SW10            121     /* SWBASE+10 */
#define SW11            122     /* SWBASE+11 */
```



```
#define SW12                123    /* SWBASE+12 */
#define SW13                124    /* SWBASE+13 */
#define SW14                125    /* SWBASE+14 */
#define SW15                126    /* SWBASE+15 */
#define SW16                127    /* SWBASE+16 */
#define SW17                128    /* SWBASE+17 */
#define SW18                129    /* SWBASE+18 */
#define SW19                130    /* SWBASE+19 */
#define SW20                131    /* SWBASE+20 */
#define SW21                132    /* SWBASE+21 */
#define SW22                133    /* SWBASE+22 */
#define SW23                134    /* SWBASE+23 */
#define SW24                135    /* SWBASE+24 */
#define SW25                136    /* SWBASE+25 */
#define SW26                137    /* SWBASE+26 */
#define SW27                138    /* SWBASE+27 */
#define SW28                139    /* SWBASE+28 */
#define SW29                140    /* SWBASE+29 */
#define SW30                141    /* SWBASE+30 */
#define SW31                142    /* SWBASE+31 */

#define AKEY                11     /* BUT10 */
#define BKEY                36     /* BUT35 */
#define CKEY                28     /* BUT27 */
#define DKEY                18     /* BUT17 */
#define EKEY                17     /* BUT16 */
#define FKEY                19     /* BUT18 */
#define GKEY                26     /* BUT25 */
#define HKEY                27     /* BUT26 */
#define IKEY                40     /* BUT39 */
#define JKEY                34     /* BUT33 */
#define KKEY                35     /* BUT34 */
#define LKEY                42     /* BUT41 */
#define MKEY                44     /* BUT43 */
#define NKEY                37     /* BUT36 */
#define OKEY                41     /* BUT40 */
#define PKEY                48     /* BUT47 */
#define QKEY                10     /* BUT9 */
#define RKEY                24     /* BUT23 */
#define SKEY                12     /* BUT11 */
#define TKEY                25     /* BUT24 */
#define UKEY                33     /* BUT32 */
#define VKEY                29     /* BUT28 */
#define WKEY                16     /* BUT15 */
#define XKEY                21     /* BUT20 */
#define YKEY                32     /* BUT31 */
#define ZKEY                20     /* BUT19 */
#define ZEROKEY             46     /* BUT45 */
#define ONEKEY              8      /* BUT7 */
#define TWOKEY              14     /* BUT13 */
#define THREEKEY            15     /* BUT14 */
#define FOURKEY             22     /* BUT21 */
#define FIVEKEY             23     /* BUT22 */
```

```
#define SIXKEY                  30      /* BUT29 */
#define SEVENKEY                31      /* BUT30 */
#define EIGHTKEY                38      /* BUT37 */
#define NINEKEY                 39      /* BUT38 */
#define BREAKKEY                1       /* BUT0 */
#define SETUPKEY                2       /* BUT1 */
#define CTRLKEY                 3       /* BUT2 */
#define CAPSLOCKKEY             4       /* BUT3 */
#define RIGHTSHIFTKEY           5       /* BUT4 */
#define LEFTSHIFTKEY            6       /* BUT5 */
#define NOSCRLKEY               13      /* BUT12 */
#define ESCKEY                  7       /* BUT6 */
#define TABKEY                  9       /* BUT8 */
#define RETKEY                  51      /* BUT50 */
#define SPACEKEY                83      /* BUT82 */
#define LINEFEEDKEY             60      /* BUT59 */
#define BACKSPACEKEY            61      /* BUT60 */
#define DELKEY                  62      /* BUT61 */
#define SEMICOLONKEY            43      /* BUT42 */
#define PERIODKEY               52      /* BUT51 */
#define COMMAKEY                45      /* BUT44 */
#define QUOTEKEY                50      /* BUT49 */
#define ACCENTGRAVEKEY          55      /* BUT54 */
#define MINUSKEY                47      /* BUT46 */
#define VIRGULEKEY              53      /* BUT52 */
#define BACKSLASHKEY            57      /* BUT56 */
#define EQUALKEY                54      /* BUT53 */
#define LEFTBRACKETKEY          49      /* BUT48 */
#define RIGHTBRACKETKEY         56      /* BUT55 */
#define LEFTARROWKEY            73      /* BUT72 */
#define DOWNARROWKEY            74      /* BUT73 */
#define RIGHTARROWKEY           80      /* BUT79 */
#define UPARROWKEY              81      /* BUT80 */
#define PAD0                    59      /* BUT58 */
#define PAD1                    58      /* BUT57 */
#define PAD2                    64      /* BUT63 */
#define PAD3                    65      /* BUT64 */
#define PAD4                    63      /* BUT62 */
#define PAD5                    69      /* BUT68 */
#define PAD6                    70      /* BUT69 */
#define PAD7                    67      /* BUT66 */
#define PAD8                    68      /* BUT67 */
#define PAD9                    75      /* BUT74 */
#define PADPF1                  72      /* BUT71 */
#define PADPF2                  71      /* BUT70 */
#define PADPF3                  79      /* BUT78 */
#define PADPF4                  78      /* BUT77 */
#define PADPERIOD               66      /* BUT65 */
#define PADMINUS                76      /* BUT75 */
#define PADCOMMA                77      /* BUT76 */
#define PADENTER                82      /* BUT81 */

/* valuators */
```

```
#define SGIRESERVED             256     /* 0+VALOFFSET */
#define DIAL0                   257     /* 1+VALOFFSET */
#define DIAL1                   258     /* 2+VALOFFSET */
#define DIAL2                   259     /* 3+VALOFFSET */
#define DIAL3                   260     /* 4+VALOFFSET */
#define DIAL4                   261     /* 5+VALOFFSET */
#define DIAL5                   262     /* 6+VALOFFSET */
#define DIAL6                   263     /* 7+VALOFFSET */
#define DIAL7                   264     /* 8+VALOFFSET */
#define DIAL8                   265     /* 9+VALOFFSET */
#define MOUSEX                  266     /* 10+VALOFFSET */
#define MOUSEY                  267     /* 11+VALOFFSET */
#define LPENX                   268     /* 12+VALOFFSET */
#define LPENY                   269     /* 13+VALOFFSET */
#define BPADX                   270     /* 14+VALOFFSET */
#define BPADY                   271     /* 15+VALOFFSET */
#define CURSORX                 272     /* 16+VALOFFSET */
#define CURSORY                 273     /* 17+VALOFFSET */
#define GHOSTX                  274     /* 18+VALOFFSET */
#define GHOSTY                  275     /* 19+VALOFFSET */

/* timer */

#define TIMER0                  515     /* 0+TIMOFFSET */
#define TIMER1                  516     /* 1+TIMOFFSET */
#define TIMER2                  517     /* 2+TIMOFFSET */
#define TIMER3                  518     /* 3+TIMOFFSET */

/* misc devices */

#define KEYBD                   513     /* keyboard */
#define RAWKEYBD                514
/* raw keyboard for keyboard manager */
#define VALMARK                 523     /* valuator mark */
#define GERROR                  524     /* errors device */
#define REDRAW                  528
/* used by port manager to signal redraws */
#define WMSEND                  529     /* data in proc's shmem */
#define WMREPLY                 530     /* reply from port manager */
#define WMGFCLOSE               531     /* gf # is no longer being used */
#define WMTXCLOSE               532     /* tx # is no longer being used */
#define MODECHANGE              533     /* the display mode has changed */
#define INPUTCHANGE             534     /* input connected or disconnected */
#define QFULL                   535     /* queue was filled */
#define PIECECHANGE             536     /* change in the window pieces */
#define WINCLOSE                537     /* window close */

/* the extended keyboard */

#define LEFTALTKEY      143
#define RIGHTALTKEY     144
#define RIGHTCTRLKEY    145
#define F1KEY           146
```

```
#define F2KEY            147
#define F3KEY            148
#define F4KEY            149
#define F5KEY            150
#define F6KEY            151
#define F7KEY            152
#define F8KEY            153
#define F9KEY            154
#define F10KEY           155
#define F11KEY           156
#define F12KEY           157
#define PRINSCREENKEY    158
#define SCROLLLOCKKEY    159
#define PAUSEKEY         160
#define INSERTKEY        161
#define HOMEKEY          162
#define PAGEUPKEY        163
#define ENDKEY           164
#define PAGEDOWNKEY      165
#define NUMBLOCKKEY      166
#define PADVIRGULEKEY    167
#define PADASTERKEY      168
#define PADPLUSKEY       169

/* input channels */

#define INPUT0                1024   /* input channels */
#define INPUT1                1025
#define INPUT2                1026
#define INPUT3                1027
#define INPUT4                1028
#define INPUT5                1029
#define INPUT6                1030
#define INPUT7                1032

/* output channels */

#define OUTPUT0               1033   /* output channels */
#define OUTPUT1               1034
#define OUTPUT2               1035
#define OUTPUT3               1036
#define OUTPUT4               1037
#define OUTPUT5               1038
#define OUTPUT6               1039
#define OUTPUT7               1040

#define MENUBUTTON       RIGHTMOUSE /* THE menu button */
```

A third header file, *get.h*, defines constants for the values of global attributes, such as display mode and update buffers. You are not required to use this header file.

```
/* include file containing definitions for
returned values of get* routines */


/* values returned by getbuffer() */

#define NOBUFFER            0
#define BCKBUFFER           1
#define FRNTBUFFER          2
#define BOTHBUFFERS         3

/* values returned by getcmmode() */

#define CMAPMULTI           0
#define CMAPONE             1

/* values returned by getdisplaymode() */

#define DMRGB               0
#define DMSINGLE            1
#define DMDOUBLE            2

/* values returned by getmonitor() and getothermonitor() */

#define HZ30                0
#define HZ60                1
#define NTSC                2
#define HZ50                3
#define MONA                5
#define MONB                6
#define MONC                7
#define MOND                8
#define PAL                 9
#define MONSPECIAL          0x20

/* individual hit bits returned by gethitcode() */

#define LEFTPLANE           0x0001
#define RIGHTPLANE          0x0002
#define BOTTOMPLANE         0x0004
#define TOPPLANE            0x0008
#define NEARPLANE           0x0010
#define FARPLANE            0x0020
```

A.2 FORTRAN Definitions

This is a listing of *fgl.h*, which should be included in each IRIS program. It contains the type definitions, useful constants, and external definitions for all commands.

```
C     graphics library header file

C     maximum X and Y screen coordinates

      INTEGER*4 XMAXSC
      INTEGER*4 YMAXSC

C     various hardware/software limits

      INTEGER*4 ATTRIB
      INTEGER*4 VPSTAC
      INTEGER*4 MATRIX
      INTEGER*4 NAMEST
      INTEGER*4 STARTT
      INTEGER*4 ENDTAG

C     names for colors in color map loaded by ginit()

      INTEGER*4 BLACK
      INTEGER*4 RED
      INTEGER*4 GREEN
      INTEGER*4 YELLOW
      INTEGER*4 BLUE
      INTEGER*4 MAGENT
      INTEGER*4 CYAN
      INTEGER*4 WHITE

C     function return values

      INTEGER*4          BLKQRE
      INTEGER*4          DOPUP
      INTEGER*4          ENDFEE
      INTEGER*4          ENDPIC
      INTEGER*4          ENDSEL
      INTEGER*4          GENOBJ
      INTEGER*4          GENTAG
      INTEGER*4          GETBUF
      LOGICAL            GETBUT
      LOGICAL            GETCMM
      INTEGER*4          GETCOL
      LOGICAL            GETDCM
      INTEGER*4          GETDIS
      INTEGER*4          GETFON
      INTEGER*4          GETHEI
```

```
      INTEGER*4           GETHIT
      LOGICAL             GETLSB
      INTEGER*4           GETLSR
      INTEGER*4           GETLST
      INTEGER*4           GETLWI
      INTEGER*4           GETMAP
      INTEGER*4           GETMEM
      INTEGER*4           GETMON
      INTEGER*4           GETOPE
      INTEGER*4           GETPAT
      INTEGER*4           GETPLA
      LOGICAL             GETRES
      INTEGER*4           GETVAL
      INTEGER*4           GETWRI
      LOGICAL             GETZBU
      INTEGER*4           GVERSI
      LOGICAL             ISOBJ
      LOGICAL             ISQUEU
      LOGICAL             ISTAG
      INTEGER*4           NEWPUP
      INTEGER*4           QREAD
      INTEGER*4           QTEST
      INTEGER*4           READPI
      INTEGER*4           READRG
      INTEGER*4           STRWID
      INTEGER*4           WINOPE
      INTEGER*4           WINSET
      INTEGER*4           WINGET
      INTEGER*4           WINATT
      LOGICAL             SETSLO
      LOGICAL             SETFAS

C     maximum X and Y screen coordinates

      PARAMETER (XMAXSC = 1023)
      PARAMETER (YMAXSC = 767)

C     various hardware/software limits

      PARAMETER (ATTRIB = 10)
      PARAMETER (VPSTAC = 8)
      PARAMETER (MATRIX = 32)
      PARAMETER (NAMEST = 1025)
      PARAMETER (STARTT = -2)
      PARAMETER (ENDTAG  = -3)

C     names for colors in color map loaded by ginit()

      PARAMETER (BLACK = 0)
      PARAMETER (RED   = 1)
      PARAMETER (GREEN = 2)
      PARAMETER (YELLOW= 3)
      PARAMETER (BLUE  = 4)
```

```
      PARAMETER (MAGENT= 5)
      PARAMETER (CYAN   = 6)
      PARAMETER (WHITE = 7)
```

Another header file, *fdevice.h*, defines symbolic names for many of the device numbers. You are not required to use it. In fact, you can define your own names for the subset of devices actually used in a particular application program.

```
C     include file with device definitions

      INTEGER*4 NULLDE
      INTEGER*4 BUTOFF
      INTEGER*4 VALOFF
      INTEGER*4 KEYOFF
      INTEGER*4 TIMOFF

      INTEGER*4 BUTCOU
      INTEGER*4 VALCOU
      INTEGER*4 TIMCOU

C     buttons

      INTEGER*4 BUT0
      INTEGER*4 BUT1
      INTEGER*4 BUT2
      INTEGER*4 BUT3
      INTEGER*4 BUT4
      INTEGER*4 BUT5
      INTEGER*4 BUT6
      INTEGER*4 BUT7
      INTEGER*4 BUT8
      INTEGER*4 BUT9
      INTEGER*4 BUT10
      INTEGER*4 BUT11
      INTEGER*4 BUT12
      INTEGER*4 BUT13
      INTEGER*4 BUT14
      INTEGER*4 BUT15
      INTEGER*4 BUT16
      INTEGER*4 BUT17
      INTEGER*4 BUT18
      INTEGER*4 BUT19
      INTEGER*4 BUT20
      INTEGER*4 BUT21
      INTEGER*4 BUT22
      INTEGER*4 BUT23
      INTEGER*4 BUT24
      INTEGER*4 BUT25
```

```
INTEGER*4 BUT26
INTEGER*4 BUT27
INTEGER*4 BUT28
INTEGER*4 BUT29
INTEGER*4 BUT30
INTEGER*4 BUT31
INTEGER*4 BUT32
INTEGER*4 BUT33
INTEGER*4 BUT34
INTEGER*4 BUT35
INTEGER*4 BUT36
INTEGER*4 BUT37
INTEGER*4 BUT38
INTEGER*4 BUT39
INTEGER*4 BUT40
INTEGER*4 BUT41
INTEGER*4 BUT42
INTEGER*4 BUT43
INTEGER*4 BUT44
INTEGER*4 BUT45
INTEGER*4 BUT46
INTEGER*4 BUT47
INTEGER*4 BUT48
INTEGER*4 BUT49
INTEGER*4 BUT50
INTEGER*4 BUT51
INTEGER*4 BUT52
INTEGER*4 BUT53
INTEGER*4 BUT54
INTEGER*4 BUT55
INTEGER*4 BUT56
INTEGER*4 BUT57
INTEGER*4 BUT58
INTEGER*4 BUT59
INTEGER*4 BUT60
INTEGER*4 BUT61
INTEGER*4 BUT62
INTEGER*4 BUT63
INTEGER*4 BUT64
INTEGER*4 BUT65
INTEGER*4 BUT66
INTEGER*4 BUT67
INTEGER*4 BUT68
INTEGER*4 BUT69
INTEGER*4 BUT70
INTEGER*4 BUT71
INTEGER*4 BUT72
INTEGER*4 BUT73
INTEGER*4 BUT74
INTEGER*4 BUT75
INTEGER*4 BUT76
INTEGER*4 BUT77
INTEGER*4 BUT78
```

```
INTEGER*4 BUT79
INTEGER*4 BUT80
INTEGER*4 BUT81
INTEGER*4 BUT82
INTEGER*4 MAXKBD
INTEGER*4 BUT100
INTEGER*4 BUT101
INTEGER*4 BUT102
INTEGER*4 BUT110
INTEGER*4 BUT111
INTEGER*4 BUT112
INTEGER*4 BUT113
INTEGER*4 BUT114
INTEGER*4 BUT115
INTEGER*4 BUT116
INTEGER*4 BUT117
INTEGER*4 BUT118
INTEGER*4 BUT119
INTEGER*4 BUT120
INTEGER*4 BUT121
INTEGER*4 BUT122
INTEGER*4 BUT123
INTEGER*4 BUT124
INTEGER*4 BUT125
INTEGER*4 BUT126
INTEGER*4 BUT127
INTEGER*4 BUT128
INTEGER*4 BUT129
INTEGER*4 BUT130
INTEGER*4 BUT131
INTEGER*4 BUT132
INTEGER*4 BUT133
INTEGER*4 BUT134
INTEGER*4 BUT135
INTEGER*4 BUT136
INTEGER*4 BUT137
INTEGER*4 BUT138
INTEGER*4 BUT139
INTEGER*4 BUT140
INTEGER*4 BUT141
INTEGER*4 BUT142
INTEGER*4 BUT143
INTEGER*4 BUT144
INTEGER*4 BUT145
INTEGER*4 BUT146
INTEGER*4 BUT147
INTEGER*4 BUT148
INTEGER*4 BUT149
INTEGER*4 BUT150
INTEGER*4 BUT151
INTEGER*4 BUT152
INTEGER*4 BUT153
INTEGER*4 BUT154
```

```
INTEGER*4 BUT155
INTEGER*4 BUT156
INTEGER*4 BUT157
INTEGER*4 BUT158
INTEGER*4 BUT159
INTEGER*4 BUT160
INTEGER*4 BUT161
INTEGER*4 BUT162
INTEGER*4 BUT163
INTEGER*4 BUT164
INTEGER*4 BUT165
INTEGER*4 BUT166
INTEGER*4 BUT167
INTEGER*4 BUT168
INTEGER*4 MOUSE1
INTEGER*4 MOUSE2
INTEGER*4 MOUSE3
INTEGER*4 LEFTMO
INTEGER*4 MIDDLE
INTEGER*4 RIGHTM
INTEGER*4 MENUBU
INTEGER*4 LPENBU
INTEGER*4 BPAD0
INTEGER*4 BPAD1
INTEGER*4 BPAD2
INTEGER*4 BPAD3
INTEGER*4 LPENVA
INTEGER*4 SWBASE
INTEGER*4 SW0
INTEGER*4 SW1
INTEGER*4 SW2
INTEGER*4 SW3
INTEGER*4 SW4
INTEGER*4 SW5
INTEGER*4 SW6
INTEGER*4 SW7
INTEGER*4 SW8
INTEGER*4 SW9
INTEGER*4 SW10
INTEGER*4 SW11
INTEGER*4 SW12
INTEGER*4 SW13
INTEGER*4 SW14
INTEGER*4 SW15
INTEGER*4 SW16
INTEGER*4 SW17
INTEGER*4 SW18
INTEGER*4 SW19
INTEGER*4 SW20
INTEGER*4 SW21
INTEGER*4 SW22
INTEGER*4 SW23
INTEGER*4 SW24
```

```
      INTEGER*4 SW25
      INTEGER*4 SW26
      INTEGER*4 SW27
      INTEGER*4 SW28
      INTEGER*4 SW29
      INTEGER*4 SW30
      INTEGER*4 SW31

      INTEGER*4 AKEY
      INTEGER*4 BKEY
      INTEGER*4 CKEY
      INTEGER*4 DKEY
      INTEGER*4 EKEY
      INTEGER*4 FKEY
      INTEGER*4 GKEY
      INTEGER*4 HKEY
      INTEGER*4 IKEY
      INTEGER*4 JKEY
      INTEGER*4 KKEY
      INTEGER*4 LKEY
      INTEGER*4 MKEY
      INTEGER*4 NKEY
      INTEGER*4 OKEY
      INTEGER*4 PKEY
      INTEGER*4 QKEY
      INTEGER*4 RKEY
      INTEGER*4 SKEY
      INTEGER*4 TKEY
      INTEGER*4 UKEY
      INTEGER*4 VKEY
      INTEGER*4 WKEY
      INTEGER*4 XKEY
      INTEGER*4 YKEY
      INTEGER*4 ZKEY
      INTEGER*4 ZEROKE
      INTEGER*4 ONEKEY
      INTEGER*4 TWOKEY
      INTEGER*4 THREEK
      INTEGER*4 FOURKE
      INTEGER*4 FIVEKE
      INTEGER*4 SIXKEY
      INTEGER*4 SEVENK
      INTEGER*4 EIGHTK
      INTEGER*4 NINEKE
      INTEGER*4 BREAKK
      INTEGER*4 SETUPK
      INTEGER*4 CTRLKE
      INTEGER*4 CAPSLO
      INTEGER*4 RIGHTS
      INTEGER*4 LEFTSH
      INTEGER*4 NOSCRL
      INTEGER*4 ESCKEY
      INTEGER*4 TABKEY
```

```
      INTEGER*4 RETKEY
      INTEGER*4 SPACEK
      INTEGER*4 LINEFE
      INTEGER*4 BACKSP
      INTEGER*4 DELKEY
      INTEGER*4 SEMICO
      INTEGER*4 PERIOD
      INTEGER*4 COMMAK
      INTEGER*4 QUOTEK
      INTEGER*4 ACCENT
      INTEGER*4 MINUSK
      INTEGER*4 VIRGUL
      INTEGER*4 BACKSL
      INTEGER*4 EQUALK
      INTEGER*4 LEFTBR
      INTEGER*4 RIGHTB
      INTEGER*4 LEFTAR
      INTEGER*4 DOWNAR
      INTEGER*4 RIGHTA
      INTEGER*4 UPARRO
      INTEGER*4 PAD0
      INTEGER*4 PAD1
      INTEGER*4 PAD2
      INTEGER*4 PAD3
      INTEGER*4 PAD4
      INTEGER*4 PAD5
      INTEGER*4 PAD6
      INTEGER*4 PAD7
      INTEGER*4 PAD8
      INTEGER*4 PAD9
      INTEGER*4 PADPF1
      INTEGER*4 PADPF2
      INTEGER*4 PADPF3
      INTEGER*4 PADPF4
      INTEGER*4 PADPER
      INTEGER*4 PADMIN
      INTEGER*4 PADCOM
      INTEGER*4 PADENT

C     screen buttons
C     Hashing algorithm: SBUTx = SCRBUTx

      INTEGER*4 SBUT0
      INTEGER*4 SBUT1
      INTEGER*4 SBUT2
      INTEGER*4 SBUT3
      INTEGER*4 SBUT4
      INTEGER*4 SBUT5
      INTEGER*4 SBUT6
      INTEGER*4 SBUT7
      INTEGER*4 SBUT8
      INTEGER*4 SBUT9
      INTEGER*4 SBUT10
```

```
      INTEGER*4 SBUT11
      INTEGER*4 SBUT12
      INTEGER*4 SBUT13
      INTEGER*4 SBUT14
      INTEGER*4 SBUT15
      INTEGER*4 SBUT16

C     valuators

      INTEGER*4 SGIRES
      INTEGER*4 DIAL0
      INTEGER*4 DIAL1
      INTEGER*4 DIAL2
      INTEGER*4 DIAL3
      INTEGER*4 DIAL4
      INTEGER*4 DIAL5
      INTEGER*4 DIAL6
      INTEGER*4 DIAL7
      INTEGER*4 DIAL8
      INTEGER*4 MOUSEX
      INTEGER*4 MOUSEY
      INTEGER*4 LPENX
      INTEGER*4 LPENY
      INTEGER*4 BPADX
      INTEGER*4 BPADY
      INTEGER*4 CURSRX
      INTEGER*4 CURSRY
      INTEGER*4 GHOSTX
      INTEGER*4 GHOSTY

C     timers

      INTEGER*4 TIMER0
      INTEGER*4 TIMER1
      INTEGER*4 TIMER2
      INTEGER*4 TIMER3

C     misc devices
C     Hash algorithm: CURSRX = CURSORX

      INTEGER*4 KEYBD
      INTEGER*4 RAWKBD
      INTEGER*4 VALMAR
      INTEGER*4 GERROR
      INTEGER*4 REDRAW
      INTEGER*4 WMSEND
      INTEGER*4 WMREPL
      INTEGER*4 WMGFCL
      INTEGER*4 WMTXCL
      INTEGER*4 MODECH
      INTEGER*4 INPTCH
      INTEGER*4 QFULL
      INTEGER*4 PIECHN
```

```
      INTEGER*4 WINCLO

C     include file with device definitions

      PARAMETER ( NULLDE = 0 )
      PARAMETER ( BUTOFF = 1 )
      PARAMETER ( VALOFF = 256 )
      PARAMETER ( KEYOFF = 512 )
      PARAMETER ( TIMOFF = 515 )

      PARAMETER ( BUTCOU = 144 )
      PARAMETER ( VALCOU = 20  )
      PARAMETER ( TIMCOU = 4   )

C     buttons

      PARAMETER ( BUT0 = 1 )
      PARAMETER ( BUT1 = 2 )
      PARAMETER ( BUT2 = 3 )
      PARAMETER ( BUT3 = 4 )
      PARAMETER ( BUT4 = 5 )
      PARAMETER ( BUT5 = 6 )
      PARAMETER ( BUT6 = 7 )
      PARAMETER ( BUT7 = 8 )
      PARAMETER ( BUT8 = 9 )
      PARAMETER ( BUT9 = 10 )
      PARAMETER ( BUT10 = 11 )
      PARAMETER ( BUT11 = 12 )
      PARAMETER ( BUT12 = 13 )
      PARAMETER ( BUT13 = 14 )
      PARAMETER ( BUT14 = 15 )
      PARAMETER ( BUT15 = 16 )
      PARAMETER ( BUT16 = 17 )
      PARAMETER ( BUT17 = 18 )
      PARAMETER ( BUT18 = 19 )
      PARAMETER ( BUT19 = 20 )
      PARAMETER ( BUT20 = 21 )
      PARAMETER ( BUT21 = 22 )
      PARAMETER ( BUT22 = 23 )
      PARAMETER ( BUT23 = 24 )
      PARAMETER ( BUT24 = 25 )
      PARAMETER ( BUT25 = 26 )
      PARAMETER ( BUT26 = 27 )
      PARAMETER ( BUT27 = 28 )
      PARAMETER ( BUT28 = 29 )
      PARAMETER ( BUT29 = 30 )
      PARAMETER ( BUT30 = 31 )
      PARAMETER ( BUT31 = 32 )
      PARAMETER ( BUT32 = 33 )
      PARAMETER ( BUT33 = 34 )
      PARAMETER ( BUT34 = 35 )
      PARAMETER ( BUT35 = 36 )
      PARAMETER ( BUT36 = 37 )
```

```
PARAMETER ( BUT37 = 38 )
PARAMETER ( BUT38 = 39 )
PARAMETER ( BUT39 = 40 )
PARAMETER ( BUT40 = 41 )
PARAMETER ( BUT41 = 42 )
PARAMETER ( BUT42 = 43 )
PARAMETER ( BUT43 = 44 )
PARAMETER ( BUT44 = 45 )
PARAMETER ( BUT45 = 46 )
PARAMETER ( BUT46 = 47 )
PARAMETER ( BUT47 = 48 )
PARAMETER ( BUT48 = 49 )
PARAMETER ( BUT49 = 50 )
PARAMETER ( BUT50 = 51 )
PARAMETER ( BUT51 = 52 )
PARAMETER ( BUT52 = 53 )
PARAMETER ( BUT53 = 54 )
PARAMETER ( BUT54 = 55 )
PARAMETER ( BUT55 = 56 )
PARAMETER ( BUT56 = 57 )
PARAMETER ( BUT57 = 58 )
PARAMETER ( BUT58 = 59 )
PARAMETER ( BUT59 = 60 )
PARAMETER ( BUT60 = 61 )
PARAMETER ( BUT61 = 62 )
PARAMETER ( BUT62 = 63 )
PARAMETER ( BUT63 = 64 )
PARAMETER ( BUT64 = 65 )
PARAMETER ( BUT65 = 66 )
PARAMETER ( BUT66 = 67 )
PARAMETER ( BUT67 = 68 )
PARAMETER ( BUT68 = 69 )
PARAMETER ( BUT69 = 70 )
PARAMETER ( BUT70 = 71 )
PARAMETER ( BUT71 = 72 )
PARAMETER ( BUT72 = 73 )
PARAMETER ( BUT73 = 74 )
PARAMETER ( BUT74 = 75 )
PARAMETER ( BUT75 = 76 )
PARAMETER ( BUT76 = 77 )
PARAMETER ( BUT77 = 78 )
PARAMETER ( BUT78 = 79 )
PARAMETER ( BUT79 = 80 )
PARAMETER ( BUT80 = 81 )
PARAMETER ( BUT81 = 82 )
PARAMETER ( BUT82 = 83 )
PARAMETER ( MAXKBD = 83 )
PARAMETER ( BUT100 = 101 )
PARAMETER ( BUT101 = 102 )
PARAMETER ( BUT102 = 103 )
PARAMETER ( BUT110 = 111 )
PARAMETER ( BUT111 = 112 )
PARAMETER ( BUT112 = 113 )
```

```
PARAMETER ( BUT113 = 114 )
PARAMETER ( BUT114 = 115 )
PARAMETER ( BUT115 = 116 )
PARAMETER ( BUT116 = 117 )
PARAMETER ( BUT117 = 118 )
PARAMETER ( BUT118 = 119 )
PARAMETER ( BUT119 = 120 )
PARAMETER ( BUT120 = 121 )
PARAMETER ( BUT121 = 122 )
PARAMETER ( BUT122 = 123 )
PARAMETER ( BUT123 = 124 )
PARAMETER ( BUT124 = 125 )
PARAMETER ( BUT125 = 126 )
PARAMETER ( BUT126 = 127 )
PARAMETER ( BUT127 = 128 )
PARAMETER ( BUT128 = 129 )
PARAMETER ( BUT129 = 130 )
PARAMETER ( BUT130 = 131 )
PARAMETER ( BUT131 = 132 )
PARAMETER ( BUT132 = 133 )
PARAMETER ( BUT133 = 134 )
PARAMETER ( BUT134 = 135 )
PARAMETER ( BUT135 = 136 )
PARAMETER ( BUT136 = 137 )
PARAMETER ( BUT137 = 138 )
PARAMETER ( BUT138 = 139 )
PARAMETER ( BUT139 = 140 )
PARAMETER ( BUT140 = 141 )
PARAMETER ( BUT141 = 142 )
PARAMETER ( BUT142 = 143 )
PARAMETER ( BUT143 = 144 )
PARAMETER ( BUT144 = 145 )
PARAMETER ( BUT145 = 146 )
PARAMETER ( BUT146 = 147 )
PARAMETER ( BUT147 = 148 )
PARAMETER ( BUT148 = 149 )
PARAMETER ( BUT149 = 150 )
PARAMETER ( BUT150 = 151 )
PARAMETER ( BUT151 = 152 )
PARAMETER ( BUT152 = 153 )
PARAMETER ( BUT153 = 154 )
PARAMETER ( BUT154 = 155 )
PARAMETER ( BUT155 = 156 )
PARAMETER ( BUT156 = 157 )
PARAMETER ( BUT157 = 158 )
PARAMETER ( BUT158 = 159 )
PARAMETER ( BUT159 = 160 )
PARAMETER ( BUT160 = 161 )
PARAMETER ( BUT161 = 162 )
PARAMETER ( BUT162 = 163 )
PARAMETER ( BUT163 = 164 )
PARAMETER ( BUT164 = 165 )
PARAMETER ( BUT165 = 166 )
```

```
PARAMETER ( BUT166 = 167 )
PARAMETER ( BUT167 = 168 )
PARAMETER ( BUT168 = 169 )
PARAMETER ( MOUSE1 = 101 )
PARAMETER ( MOUSE2 = 102 )
PARAMETER ( MOUSE3 = 103 )
PARAMETER ( LEFTMO = 103 )
PARAMETER ( MIDDLE = 102 )
PARAMETER ( RIGHTM = 101 )
PARAMETER ( MENUBU = 101 )
PARAMETER ( LPENBU = 104 )
PARAMETER ( BPAD0 = 105 )
PARAMETER ( BPAD1 = 106 )
PARAMETER ( BPAD2 = 107 )
PARAMETER ( BPAD3 = 108 )
PARAMETER ( LPENVA = 109 )
PARAMETER ( SWBASE = 111 )
PARAMETER ( SW0 = 111 )
PARAMETER ( SW1 = 112 )
PARAMETER ( SW2 = 113 )
PARAMETER ( SW3 = 114 )
PARAMETER ( SW4 = 115 )
PARAMETER ( SW5 = 116 )
PARAMETER ( SW6 = 117 )
PARAMETER ( SW7 = 118 )
PARAMETER ( SW8 = 119 )
PARAMETER ( SW9 = 120 )
PARAMETER ( SW10 = 121 )
PARAMETER ( SW11 = 122 )
PARAMETER ( SW12 = 123 )
PARAMETER ( SW13 = 124 )
PARAMETER ( SW14 = 125 )
PARAMETER ( SW15 = 126 )
PARAMETER ( SW16 = 127 )
PARAMETER ( SW17 = 128 )
PARAMETER ( SW18 = 129 )
PARAMETER ( SW19 = 130 )
PARAMETER ( SW20 = 131 )
PARAMETER ( SW21 = 132 )
PARAMETER ( SW22 = 133 )
PARAMETER ( SW23 = 134 )
PARAMETER ( SW24 = 135 )
PARAMETER ( SW25 = 136 )
PARAMETER ( SW26 = 137 )
PARAMETER ( SW27 = 138 )
PARAMETER ( SW28 = 139 )
PARAMETER ( SW29 = 140 )
PARAMETER ( SW30 = 141 )
PARAMETER ( SW31 = 142 )
PARAMETER ( AKEY = 11 )
PARAMETER ( BKEY = 36 )
PARAMETER ( CKEY = 28 )
PARAMETER ( DKEY = 18 )
```

```
PARAMETER ( EKEY = 17 )
PARAMETER ( FKEY = 19 )
PARAMETER ( GKEY = 26 )
PARAMETER ( HKEY = 27 )
PARAMETER ( IKEY = 40 )
PARAMETER ( JKEY = 34 )
PARAMETER ( KKEY = 35 )
PARAMETER ( LKEY = 42 )
PARAMETER ( MKEY = 44 )
PARAMETER ( NKEY = 37 )
PARAMETER ( OKEY = 41 )
PARAMETER ( PKEY = 48 )
PARAMETER ( QKEY = 10 )
PARAMETER ( RKEY = 24 )
PARAMETER ( SKEY = 12 )
PARAMETER ( TKEY = 25 )
PARAMETER ( UKEY = 33 )
PARAMETER ( VKEY = 29 )
PARAMETER ( WKEY = 16 )
PARAMETER ( XKEY = 21 )
PARAMETER ( YKEY = 32 )
PARAMETER ( ZKEY = 20 )
PARAMETER ( ZEROKE = 46 )
PARAMETER ( ONEKEY = 8 )
PARAMETER ( TWOKEY = 14 )
PARAMETER ( THREEK = 15 )
PARAMETER ( FOURKE = 22 )
PARAMETER ( FIVEKE = 23 )
PARAMETER ( SIXKEY = 30 )
PARAMETER ( SEVENK = 31 )
PARAMETER ( EIGHTK = 38 )
PARAMETER ( NINEKE = 39 )
PARAMETER ( BREAKK = 1 )
PARAMETER ( SETUPK = 2 )
PARAMETER ( CTRLKE = 3 )
PARAMETER ( CAPSLO = 4 )
PARAMETER ( RIGHTS = 5 )
PARAMETER ( LEFTSH = 6 )
PARAMETER ( NOSCRL = 13 )
PARAMETER ( ESCKEY = 7 )
PARAMETER ( TABKEY = 9 )
PARAMETER ( RETKEY = 51 )
PARAMETER ( SPACEK = 83 )
PARAMETER ( LINEFE = 60 )
PARAMETER ( BACKSP = 61 )
PARAMETER ( DELKEY = 62 )
PARAMETER ( SEMICO = 43 )
PARAMETER ( PERIOD = 52 )
PARAMETER ( COMMAK = 45 )
PARAMETER ( QUOTEK = 50 )
PARAMETER ( ACCENT = 55 )
PARAMETER ( MINUSK = 47 )
PARAMETER ( VIRGUL = 53 )
```

```
      PARAMETER ( BACKSL = 57 )
      PARAMETER ( EQUALK = 54 )
      PARAMETER ( LEFTBR = 49 )
      PARAMETER ( RIGHTB = 56 )
      PARAMETER ( LEFTAR = 73 )
      PARAMETER ( DOWNAR = 74 )
      PARAMETER ( RIGHTA = 80 )
      PARAMETER ( UPARRO = 81 )
      PARAMETER ( PAD0 = 59 )
      PARAMETER ( PAD1 = 58 )
      PARAMETER ( PAD2 = 64 )
      PARAMETER ( PAD3 = 65 )
      PARAMETER ( PAD4 = 63 )
      PARAMETER ( PAD5 = 69 )
      PARAMETER ( PAD6 = 70 )
      PARAMETER ( PAD7 = 67 )
      PARAMETER ( PAD8 = 68 )
      PARAMETER ( PAD9 = 75 )
      PARAMETER ( PADPF1 = 72 )
      PARAMETER ( PADPF2 = 71 )
      PARAMETER ( PADPF3 = 79 )
      PARAMETER ( PADPF4 = 78 )
      PARAMETER ( PADPER = 66 )
      PARAMETER ( PADMIN = 76 )
      PARAMETER ( PADCOM = 77 )
      PARAMETER ( PADENT = 82 )

C     valuators

      PARAMETER ( SGIRES = 256 )
      PARAMETER ( DIAL0 = 257 )
      PARAMETER ( DIAL1 = 258 )
      PARAMETER ( DIAL2 = 259 )
      PARAMETER ( DIAL3 = 260 )
      PARAMETER ( DIAL4 = 261 )
      PARAMETER ( DIAL5 = 262 )
      PARAMETER ( DIAL6 = 263 )
      PARAMETER ( DIAL7 = 264 )
      PARAMETER ( DIAL8 =  265 )
      PARAMETER ( MOUSEX = 266 )
      PARAMETER ( MOUSEY = 267 )
      PARAMETER ( LPENX =  268 )
      PARAMETER ( LPENY =  269 )
      PARAMETER ( BPADX =  270 )
      PARAMETER ( BPADY =  271 )
      PARAMETER ( CURSRX =  272 )
      PARAMETER ( CURSRY =  273 )
      PARAMETER ( GHOSTX =  274 )
      PARAMETER ( GHOSTY =  275 )

C  timers

      PARAMETER ( TIMER0 = 515 )
```

```
      PARAMETER ( TIMER1 = 516 )
      PARAMETER ( TIMER2 = 517 )
      PARAMETER ( TIMER3 = 518 )

C  misc devices

      PARAMETER ( KEYBD = 513 )
      PARAMETER ( RAWKBD = 514 )
      PARAMETER ( VALMAR =  523 )
      PARAMETER ( GERROR =  524 )
      PARAMETER ( REDRAW =  528 )
      PARAMETER ( WMSEND =  529 )
      PARAMETER ( WMREPL =  530 )
      PARAMETER ( WMGFCL =  531 )
      PARAMETER ( WMTXCL =  532 )
      PARAMETER ( MODECH =  533 )
      PARAMETER ( INPTCH =  534 )
      PARAMETER ( QFULL =  535 )
      PARAMETER ( PIECHN =  536 )
      PARAMETER ( WINCLO =  537 )

C  the extended keyboard

      PARAMETER ( LEFTALTKEY   = 143 )
      PARAMETER ( RIGHTALTKEY  = 144 )
      PARAMETER ( RIGHTCTRLKEY = 145 )
      PARAMETER ( F1KEY = 146 )
      PARAMETER ( F2KEY = 147 )
      PARAMETER ( F3KEY = 148 )
      PARAMETER ( F4KEY = 149 )
      PARAMETER ( F5KEY = 150 )
      PARAMETER ( F6KEY = 151 )
      PARAMETER ( F7KEY = 152 )
      PARAMETER ( F8KEY = 153 )
      PARAMETER ( F9KEY = 154 )
      PARAMETER ( F10KEY = 155 )
      PARAMETER ( F11KEY = 156 )
      PARAMETER ( F12KEY = 157 )
      PARAMETER ( PRINTSCREENKEY = 158 )
      PARAMETER ( SCROLLLOCKKEY   = 159 )
      PARAMETER ( PAUSEKEY        = 160 )
      PARAMETER ( INSERTKEY       = 161 )
      PARAMETER ( HOMEKEY = 162 )
      PARAMETER ( PAGEUPKEY       = 163 )
      PARAMETER ( ENDKEY  = 164 )
      PARAMETER ( PAGEDOWNKEY     = 165 )
      PARAMETER ( NUMLOCKKEY      = 166 )
      PARAMETER ( PADVIRGULEKEY   = 167 )
      PARAMETER ( PADASTERKEY     = 168 )
      PARAMETER ( PADPLUSKEY      = 169 )
```

A third file, *fget.h*, defines constants for the values of global attributes, such as display mode and update buffers. You are not required to use this header file.

```
      integer*4 NOBUFF
      integer*4 BCKBUF
      integer*4 FRNTBU
      integer*4 BOTHBU
      integer*4 CMAPMU
      integer*4 CMAPON
      integer*4 DMRGB
      integer*4 DMSING
      integer*4 DMDOUB
      integer*4 HZ30
      integer*4 HZ60
      integer*4 NTSC
      integer*4 PAL
      integer*4 HZ50
      integer*4 MONA
      integer*4 MONB
      integer*4 MONC
      integer*4 MOND
      integer*4 MONSPE
      integer*4 FARPLA
      integer*4 NEARPL
      integer*4 TOPPLA
      integer*4 BOTTOM
      integer*4 RIGHTP
      integer*4 LEFTPL

c include file containing definitions for
  returned values of get* routines


c values returned by getbuffer()

      PARAMETER ( NOBUFF =        0 )
      PARAMETER ( BCKBUF =        1 )
      PARAMETER ( FRNTBU =        2 )
      PARAMETER ( BOTHBU =        3 )

c values returned by getcmmode()

      PARAMETER ( CMAPMU =        0 )
      PARAMETER ( CMAPON =        1 )

c values returned by getdisplaymode()

      PARAMETER ( DMRGB =0 )
      PARAMETER ( DMSING =        1 )
      PARAMETER ( DMDOUB =        2 )
```

```
c values returned by getmonitor()

      PARAMETER ( HZ30    =        0 )
      PARAMETER ( HZ60    =        1 )
      PARAMETER ( NTSC    =        2 )
      PARAMETER ( PAL     =        2 )
      PARAMETER ( HZ50    =        3 )
      PARAMETER ( MONA    =        5 )
      PARAMETER ( MONB    =        6 )
      PARAMETER ( MONC    =        7 )
      PARAMETER ( MOND    =        8 )
      PARAMETER ( MONSPE  =        32 )

c individual hit bits returned by gethitcode()

      PARAMETER ( FARPLA  =        1 )
      PARAMETER ( NEARPL  =        2 )
      PARAMETER ( TOPPLA  =        4 )
      PARAMETER ( BOTTOM  =        8 )
      PARAMETER ( RIGHTP  =        16 )
      PARAMETER ( LEFTPL  =        32 )
```

Appendix B: Geometry Engine Computations

The Geometry Engine system has three parts:

- **Matrix Subsystem** - A stack of 4x4 floating point matrices transform coordinate data.
- **Clipping Subsystem** - Transformed coordinate data is clipped to a 2D or 3D window.
- **Scaling Subsystem** - Clipped 2D or 3D coordinate data is scaled to the dimensions of the output device. In 3D, the system provides orthographic or perspective projection as well.

The Matrix Subsystem does the following computation:

$$[\ x\ y\ z\ w\ ] = [\ x'\ y'\ z'\ w'\ ]\ M$$

where the current transformation matrix is M and the vector to be transformed is $[\ x'\ y'\ z'\ w'\ ]$. The coordinates $[\ x\ y\ z\ w\ ]$ are given to the Clipping Subsystem. The clipped results satisfy the following equations:

$$-w < x < w$$
$$-w < y < w$$
$$-w < z < w$$

After clipping, the Scaling Subsystem does the final mapping to screen coordinates with the following computations:

$$X_{screen} = \frac{x}{w} \times V_{sx} + V_{cx}$$

$$Y_{screen} = \frac{y}{w} \times V_{sy} + V_{cy}$$

$$Z_{screen} = \frac{z}{w} \times V_{sz} + V_{cz}$$

V_{cx} is the center of the viewport in the coordinate system of the output device, and V_{sx} is the horizontal distance from the center to the edge of the viewport. V_{cy}, V_{sy}, V_{cz}, and V_{sz} are similarly defined.

The IRIS user declares the viewport boundaries with `viewport` and `setdepth`. The viewport centers and half-widths are computed as follows:

$$V_{cx} = \frac{left+right}{2} \qquad V_{sx} = \frac{right-left+1}{2}$$

$$V_{cy} = \frac{bottom+top}{2} \qquad V_{sy} = \frac{top-bottom+1}{2}$$

$$V_{cz} = \frac{near+far}{2} \qquad V_{sz} = \frac{far-near+1}{2}$$

Appendix C: Transformation Matrices

Here are the matrices that are created by the transformation routines.

C.1 Translation

$$Translate(T_x, T_y, T_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

C.2 Scaling and Mirroring

$$Scale(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

C.3 Rotation

$$Rot_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

C.4 Viewing Transformations

$$Polarview\ (dist, azim, inc, twist) = Rot_z(-azim) \times Rot_x(-inc) \times Rot_z(-twist) \times Trans\ (0.0,\ 0.0,\ -dist)$$

$$Lookat\ (v_x, v_y, v_z, p_x, p_y, p_z,\ twist) = Trans\ (-v_x,\ -v_y,\ -v_z) \times Rot_y(\theta) \times Rot_x(\phi) \times Rot_z(-twist)$$

where $\sin(\theta) = \dfrac{p_x - v_x}{\sqrt{(p_x - v_x)^2 + (p_z - v_z)^2}}$

$$\cos(\theta) = \frac{v_z - p_z}{\sqrt{(p_x - v_x)^2 + (p_z - v_z)^2}}$$

$$\sin(\phi) = \frac{v_y - p_y}{\sqrt{(p_x - v_x)^2 + (p_y - v_y)^2 + (p_z - v_z)^2}}$$

$$\cos(\phi) = \frac{\sqrt{(p_x - v_x)^2 + (p_z - v_z)^2}}{\sqrt{(p_x - v_x)^2 + (p_y - v_y)^2 + (p_z - v_z)^2}}$$

C.5 Perspective Transformations

$$Perspective(fov, aspect, near, far) = \begin{bmatrix} \frac{\cot\left[\frac{fov}{2}\right]}{aspect} & 0 & 0 & 0 \\ 0 & \cot\left[\frac{fov}{2}\right] & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -1 \\ 0 & 0 & -\frac{2\times far\times near}{far-near} & 0 \end{bmatrix}$$

$$Window(left, right, bottom, top, near, far) = \begin{bmatrix} \frac{2\times near}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2\times near}{top-bottom} & 0 & 0 \\ \frac{right+left}{right-left} & \frac{top+bottom}{top-bottom} & -\frac{far+near}{far-near} & -1 \\ 0 & 0 & -\frac{2\times far\times near}{far-near} & 0 \end{bmatrix}$$

C.6 Orthographic Transformations

$$Ortho_{3d}(left,right,bottom,top,near,far) = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & -\frac{2}{far-near} & 0 \\ -\frac{right+left}{right-left} & -\frac{top+bottom}{top-bottom} & -\frac{far+near}{far-near} & 1 \end{bmatrix}$$

$$Ortho_{2d}(left,right,bottom,top) = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -\frac{right+left}{right-left} & -\frac{top+bottom}{top-bottom} & 0 & 1 \end{bmatrix}$$

Appendix D: Feedback Parser

This feedback parser simplifies the use of the Geometry Engines in feedback mode. (See Chapter 10 for a discussion of the IRIS feedback utility.) The purpose of the parser is to pass to an application program the commands and data it is interested in while discarding unwanted commands.

The parser accepts a buffer of feedback data and parses it in normal or debugging mode. In normal mode the parser passes the indicated commands and data to the application. In debugging mode the parser produces a printed trace of all commands and data not passed to the application. If no commands are requested by the application then the entire feedback buffer is parsed and printed.

The interface to the parser consists of four routines. `parsefb` parses a feedback buffer. Its arguments are the name of the buffer, the size of the buffer, a flag indicating whether z-buffering is on, and a flag indicating whether depth-cueing is on. `parsefb` returns TRUE or FALSE to indicate whether the buffer was successfully parsed.

```
parsefb(buffer, count, zflag, depthflag)
short buffer[], count;
Boolean zflag, depthflag;

(not implemented in FORTRAN)

(not implemented in Pascal)
```

`setfbdebugging` turns on and off the debugging trace.

```
setfbdebugging(flag)
Boolean flag;

(not implemented in FORTRAN)

(not implemented in Pascal)
```

`bindfbfunc` associates an application-defined handling routine with a particular command.

```
bindfbfunc(command, handlingfunc)
short command;
int (*handlingfunc)();

(not implemented in FORTRAN)

(not implemented in Pascal)
```

`getfbword` is used by application-handling routines to get feedback words.

```
short getfbword()

(not implemented in FORTRAN)

(not implemented in Pascal)
```

The following program illustrates the use of the feedback parser. First, a feedback buffer is created and then the buffer is parsed.

```
#include "gl.h"

#define BUFFSIZE 400

main()
{
    char        zflag, depthflag;
    short       buffer[BUFFSIZE];
    short       count;

    zflag = getzbuffer();          /* set zflag to TRUE if
                                   z-buffering is on */
    depthflag = getdcm();          /* set depthflag to TRUE if
                                   depth-cueing is on */
    feedback(buffer, BUFFSIZE);    /* enter feedback mode */
```

```
    color(BLACK);
    clear();

    color(RED);
    circi(200, 200, 200);

    color(GREEN);
    pnt2i(200, 200);

    color(BLUE);
    recti(0, 0, 400, 400);

    count = endfeedback(buffer);  /* exit feedback mode */

    setfbdebugging(TRUE);   /* turn on debugging mode
       for parser */

    if (parsefb(buffer, count, zflag, depthflag)) /* parse
       the feedback buffer */
        printf("no errors in parse\n");
    else
        printf("error while parsing\n");

    gexit();
}
```

This parsing utility can generate data for devices such as off-line plotters. A program can use `bindfbfunc` and `getfbword` to extract primitives such as line and point drawing commands and output them to a plotter. The functions passed in the `bindfbfunc` are of the form

```
handlingroutine(count, command, string)
    int count;
    int command;
    char *string;
```

where `count` is the number of data words associated with the command, `command` is the command encountered, and `string` is a string corresponding to the command.

The following program produces a UNIX plot file from a feedback buffer:

```
#include "gl.h"
#include "gl2cmds.h"
#include "stdio.h"

#define BUFFSIZE 100
```

```
int plotpoint();
int plotmove();
int plotdraw();

main()
{
    charzflag, depthflag;
    short       buffer[BUFFSIZE];
    short       count;

    zflag = getzbuffer();
    depthflag = getdcm();
    feedback(buffer, BUFFSIZE);

    color(BLACK);
    clear();

    color(RED);
    circi(200, 200, 200);

    color(GREEN);
    pnt2i(200, 200);

    color(BLUE);
    recti(0, 0, 400, 400);

    count = endfeedback(buffer);

    bindfbfunc(FBCpoint, plotpoint);
    bindfbfunc(FBCmove, plotmove);
    bindfbfunc(FBCdraw, plotdraw);

    openpl();
    if (parsefb(buffer, count, zflag, depthflag))
        fprintf(stderr,"no errors in parse\n");
    else
        fprintf(stderr,"error while parsing\n");
    closepl();

    gexit();
}

static  int     lastx, lasty;

plotpoint(count, command, string)
    int count;
    int command;
    char*string;
{
    int i, x, y;
```

```
    x = getfbword();
    y = getfbword();
    point(x, y);
    for (i = 2; i < count; i++)
        getfbword();
}

plotmove(count, command, string)
    int count;
    int command;
    char*string;
{
    int i, x, y;

    lastx = getfbword();
    lasty = getfbword();
    for (i = 2; i < count; i++)
        getfbword();
}

plotdraw(count, command, string)
    int count;
    int command;
    char*string;
{
    int i, x, y;

    x = getfbword();
    y = getfbword();
    line(x, y, lastx, lasty);
    lastx = x;
    lasty = y;
    for (i = 2; i < count; i++)
        getfbword();
}
```

Here is the code listing for the feedback parser:

```
/*  "parse.T"  */
#include "gl.h"
#include "uc4.h"

typedef struct {
    char*fbcstring;
    int         two_d_count;
    int         three_d_count;
    int         (*handler)();
} Feedbacktable;
```

```
#include "parsetab.h"

static  short   *buff_ptr;
static  short   debugging;
static  short   buff_count;
static  short   word_no;
static  char    parse_not_done;


parsefb(buffer, count, zbuff, depthcue)
    short               *buffer;
    short               count;
    short               zbuff, depthcue;
{

    int         (*handling_routine)();
    char*FBCstring;
    short       in_3d;
    charno_error = TRUE;
    short       opcode;
    short       highbyte;
    short       passthroughcount;
    int         paramcount;

    word_no = 0;
    parse_not_done = TRUE;
    buff_count = count;
    buff_ptr = buffer;
    in_3d = zbuff || (depthcue<<1);

    while (parse_not_done && no_error) {
        opcode = 0x3f & (highbyte = getfbword());
        highbyte = highbyte >> 8;
        if ((opcode < TABLE_END) && parse_not_done) {
            switch (opcode) {
                case FBCdrawmode :
                    if (buff_ptr[1])
                        in_3d |= 1;
                    else
                        in_3d &= ( 1);
                    break;
                case FBCconfig :
                    if (*buff_ptr & UC_DEPTHCUE)
                        in_3d |= 2;
                    else
                        in_3d &= ( 2);
                    break;
                case FBCforcecompletion :
                    passthroughcount = highbyte;
                    break;
            }

            handling_routine =
```

```
                feedbacktable[opcode].handler;
            FBCstring = feedbacktable[opcode].fbcstring;
            if (in_3d)
                paramcount =
                    feedbacktable[opcode].three_d_count;
            else
                paramcount =
                    feedbacktable[opcode].two_d_count;
            if (paramcount == -1)
                paramcount = passthroughcount;
            (*handling_routine)(paramcount, opcode,
            FBCstring);
        } else
            no_error = FALSE;
    }
    if (parse_not_done)
        return FALSE;
    else
        return TRUE;
}

static char *UNDEFINED = {"ignored"};
static parsefdback(count, opcode, fbcstring)
    int         count;
    int         opcode;
    char        *fbcstring;
{
    int i;
    int inputword;

    if (debugging)
        printf("#%d: %s\n", word_no,
            (fbcstring?fbcstring:UNDEFINED));

    for (i = 0; i < count; i++) {
        inputword = getfbword();
        if (debugging)
            printf("\t\t#%d: %d:0x%x\n", word_no,
                inputword, inputword);
    }
}

getfbword()
{
    if (!(--buff_count))
        parse_not_done = FALSE;
    word_no++;
    return(*buff_ptr++);
}

setfbdebugging(flag)
{
    debugging = flag;
```

```
}

bindfbfunc(entryno, function)
    int entryno;
    int (*function)();
{
    if ((entryno >= 0) && (entryno < TABLE_END)) {
        feedbacktable[entryno].handler = function;
        return TRUE;
    } else
        return FALSE;
}
```

```
/*   "parsetab.h"   */
#include "gl2cmds.h"

static parsefdback();

/* fbcname 2d shorts, 3d shorts, function */
static Feedbacktable feedbacktable[] = {
/*0*/ {"FBCinitfbc",                0, 0,       parsefdback},
/*1*/ {0,                           0, 0,       parsefdback},
/*2*/ {"FBCmasklist",               -1,-1,      parsefdback},
/*3*/ {"GEstoremm",                 32,32,      parsefdback},
/*4*/ {"FBCrgbcolor",               3, 3,       parsefdback},
/*5*/ {"FBCrgbwrten",               3, 3,       parsefdback},
/*6*/ {"FBCsethitmode",             0, 0,       parsefdback},
/*7*/ {"FBCclearhitmode",           0, 0,       parsefdback},
/*8*/ {"FBCforcecompletion",         0, 0,      parsefdback},
/*9*/ {"FBCbaseaddress",            1, 1,       parsefdback},
/*10*/{"FBCcdcolorwe",              2, 2,       parsefdback},
/*11*/{"GEstoreviewport",           12,12,      parsefdback},
/*12*/{"GEreconfigure",             0, 0,       parsefdback},
/*13*/{"FBCdrawpixels",             -1,-1,      parsefdback},
/*14*/{"FBCreadpixels",             1, 1,       parsefdback},
/*15*/{"GEnoop",                    0, 0,       parsefdback},
/*16*/{"FBCmove",                   2, 4,       parsefdback},
/*17*/{"FBCdraw",                   2, 4,       parsefdback},
/*18*/{"FBCpoint",                  2, 4,       parsefdback},
/*19*/{"GEcurve",                   0, 0,       parsefdback},
/*20*/{"FBCcolor",                  1, 1,       parsefdback},
/*21*/{"FBCwrten",                  1, 1,       parsefdback},
/*22*/{"FBCconfig",                 2, 2,       parsefdback},
/*23*/{"FBCloadmasks",              -1,-1,      parsefdback},
/*24*/{"FBCselectrgbcursor",        9, 9,       parsefdback},
/*25*/{"FBClinewidth",              1, 1,       parsefdback},
/*26*/{"FBCcharposnabs",            3, 5,       parsefdback},
```

```
/*27*/{"FBCcharposnrel",          3, 5,       parsefdback},
/*28*/{"FBCdrawchars",            -1,-1,      parsefdback},
/*29*/{"FBCselectcursor",         7, 7,       parsefdback},
/*30*/{"FBCdrawcursor",           2, 2,       parsefdback},
/*31*/{"FBCundrawcursor",         0, 0,       parsefdback},
/*32*/{"FBClinestipple",          2, 2,       parsefdback},
/*33*/{"FBCpolystipple",          1, 1,       parsefdback},
/*34*/{"FBCsaveregs",             -1,-1,      parsefdback},
/*35*/{"FBCunsaveregs",           -1,-1,      parsefdback},
/*36*/{"FBCdepthsetup",           6, 6,       parsefdback},
/*37*/{"FBCfeedback",             0, 0,       parsefdback},
/*38*/{"FBCeof",                  1, 1,       parsefdback},
/*39*/{"FBCreadcharposn",         0, 0,       parsefdback},
/*40*/{"FBCcopyfont",             3, 3,       parsefdback},
/*41*/{"FBCpushname",             1, 1,       parsefdback},
/*42*/{"FBCloadname",             1, 1,       parsefdback},
/*43*/{"FBCpopname",              0, 0,       parsefdback},
/*44*/{"FBCfixharload",           3, 3,       parsefdback},
/*45*/{"FBCfixchardraw",          1, 1,       parsefdback},
/*46*/{"FBCinitnamestack",        0, 0,       parsefdback},
/*47*/{"FBCpixelsetup",           2, 2,       parsefdback},
/*48*/{"FBCmovepoly",             2, 4,       parsefdback},
/*49*/{"FBCdrawpoly",             2, 4,       parsefdback},
/*50*/{"FBCloadram",              -1,-1,      parsefdback},
/*51*/{"FBCclosepoly",            0, 0,       parsefdback},
/*52*/{"FBCdrawmode",             2, 2,       parsefdback},
/*53*/{"FBCsetintensity",         1, 1,       parsefdback},
/*54*/{"FBCsetbackfacing",        1, 1,       parsefdback},
/*55*/{"BPCreadbus",              -1,-1,      parsefdback},
/*56*/{"FBCxformpt",              8, 8,       parsefdback},
/*57*/{"FBCblockfill",            4, 4,       parsefdback},
/*58*/{"FBCdumpram",              -1,-1,      parsefdback},
/*59*/{"FBCzshadescanline",       10,10,      parsefdback},
/*60*/{"BPCcommand",              -1,-1,      parsefdback},
/*61*/{"FBCcopyscreen",           5, 5,       parsefdback},
/*62*/{"FBCinitscanline",          1, 1,      parsefdback}
};
#define TABLE_END 63
```

Appendix E: Window Manager Programs

This appendix describes the tools and window manager programs that are found in the */usr/people/gifts/mextools* directory. This directory contains a README file that explains how to begin using the material in it. Under the directory, two subdirectories, *tools* and *imgtools* contain all the programs in this appendix. The programs fall into six categories:

- Color Tools
- Image Tools
- General Desktop Tools
- Utilities
- Device Usage Examples
- Fantasy Demonstrations

The introduction to each category tells which subdirectory of */usr/people/gifts/mextools* contains the programs in that category.

E.1 Color Tools

In this category *loadmap* and *savemap* are contained in the *imgtools* subdirectory; the other programs are contained in *tools*.

- *cedit [number]* changes the mapping of a color index. Three sliding bars are displayed, along with a sample patch of the color being edited. A click of the left mouse button outside the screen area of *cedit* picks up a color index to be edited. This color becomes the current color, and appears in the sample patch. The sliding bars indicate components of the

current color. The color can be changed by clicking down on a sliding bar, and adjusting its position by moving the mouse.

The right mouse button brings up a menu that allows you to choose from among four color systems: rgb (red-green-blue), cmy (cyan-magenta-yellow), hsv (hue-saturation-value), and hls (hue-lightness-saturation). If you know which color system you want, you can specify *number*, an integer from 1 to 4 that corresponds to the position of the color system in the menu. The *cedit* window then appears with the color system already set.

If another process changes the color entry being edited, the *cedit* sliding bars do not indicate the correct positions for that color index.

- *gamcal* displays a checkerboard pattern. You can use this as a test pattern to check the gamma correction value by comparing the checkerboard to 50 percent gray.

- *gamma <gammavalue>* gets or sets the gamma correction value for mapping colors. The luminous intensity displayed by the monitor is a power function of the input drive voltage. When a color index is mapped using *mapcolor(index,r,g,b)*, the red, green, and blue drive voltages are specified within the range 0 to 255. However, a color with *r*, *g*, and *b* components of 128 is not 50 percent gray. On the monitor, this gray appears considerably darker because the luminous intensity is not linearly related to drive voltage.

  /usr/people/gifts/mextools/portlib provides the function *gammapcolor(i,r,g,b)* to correct for the non-linearity of the display. The gamma correction value used by *gammapcolor* and *makemap* is stored in *~/gamma*.

  gamma followed by a floating point argument sets the gamma correction value. *gamma* with no arguments prints the current gamma correction value. For a discussion of gamma correction, see *Fundamentals of Interactive Computer Graphics* by James D. Foley and Andries Van Dam (Addison-Wesley Publishing Company, 1982), pp. 594-597.

- *interp [colorsys]* interpolates a ramp between two colors. *colorsys* is a number from 1 to 4, and corresponds to the four available color systems (see *cedit*).

- *loadmap <infile>* loads a part of the color map from a file. This file must be in binary format and must have been created with *savemap*.

- *makemap* makes the default color map. Most of the *mextools* use the colors mapped by this program. *makemap* maps the lowest eight colors in the color map to the default eight colors in the Graphics Library. It then reads color mapping commands from a file in your home directory called *~/.desktop*.

  Color mapping commands contain the numbers of colors to be mapped and red, green, and blue values. If *~/.desktop* is not found, *makemap* uses default values. *makemap* then maps a gray scale, followed by an ordered color map for dithering. If there are more than eight planes, *makemap* makes a random color map from colors 64 to 127, and another ramp from 128 to 255, followed by another ordered color map. *makemap* maps the remaining colors to red.

- *pallette* displays a pallette of colors. Select a new palette by pointing with the mouse and clicking the left mouse button.

- *randmap <see> <p1> <p2>* randomizes a section of the color map. If there are less than eight bitplanes and no argument is given, *randmap* randomizes colors 16 to 31. If there are more than 8 bitplanes and no argument is given, it randomizes colors 128 to 255. *randmap* can also be given a seed and a range of colors to randomize. *seed* is an integer used to generate random colors. *p1* and *p2* specify a range of color indices to randomize.

- *savedesktop* saves color map entries 0 and 7 to 15 for use as text, border, and background colors. When you execute this command, make sure to be in your home directory. *savedesktop* then creates a file called *.desktop* in your home directory. This file stores the color assignments, which *makemap* uses to make the color map.

- *showmap* displays the color map, that is, a square filled with patches of color, one for each color index. The *showmap* menu allows you to invoke other color tools: *makemap*, *cedit*, *interp*, *palette*, and *savedesktop*. If you invoke *savedesktop*, make sure to start *showmap* from your home directory.

- *showramp* displays a ramp of colors.

- *savemap <outfile> [-r min max]* saves part or all of the color map in a file. *savemap* saves the color map range from *min* to *max* in the named file. *min* and *max* are indices into the color map. By default, *savemap* saves the first 512 colors in the color map. The colors are saved in a file that *loadmap* can later load.

E.2 Image Tools

This category includes tools that you can use to create and modify images. All programs in this category are found in the *imgtools* subdirectory, except as noted.

- *cbal <rbal gbal bbal>* sets the RGB color balance of the display. This setting affects the colors that *gammapcolor* maps. *cbal* is found in the *tools* subdirectory.
- *clip* saves a part of the screen in an independent window. Use *clip* to transiently replicate a portion of the display.
- *dither [–o] [–q] <imagein> <imageout>* converts a full color image to a 1-byte-per-pixel image, to be displayed with a color map that has 3 bits for red, 3 bits for green, and 2 bits for blue. The **–o** and **–q** flags stand for *ordered* and *quantized.*
- *hist*<infile> generates a color usage histogram of an image file.
- *ipaste <imagefile> [–m]* displays an image file on the screen, using part of the color map that *iset* sets. The **–m** flag multiplies each pixel by a factor of two.
- *iset <newtype> <imgfiles>i* sets the color map for an image. *iset* determines which part of the color map *ipaste* uses to display the image. *newtype* can be one of these: NORMAL, DITHERED, or SCREEN.
- *istat*<imgfiles> prints the header of the named image files.
- *mapimg <imagein> <imageout> <mapfile>* uses a color map to transform a screen image into an RGB image.
- *movie <image files> [–sx]* shows a series of images as a movie. The flag shows the movie in sx70 presentation format.
- *rle <infile> <outfile>* converts a verbatim, uncompacted image to an RLE (run length encoded) image.
- *showci <rgbfile>* displays an RGB image on the screen. You cannot use this program if the window manager is running.

- *shrink <shrink factor> <infile> <outfile>* uses pixel averaging to shrink the image in *infile* by a factor of *shrink factor*. *shrink factor* must be an integer. Since *shrink* uses pixel averaging, shrinking screen or dithered images can produce peculiar results.
- *snap <imageout> <width> <depth>* captures a section of the screen in an image file.
- *tobw <imagein> <bwout>* converts a color image to a black and white image by combining the bands of red, green, and blue. Red contributes 30 percent, green 59 percent, and blue 11 percent.
- *verbatim <infile> <outfile>* converts an RLE (run length encoded) image to a verbatim, uncompacted image.

E.3 General Desktop Tools

This category includes tools for creating a desktop. (You can think of the full screen presentation under the window manager as a desktop.) The programs in this category are all found in the *tools* subdirectory.

- *clock* displays an analog clock in a window.
- *fade [duration]* paints a gray background and makes windows fade away gracefully over the background, rather than just suddenly disappear. *fade* fades windows in 16 steps. The optional *duration* argument is the number of frames used to fade each of these steps.
- *fed <fontfile>* is a very simple font editor.
- *gexec <picture> <description>* executes UNIX commands from a menu. In your home directory, create two subdirectories: *.gexec* and *.images*. In *.gexec*, create a file called *description*. The following are typical lines from a *description* file.

```
texback:texback 12
mag:mag
cat picture:ipaste /usr/people/gifts/mextools/images/cat.bw
cedit:cedit
```

In *.images*, create an image file called *picture.icon*. To make *gexec* part of your desktop each time you log in, edit your *.login* file. Make sure

/usr/people/gifts/mextools/tools is in your search path, and add this line:

```
gexec picture description
```

Then, each time you log in, your desktop includes a window containing *picture.icon*. When you attach to this window, the right mouse button brings up a menu with the commands listed in *description*.

- *ical [month year]* is a simple desk calendar. Use the left and middle mouse buttons to display months forward or backward in time. *month* and *year* specify a certain calendar; the default is the calendar for the current month. See the UNIX *cal*(1) command for the syntax for *month* and *year*.
- *loadav* graphically displays the system load average with bars of color. The three bars represent the load average over the last ten, three, and one minutes. This program must be owned by root and have the *set uid* bit set to work correctly.

```
% make loadav
% su
# chown root loadav
# chmod u+s loadav
```

- *mag <factor>* magnifies the pixels of anything on the screen. Select the region to be magnified by pointing to the region and pressing the left mouse button. Pressing the middle mouse button inside the window changes the amount of magnification using the *x* position of the mouse. Pressing the middle mouse button outside the window turns the grid on and off. *factor* is an integer that specifies the initial magnification. The minimum magnification is by a factor of two.
- *texback <index>* makes a two-colored textured background. *index*, which ranges from 0 to 18, specifies different patterns for display.

E.4 Utilities

This category includes general utilities. All programs in this category are found in the *tools* subdirectory.

- *blanktime <nframes>* sets the number of frames before the screen blanks. If *nframes* equals 0, the screen never blanks.
- *ismex* returns a status of TRUE if the window manager is running and a status of FALSE if it is not running. This is useful in shell scripts or *.login* files.
- *loadfont <fontfile>* loads a font to replace the system font. This program doesn't work under the window manager.
- *mousewarp <min> <mult>* sets the mouse-warping parameters. Any raw movement of *min* pixels or more is magnified by *mult* factor. Quick motions are magnified, while slow motions move one-to-one.
- *showpie <color>* shows the division of a window's visible region into display rectangles. If you specify the *color* argument, *showpie* displays the outlines of the rectangles in that color.
- *textcolors <textcolor> <pagecolor> <highlightcolor> <cursorcolor>* sets the color indices used for the textport.
- *vis <file> [–f] [–w widthstep]* copies the bytes of a file to the screen. This works correctly only on a system with at least 12 bitplanes. The **–f** flag makes *vis* display the image as it writes to the file. The **–w** flag adjusts the width of the window when you open it. The width is a multiple of *stepwidth* pixels, which defaults to 16. Try looking at executables, libraries, and textfiles.

E.5 Device Usage Examples

This category includes sample programs that monitor devices. All the programs in this category are contained in the *tools* subdirectory.

- *keyboard* draws a keyboard that displays which keys are held down.
- *mouse* is a mouse motion tester. The program displays a mouse that moves with the movements of the real mouse. If you attach to the window and the cursor is inside the window, the mouse buttons change color when you press the real mouse buttons.

- *mousemon* monitors the mouse buttons. You can use this program to demonstrate graphically which mouse buttons a user presses as he or she uses the system.

E.6 Fantasy Demonstrations

This category includes demonstration programs. *imged* and *melt* are contained in the *imgtools* subdirectory; the rest of the programs are contained in *tools*.

- *curved* is a minimal object space curve editor.
- *paint* is a minimal object space paint program. Choose a color to use for painting by clicking the left mouse button on a color outside the window. Clear the window with the middle mouse button. This program accepts input from the digitizer tablet as well as the mouse.
- *readimage <imagefile>* reads an image file but doesn't do anything with it. This program is a sample that shows how image files are read.
- *scribble* puts the IRIS in full screen mode, and draws anywhere on the screen, even under the window manager.
- *spiral <angle> <growth>* draws a spiral pattern using a version of turtle graphics. Try

  ```
  spiral 121 0.3
  ```

  as a first example.
- *stars* creates a random field of stars in 3-D space and moves them around with the mouse.
- *sunflower <nseeds> <seedsize> <growth>* makes a sunflower-like pattern out of circles. Try

  ```
  sunflower 40 0.05 1.1
  ```

  as a first example.
- *worms [–field] [–length #] [–number #] [–trail]* is the UNIX worms program running on the IRIS.

- *zoing* makes a spiral out of circles.
- *imged <infile> [xsize ysize]* is a minimal image editor.
- *melt [size]* makes the part of the screen under its window appear to melt. To restore this part of the screen, move or remove the *melt* window. *size* indicates the size in pixels of the melting chunks; the default size is one pixel.

Appendix F: IRIS Programming Tutorial Manual Pages

This appendix contains the manual pages for the Graphics Lab of the *IRIS Programming Tutorial.*

NAME

backface – shows the difference between an object displayed with and without backface removal.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/backface

DESCRIPTION

backface contrasts two views of a solid, convex object (a cube). The *backface* screen has four different windows: an information window, two view windows, and an example window that illustrates the principle of backface removal. The left view window shows the object without backface removal. The right view window shows the same object with backface removal.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

You use the first three menu entries to animate the cube, turn on and off the arrows that represent surface normals, and turn on and off the shading for the object. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Information Window

The information window is in the upper left portion of the screen. The text in the information window changes throughout the program to give you different instructions.

View Windows

The lower left window shows the object without backface removal. The lower right window shows the same object with backface removal.

Example Window

The example window in the upper right corner of the screen shows how the IRIS decides whether to display polygon. When the surface normal of a polygon faces away from the eye position, the IRIS doesn't draw the polygon. The IRIS determines the surface normal by examining the order in which you specify the vertices of a polygon. If you specify the vertices in a counter-clockwise direction, the IRIS knows when to draw the polygon. If you specify them in a clockwise direction, the IRIS doesn't know when to draw it.

WINDOW MANAGER

This program runs only in the window manager. It uses the first ten colors in the color map, so any other programs using those colors affect the display on this program or may be affected by this program.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

AUTHOR

Michael Clark

ORIGIN

Silicon Graphics, Inc.

NAME

buffer – contrasts single and double buffering in an animated scene.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/buffer

DESCRIPTION

buffer shows an animated scene that you contrasts the two display modes.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

Select "single buffer animation" to view the scene in single buffered mode. Select "double buffer animation" to view it in double buffered mode. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

The single buffered animation will flicker a bit because you are watching the IRIS draw and erase graphics from the screen. The double buffered animation is very smooth.

WINDOW MANAGER

This program runs only in the window manager.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

AUTHORS

Vince Uttley and Mason Woo

ORIGIN

Silicon Graphics, Inc.

NAME

curve – shows the difference between three types of curves.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/curve

DESCRIPTION

curve shows how the curve routines in the IRIS Graphics Library work. Use the left mouse button to select the precision with which a curve is drawn, to control the slider bars, and to select control points. Use the right mouse button to make selections from the pop-up menu.

You have several options for action. Use the left mouse button to select the curveprecision routine in the status window. Then a control bar appears in control bar window. Move the cursor over the small rectangle on the right edge of the slider bar, press and hold the left mouse button, and, as you move the mouse to the left, the slider bar moves with it. When you feel comfortable with this, select a point labeled 1,2,3 or 4 in the z-axis view window. Three slider bars appear in the control bar window. They control the x, y and z values for the selected point. As you change the values of the points you see how they affect the shape of the curve. Experiment with different basis matrices by selecting "change basis" from the menu.

Display Windows

The display has several windows. The information window describes how to use the program. The control bar window contains the bars. The status window shows the current state of the program, that is it displays the curvebasis and curveprecision routines that you select. The other windows are views of the curve. Note that the control points in the curve picture are labeled 1,2,3 and 4. These are not the values of the points but rather their positions in the crv(g) routine.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

BUGS

You can move the control points out of view so that you can not select them again. Select "reset" from the pop-up menu to move the control points back to their initial position.

AUTHOR

Michael Clark

ORIGIN

Silicon Graphics, Inc.

NAME

depthcue – demonstrates how to create the illusion of depth.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/depthcue

DESCRIPTION

depthcue shows how the IRIS Graphics Library *perspective*, *setdepth*, and *shaderange* routines give a wireframe model depth. Use the left mouse button to control the slider bars that change the value of parameters to the routines. Use the middle mouse button to rotate the object, the letter 'F.' Use the right mouse button to display a pop-up menu from which you select routines.

Display Windows

buffer has six windows. The information window displays instructions for using the program. The control bar window contains the control bar you use to edit parameter values. The status window displays the syntax of the selected Graphics Library routine. The ramp window displays the colors you can use. The side view window displays the viewing volume, the object, and the ramp of colors that span the clipping region. The main view window displays the graphic image generated by the routines you selected.

The object is a 3D letter 'F'. The coordinate axes in the object coordinate system are attached to the letter. The IRIS uses the viewing volume displayed in the side view window to generate the image that you see in the main view window.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

Select the first entry, "reset values", to reset all parameters to their default values. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager. It uses the first ten colors in the color map, so any other programs using those colors affect the display on this program, or may be affected by this program.

HARDWARE REQUIREMENTS

Your IRIS needs 16 bitplanes of image memory to run this program, and the Z-clipping hardware to perform the near and far clipping.

BUGS

Since the parameters are 32-bit floating point numbers, or 16-bit integers, you can reach a maximum and minimum value. If you exceed these values, the program has undefined results. If the near or far clipping values are set to negative numbers, incorrect images will be displayed. Also, using the left mouse button to alter values and simultaneously using the right mouse button to work the pop-up menu causes unpredictable results.

AUTHOR

Thant Tessman

ORIGIN

Silicon Graphics, Inc.

NAME

gamma – demonstrates gamma correction using the color map.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/gamma

DESCRIPTION

gamma shows the effects of gamma correcting a color ramp using the IRIS Graphics Library *mapcolor* routine. Use the left mouse button to control the slider bars that change the value of the gamma constant.

Display Windows

gamma has three windows. The information window contains instructions for using the mouse buttons. The control window contains the control bar for editing gamma values. The graph window contains a curve that shows the intensity ramp generated by the given gamma. There are also two color ramps. One is gamma corrected, and the other is linear.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

When you select the first item, "reset gamma to 1.0", the IRIS resets the gamma value to 1.0. This produces a linear color ramp. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

To change a slider bar value, move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value. In this program, the slider bar changes the gamma value. The top color ramp changes to reflect the new value of gamma. If you move the cursor off the ends of the slider bar, the cursor automatically jumps to the other end of the bar, the values at the ends of the bar change to mark the new range, and you can continue moving the cursor in the desired direction.

Window Manager

This program runs only in the window manager. It uses the first 48 colors in the color map so any other programs using those colors affect the display on this program or may be affected by this program when running in the window manager.

HARDWARE REQUIREMENTS

Your IRIS needs 16 bitplanes to run this program.

BUGS

Since the parameters are 32-bit floating point numbers, you can obtain a maximum and minimum value. If you exceed these values, the program has undefined results. Also, using the left mouse button to alter values and simultaneously using the right mouse button to work the pop-up menu causes unpredictable results.

AUTHOR

Mason Woo

ORIGIN

Silicon Graphics, Inc.

NAME

gouraud – demonstrates Gouraud shading concepts.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/gouraud

DESCRIPTION

gouraud simulates a shaded polygon with large pixels. You can edit both the position and intensity of each vertex of a four sided polygon. After each edit, the IRIS recalculates the color intensities of the pixels in the polygon.

Display Windows

gouraud consists of three individual windows. The console window contains 32 color indices and a slider which allows you to change the color index of the current vertex. The grid window contains a large red grid and a Gouraud shaded polygon. The current (selected) vertex is highlighted in red, and the others are highlighted in yellow.

The help window tells you which options are available to you at any time.

Program Operation

Except to exit, you run the program entirely with the left mouse button.

You always edit the current (selected) vertex. To select a vertex, position the cursor over one of the four vertices of the polygon and press the left mouse button. The vertex turns red, and the slider in the console window positions itself over the color index of that vertex.

To change the color index of the vertex, position the cursor over the slider and hold down the left mouse button. You can now drag the slider with the cursor. The number inside the slider changes to tell you which color index the current vertex is using, and this vertex changes color accordingly. When you release the left mouse button, the IRIS redraws the polygon with the new color index.

To move a vertex of the polygon, position the cursor over the vertex you want to move, press and hold the left mouse button, and move the cursor to a new position. When you move the mouse, the polygon becomes a wire frame model and changes according to your mouse movements. When you release the left mouse button, the IRIS redraws the polygon in its new shape.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

The only item in the pop-up menu is "exit program". When you select it, a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to exit from the program; select "no" to return control of the program to the original pop-up menu.

Window Manager

This program runs only in the window manager. It uses colors 31 through 63 of the color map, so any other programs using those colors affect the display on this program, or may be affected by this program.

HARDWARE REQUIREMENTS

Your IRIS needs 16 bitplanes of image memory to run this program.

AUTHOR

Michael Clark

ORIGIN

Silicon Graphics, Inc.

NAME

patch – demonstrates bi-cubic patches

SYNOPSIS

/usr/people/tutorial/c.graphics/online/patch

DESCRIPTION

patch demonstrates how to use the patch routines in the IRIS Graphics Library. *patch* is very similar to the *curve(1t)* program, and you should run it only after you have a fair understanding of *curve*.

You have several options for action. You can use the left mouse button to select the patchcurves routine in the status window. Then a control bar appears in the control bar window. Change the parameters to the patchcurve routine using the slider bar (explained below). Use the slider bars to change parameters to the patchprecision routine. When you feel comfortable with this, select one of the points in the z-axis view window. Now three slider bars appear. They control the x, y and z values for the selected point. As you change the values of the points you see how they affect the shape of the patch. Experiment with different basis matrices by selecting "change basis" from the menu described below.

Display Windows

patch has six windows. The information window describes how to use this program. The control bar window contains the control or slider bars. The status window displays the patchbasis, patchcurves, and patchprecision routines that you select. The other three windows are views of the patch. Note that the control points in the curve picture are labeled sequentially. These are not the values of the points; they are their positions in the patch(g) routine.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

BUGS

You can move the control points out of view so that you can not select them again. Select "reset" from the pop-up menu to move the control points back to their initial position.

ORIGIN

Silicon Graphics, Inc.

NAME

projection – illustrates projection and viewing routines.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/projection

DESCRIPTION

projection shows how the IRIS Graphics Library projection transformations (*perspective*, *window*, and *ortho*) and viewing transformations (*polarview* and *lookat*) affect the way the IRIS displays a 3D object. Use the left mouse button to control the slider bars that change the value of parameters passed to the routines. Use the right mouse button to display a pop-up menu from which you can select routines.

Display Windows

The display has six windows. The information window contains instructions for using the mouse buttons. The control bar window contains bars that you use to change parameter values. The status window shows the syntax of the Graphics Library routine it is using to display the model. The viewport window displays the graphics generated by the routines. The two view windows show the viewing volume and object as seen from different fixed eye positions; one eye is fixed on the X axis in the object coordinate system, and the other is fixed on the Z axis.

The object is a 3D letter 'F'. The coordinate axes are attached to the letter in the object coordinate system. The IRIS uses the viewing volumes in the two view windows to generate the image in the viewport window.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

The first two menu items have sub-menus. "projection transformation" has a sub-menu that contains "ortho", "window", and "perspective". When you select one of these routines, the IRIS displays the Graphics Library C code and uses that code to project graphics on to the screen. "viewing transformation" has a sub-menu that contains "polarview" and "lookat". This sub-menu works the same way as the sub-menu to "projection transformation". The next item, "reset values", resets all parameters to their default values.

When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager. It uses the first ten colors in the color map, so any other programs using those colors affect the display of this program, or may be affected by this program.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program. It also needs the Z-clipping hardware to perform near and far clipping.

BUGS

Since the parameters are 32-bit, floating point numbers, or 16-bit integers, you can reach a maximum and minimum value. If you exceed these values, the program has undefined results. If you set the near or far clipping values to negative numbers, the IRIS displays incorrect images. Also, using the left mouse button to alter values and simultaneously using the right mouse button to work the pop-up menu causes unpredictable results.

AUTHOR

Thant Tessman

ORIGIN

Silicon Graphics, Inc.

NAME

queue – demonstrates how the event queue processes entries.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/queue

DESCRIPTION

queue simulates the IRIS event queue. You can queue input buttons on the keyboard and mouse, and experiment with the *qdevice*, *qtest*, *qread* and *qreset* Graphics Library routines. The user can queue input buttons on the keyboard and mouse.

Display Windows

queue has four windows. The information window gives you instructions on how to use the program. The menu window contains three menus. Use the top menu to queue and unqueue several input devices, the middle menu to tie and untie the RIGHT MOUSE button to the MOUSEX and MOUSEY valuators, and the bottom menu to select a series of actions that represent Graphics Library routines which query and read the status of the queue. Use the left mouse button to select menu items. The queue window contains the event queue. The variables window displays the values of three variables. You change the variables when you select them from the bottom menu.

WINDOW MANAGER

queue runs only in the window manager.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

AUTHOR

Thant Tessman

ORIGIN

Silicon Graphics, Inc.

NAME

scrmask – illustrates using the scrmask routine with text strings.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/scrmask

DESCRIPTION

scrmask shows how the IRIS Graphics Library *viewport* and *scrmask* routines affect text strings. Use the left mouse button to control the slider bars that change the value of parameters to the routines. Use the right mouse button to display a pop-up menu from which you select routines.

Display Windows

scrmask has four windows. The information window displays instructions for using the program. The control bar window contains the control bar you use to edit parameter values. The status window displays the syntax of the Graphics Library that you selected. The view window displays the graphics generated by the routines you select.

The graphics are character strings that state the move routines which positioned them. There are also two boxes: the green represents the viewport, and the red represents the screenmask.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

The first item toggles between two choices. If you are using the default screenmask, the first item is "choose screenmask values". Select this to change the screenmask values. If you have chosen new values, the first item is "use default screenmask". Select this to use the viewport values as the screenmask values. "reset values" resets all parameters to their default values. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager. It uses the first ten colors in the color map, so any other programs using those colors affect the

display on this program or may be affected by this program.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

BUGS

Since the parameters are 16-bit integers, you can reach a maximum and minimum value. If you exceed these values, the program has undefined results. Also, using the left mouse button to alter values and simultaneously using the right mouse button to work the pop-up menu causes unpredictable results.

AUTHOR

Mason Woo

ORIGIN

Silicon Graphics, Inc.

NAME

shape – demonstrates 2D drawing routines.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/shape

DESCRIPTION

shape shows the geometric shapes that you can display on the screen with these IRIS Graphics Library drawing routines: *recti*, *rectfi*, *circi*, *circfi*, *arc*, *arcfi*, *pnt2i*, *move2i*, *draw2i*, *pmv2i*, *pdr2i*, and *pclos*.

Use the left mouse button to control the slider bars that change the value of parameters to the routines. Use the right mouse button to displays a pop-up menu from which you select routines.

Display Windows

The display has four different windows. The information window displays instructions for using the program. The control bar window contains the control bar you use to edit parameter values. The status window displays the syntax of the selected Graphics Library routine. The graph window displays the graphics generated by the routines you select.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

The first nine menu items are drawing routines. When you select one the IRIS displays the Graphics Library C code for the routine. When you select the next item, "reset values", it resets all parameters to their default values. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager. It uses the first ten colors in the color map, so any other programs using those colors affect the display on this program, or may be affected by this program.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

BUGS

Since the parameters are 16-bit integers, you can reach a maximum and minimum value. If you exceed these values, the program has undefined results. Also, using the left mouse button to alter values and simultaneously using the right mouse button to work the pop-up menu causes unpredictable results.

AUTHOR

Mason Woo

ORIGIN

Silicon Graphics, Inc.

NAME

swap – demonstrates swapping buffers to display smooth animation.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/swap

DESCRIPTION

swap simulates drawing an object to both front and back buffers, and also swapping those buffers. You can experiment with the *frontbuffer, backbuffer* and *swapbuffers* Graphics Library routines. There are four main sections of the window: a help area, an onscreen menu, a representation of the front buffer, and the contents of both front and back buffers.

Display Windows

swap has four different windows. The information window displays instructions for using this program. The console window contains three menus. Use the top menu to clear or draw to all enabled buffers, to swap the contents of the front and back buffers, and to exit from the program. Use the middle menu to enable and disable the front buffer for drawing (by default, the front buffer is disabled). Use the third menu to enable and disable the back buffer (by default, drawing takes place in the back buffer). Always use the left mouse button to select items from these menus. The screen view window is actually an enlargement of the contents of the front buffer, that is, the buffer that contains the image that you see. The buffers window shows the contents of both front and back buffers. When you enable a buffer, it is highlighted with a contrasting color.

WINDOW MANAGER

swap runs only in the window manager.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

AUTHOR

Mason Woo

ORIGIN

Silicon Graphics, Inc.

NAME

transform – illustrates modeling/object transformation routines in real-time.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/transform

DESCRIPTION

transform shows how the IRIS Graphics Library *rotate*, *translate*, and *scale* commands affect a 3D object. Use the left mouse button to control the slider bars that change the value of parameters to the routines. Use the right mouse button to display a pop-up menu from which you select routines.

Display Windows

transform has six windows. The information window contains instructions for using the program. The control bar window contains the control bar you use to edit parameter values. The status window displays the syntax of the selected Graphics Library routine. The viewport window contains the graphics generated by the selected routine. In this window, the eye position is at a point on the positive Z axis in the world coordinate system. The two view windows show the same graphics with the eye positioned on the Y axis and on the X axis.

The object is a 3D letter 'F'. The coordinate axes in the object coordinate system are attached to the letter. *translate*, *rotate* or *scale* affect these object coordinate system axes.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

The first three menu items are "rotate", "translate", and "scale". When you select one of these routines the IRIS displays the Graphics Library C code for the routine. (When you select "rotate" the IRIS displays three lines of code.) When you select the next item, "reset values", it resets all parameters to their default values. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager. It uses the first ten colors in the color map, so any other programs using those colors affect the display on this program or may be affected by this program.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program.

BUGS

Since the parameters are 32-bit, floating point numbers, or 16-bit integers, you can reach a maximum and minimum value. If you exceed these values, the program has undefined results. Also, using the left mouse button to alter values and simultaneously using the right mouse button to work the pop-up menu causes unpredictable results.

AUTHOR

Mason Woo

ORIGIN

Silicon Graphics, Inc.

NAME

viewport – illustrates how the IRIS displays a 3D world on a 2D screen (viewport).

SYNOPSIS

/usr/people/tutorial/c.graphics/online/viewport

DESCRIPTION

viewport shows how the IRIS Graphics Library *viewport* and *ortho* routines affect the images you see on the screen. Use the left mouse button to control the slider bars that change the value of parameters to the routines. Use the right mouse button to display a pop-up menu from which you select routines.

Display Windows

viewport has five windows. The information window displays instructions for using the program. The control bar window contains the control bar you use to edit parameter values. The status window displays the syntax of the selected Graphics Library routines. The screen space window displays the graphics generated by the selected routines. The world space window displays the 3D viewing volume and the viewport screen generated by the selected routines.

The graphics are several F's and some dots. Attached to each dot is a character string showing the 3D position of the dot.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

When you select the first entry, "reset values", the IRIS resets all parameters to their default values. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager. It uses the first ten colors in the color map, so any other programs using those colors affect the display on this program, or may be affected by this program.

HARDWARE REQUIREMENTS

Your IRIS needs 12 bitplanes of image memory to run this program, and Z-clipping hardware to use the near and far clipping.

BUGS

Since the parameters are 16-bit integers, you can reach a maximum and minimum value. If you exceed these values, the program has undefined results. Also, using the left mouse button to alter values and simultaneously using the right mouse button to work the pop-up menu causes unpredictable results.

AUTHOR

Mason Woo

ORIGIN

Silicon Graphics, Inc.

NAME

writemask – demonstrates how the writemask routine affects colors.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/writemask

DESCRIPTION

writemask lets you experiment with the *writemask* Graphics Library routine. You select a color and writemask, then use the mouse to draw streaks of color on the screen. You can also select different color maps to achieve different effects.

The Display

writemask has three windows. The information window displays instructions for using the program. The console window contains a menu, some binary digits that determine the color, writemask, and number of bitplanes to which to write new values, and a palette of colors. The paint area window is the area in which you can draw bars or streaks of color. Use the left mouse button to select menu items, toggle the binary numbers, or draw in the paint area.

What is a writemask?

Each pixel on your screen contains a color index number. This number tells the IRIS where to look in the color map to find out which color to display for that pixel. When you draw a line on the screen, you change the values of the color indices for the pixels along the line.

The IRIS stores these indices in binary form in its bitplanes, and the writemask protects the values stored in the bitplanes. The default writemask lets the IRIS write to all bitplanes, that is, all bitplanes are *enabled.* A bitplane with a writemask of 1 is enabled; a bitplane with a writemask of 0 is protected.

Selecting a color

You can select a drawing color in two ways: change the binary digits of the color index, or select a color from the palette. The binary digits you can change are inside three yellow squares next to the word *color* in the console window. You can toggle the value of a bit by selecting it with the left mouse button. For example, if the value of the bit is zero when you select it, the value changes to one. The IRIS displays the new value to the left of the bit. The IRIS also displays the decimal value and binary digits of the colors in the palette.

Using the Menu

Select "clear drawing area" to completely erase everything in the drawing area. Select "clear using writemask" to completely erase (zero out) only the enabled bitplanes.

Select "additive color map" to load colors 0 through 7 into the color map. The color map is called *additive* because each bitplane adds a different color value to the screen. With this color map, the 1's bit (the rightmost bit) 'adds' red. The 2's bit (middle bit) 'adds' green. The 4's (left bit) 'adds' blue. Therefore color number 6 is 4 + 2, or blue + green, which is cyan. Experiment with colors and writemasks with this color map as a small paint system. Paint the color blue (index 4). Then set the writemask to 2 or 3, which freezes the 'blue' bitplane. Then draw color 2 on top of color 4. Since the 4's bitplane is frozen, the new color 2 combines with the color 4 to create color 6.

Select "overlay color map" to load colors 0 through 7 into another type of color map. This map is called an *overlay* map because it makes the 4's bitplane (the left bit) an overlay bitplane. When you set the writemask to 3, the IRIS can draw only colors 0 to 3. Think of these colors as the colors of your underlying drawing. When you set the writemask to 4, the IRIS can change only the 4's bit in the drawing. This bitplane is an overlay on top of your drawing. You can draw and erase your overlay without affecting the underlying drawing.

Select "color bars" to draw a test pattern of colors 1 through 7. This saves you from doing a lot of drawing.

WINDOW MANAGER

Writemask runs only in the window manager.

HARDWARE REQUIREMENTS

Your IRIS needs 20 bitplanes of image memory to run this program.

AUTHOR

Thant Tessman

ORIGIN

Silicon Graphics, Inc.

NAME

zbuffer – illustrates how zbuffering works.

SYNOPSIS

/usr/people/tutorial/c.graphics/online/zbuffer

DESCRIPTION

zbuffer shows how the IRIS Graphics Library *zbuffer* routine affects 3D objects. Use the left mouse button to control the slider bars that change the value of parameters to the routines. Use the right mouse button to display a pop-up menu from which you select routines.

Display Windows

zbuffer has six windows. The information window displays instructions for using the program. The console window contains the control bar you use to edit parameter values to the rotate routines. The status window displays the syntax of the selected Graphics Library routine that has rotated the object. The main view window displays the graphics generated by the routines you selected. In this window, the eye position is at a point on the positive Z axis in the world coordinate system. The two view windows display the same graphics with the eye position on the Y axis and on the X axis.

The object is a 3D letter 'F'. When you draw it as a wireframe figure, the coordinate axes are attached to the letter. All rotation occurs around the object coordinate system axes.

Pop-up Menu Operation

Press the right mouse button to display a pop-up menu. To select an item from the menu, move the cursor until the item is highlighted, and then release the button. To eliminate the menu without selecting an item, move the cursor out of the menu and release the button.

Select the first item on the menu, "z-buffer filled object", to draw the object as a solid with hidden surfaces removed using the z-buffer. Select "filled object without z-buffer" to draw the object as a solid with no hidden surface removal at all. Select "reset values" to reset all parameters to their default values. When you select the last item, "exit program", a second pop-up menu appears which asks you to confirm this selection. Select "yes" from this menu to terminate the program; select "no" to return control of the program to the original pop-up menu.

Slider Bar Operation

Use the left mouse button to select the parameter that you want to change. Move the cursor over the bar, press and hold the left mouse button, and move the cursor back an forth to change the value of the selected parameter.

Window Manager

This program runs only in the window manager. It uses the first ten colors in the color map, so any other programs using those colors affect the display on this program, or may be affected by this program. The z-buffering works only while there are no programs running in double buffer mode in other windows.

HARDWARE REQUIREMENTS

Your IRIS needs 32 bitplanes of image memory and z-clipping to run this program.

BUGS

Since the parameters of rotate are 16-bit integers, you can reach a maximum and minimum value. If you exceed these values, the program has undefined results. Also, using the left mouse button to alter values and simultaneously using the right mouse button to work the pop-up menu causes unpredictable results.

AUTHOR

Mason Woo

ORIGIN

Silicon Graphics, Inc.

Appendix G: Fast Immediate Mode and User-Defined Display Lists

G.1 Introduction and Overview

The use of the fast immediate mode macros can significantly speed up the execution of graphics code. This paper presents the advantages and disadvantages of fast immediate mode macros, gives strategies for their use, and provides examples. This section describes their general use, and in particular their use in connection with user-defined display lists.

This material applies only to 2000 and 3000 series workstations. Silicon Graphics does not guarantee support for these macros in their current form for future products.

What are the fast immediate mode macros?

The fast immediate mode macros are contained in four files in */usr/include/gl2*: *imsetup.h*, *imattrib.h*, *imdraw.h*, and *immatrix.h*. These files send graphics tokens and values directly to the Geometry Pipeline. You can include all four files with */usr/include/gl2/immed.h*.

Under Graphics Library 2 (GL2) there are three modes for accessing the Graphics Library: *immediate mode*, *display list mode*, and *fast immediate mode*. For example, the ordinary immediate mode version of a **draw()** command is simply a function call to the Graphics Library routine **draw()** which in turn sends the draw token to the Geometry Pipeline followed by the *x*, *y*, and *z* values. This subroutine contains one of the fast immediate mode macros, **im_draw()**, which actually moves these tokens and values to

the pipe. **Almost every ordinary immediate mode routine contains one or more of the fast immediate mode macros. Im_draw** roughly corresponds to:

```
*GE =  draw
*GE = x;
*GE = y;
*GE = z;
```

where GE is the address of the pipe. The token is 0x211.

With the display list version, a call to **draw()** between **makeobj()** and **closeobj()** places the address of the **draw** display list interpreter routine in display list memory followed by its arguments (*x*, *y*, and *z*). Subsequent calls to the object via **callobj()** traverse the display list, sending the same tokens to the pipe as in the immediate mode version. Fast immediate mode uses in-line macros that have the same effect as immediate and display list mode, but do not incur the subroutine overhead of immediate mode and and do not build a display list for subsequent traversal. The result in all three cases is the same–the same tokens and values are sent to the pipe. Fast immediate mode can be faster than ordinary immediate mode, while display list traversal is a close second.

When to use the fast immediate macros?

- **To increase speed**

The fast immediate mode macros can increase speed. Using them in inner loops can save the repeated overhead of subroutine calls. For example, they can increase speed when drawing a great number of vectors or small polygons. The type of system you have, 68010- or 68020-based, affects the amount of speed you can gain. The tradeoffs involving speed are discussed below.

- **To create your own display lists**

You can use the fast immediate mode macros to create customized display lists. These display lists can increase flexibility and speed in editing, as well as provide other benefits. The tradeoffs involving user-defined display lists are discussed below.

When not to use the fast immediate macros?

- **If performance is not critical**

Using the fast immediate mode macros is extra work, so if speed is not important, using them may not be worth your while. For example, using fast immediate mode for menus and backgrounds does not save a significant amount of time because backgrounds require much time to draw and menus have to react only at human speeds.

- **If you are not using C**

The fast immediate mode macros are written in C and do not work in FORTRAN or Pascal. You can call C routines from FORTRAN or Pascal to do fast drawing. This is only advantageous when you can concentrate several calls to the fast immediate macros in a single C routine so you do not repeatedly incur subroutine and wrapper overhead.

- **If the Graphics Library display lists are more convenient**

The Graphics Library display lists can be convenient and flexible, and in certain instances it may not be worth the time to code your own display lists. If you are porting GL1 code to GL2, it can also be more convenient to retain the Graphics Library notion of objects.

- **If you aren't editing objects**

Although other factors are involved, a general rule of thumb is that if you never edit your objects, the Graphics Library display lists are sufficient. In terms of traversal speed, you cannot do much better than the Graphics Library display lists, but if you are doing a lot of editing, you can probably speed up your application by bypassing the Graphics Library.

- **If fast immediate mode isn't available for all routines**

In general, the simple routines (**move(), draw(), poly(), rect(), pnt(), color(), writemask(), translate(), rotate(), scale()**) have fast immediate mode versions. But when subroutine overhead is small compared to execution time, there is generally no fast immediate mode equivalent. For example, there are no fast immediate mode equivalents for **clear(), circle(), arc(), ortho(), polarview(),** and **lookat()**. To see which routines have fast immediate mode versions, see the file */usr/include/gl2/immed.h*.

- **If you are using z-buffer mode**

Z-buffering is slow. Consequently, it is just as fast to call a subroutine. Fast immediate mode does not significantly increase the speed when compared to drawing time.

How do you use the fast immediate macros?

Here is a simple fast immediate mode example. It does not illustrate a necessarily *advantageous* use of fast immediate mode because the registers are set up for only one routine. A more realistic example appears later.

```
#include <gl2/immed.h>
#include <gl.h>

main()
{
        ginit();
        fastrect();
        gexit();
}

fastrect()
{
        im_setup;    /* to assure address
                        register assignment */

        color(BLACK);
        clear();
        color(RED);
        im_rects(100, 100, 300, 300);

                /* immediate mode version of
                        "rects" command */
                        /* actually draws the
                            /rectangle*/

}
```

Even this simple example must call a subroutine. This is because **ginit()** must be called before any other graphics routines, and **im_setup** must appear first in every routine that uses fast immediate mode. In fact, **ginit** and **im_setup** cannot appear in the same function due to conflicts in setting up the registers. **im_setup** initializes two address registers in the 680x0, leaving two for the user. The first register gets the address of the Geometry Pipeline. The other register gets the address of an attribute vector. Check assembler listings to make sure address registers have been assigned by the

compiler. If not, the program will work, but all the fast immediate mode performance advantage may be lost.

Fast immediate mode macros are accessible through the file *fastimmed.h* (3000 series) or *immed.h* (2000 series) listed below. These files often change from release to release. All the include files are in */usr/include/gl2*.

3000 series -- *fastimmed.h*

```
#define UNIX
#define IP2
#define DC4
#define UC4
#include "gl2/gltypes.h"
#include "gl2/globals.h"
#include "gl2/immed.h"
#include "gl2/imsetup.h"
```

Here *immed.h* contains:

```
#ifndef  IMMEDDEF
#define  IMMEDDEF

#ifndef KERNEL
#include"globals.h"
#endif
#include"imsetup.h"
#include"imattrib.h"
#include"imdraw.h"
#include"immatrix.h"

#endif  IMMEDDEF
```

Judging from the redundancy in the contents of the above files, it is better to include *immed.h* on 3000 series machine and add the appropriate defines and includes which *immed.h* does not have. You should also include *gltypes.h* (see examples below).

2000 series -- *immed.h*

```
#ifndef  IMMEDDEF
#define  IMMEDDEF

#ifndef KERNEL
#include"globals.h"

#endif
#include"imsetup.h"
```

```
#include"imattrib.h"
#include"imdraw.h"
#include"immatrix.h"
#endif  IMMEDDEF
```

The 2000 series include file is less comprehensive. It may also be necessary to include the following when using the fast immediate mode macros on 2000 series machines:

```
#define UNIX
#define PM2
#define DC4
#define UC4

#include <gl2/gltypes.h>
#include <gl2/glerror.h>
```

Macros work differently from subroutines in C. The expressions are evaluated in order from left to right. Thus, if *array[]* is a list of short integer coordinates ordered *x*, *y*, *z*:

```
im_setup;
register int *ptr = &array[0];

while(ptr<3000)
        im_pnts = (*ptr++, *ptr++, *ptr++);
```

draws 1000 points correctly in fast immediate mode.

It is worthwhile to familiarize yourself with a fast immediate mode macro. The macros themselves are their best documentation, although they may seem unfamiliar at first. Since the macros are often nested, it is necessary to preprocess them to see what they will look like. The **-E** flag to **cc(1)** is useful for preprocessing. The **-E** flag scrolls the preprocessed file to standard output. The following two lines:

```
#include <gl2/immed.h>
im_draw(a,b,c);
```

produce:

```
{;*(short *)GE = (0x11 |(0x200 | 0x000));
*(float *)GE = (a);
*(float *)GE = (b);
*(float *)(GE-0x800) = (c);};
```

There are a few things worth noting here:

- The first line moves the token (0x211) to **GE** (the address of the Geometry Pipeline). Moving this first token to the pipe locks the pipe until the last argument is written. This is necessary because the graphics application and the system's text window share the graphics subsystem. Moving part of the graphics instruction into the pipe, breaking away, and starting a new instruction can be disastrous. The locking mechanism prevents this from happening. If you step through a graphics program using **adb**, you will notice this. You will see the instruction that moves the first token of a graphic primitive (e.g., **draw**) into the pipeline, but the screen remains blank until the last argument of the primitive has been moved to the pipe. Then the other **move** instructions will suddenly appear.
- The values of *a* and *b* are moved into **GE**
- The value of *c* is moved into **GE** offset by 0x800 (4096). This is the same physical address as **GE**, but it is wired so that the pipe unlocks when it is written to.
- Because **im_draw** was preprocessed above, one instruction didn't show up in the expanded version. **GEWAIT** is part of the definition of **im_draw** (see */usr/include/gl2/imdraw.h*). **GEWAIT** is used internally at Silicon Graphics and can be ignored. It is mentioned here only to avoid confusion.

G.2 User-Defined Display Lists

When should you create your own display list using the macros?

There are numerous factors that you must take into account if you use the immediate mode macros to build your own display list. This section addresses the question of when and if it's appropriate to use them, and when to define your own display list.

Display lists were necessary in Graphics Library 1. Since the conversion from IRIS floating point to Geometry Engine floating point was done in software, much was gained (by a factor of 25) by doing the conversion in

advance and placing the results in the display list. Then, when the display list was traversed, the values in the display list could be sent down the pipe as is.

The inclusion of the Geometry Accelerators on Graphics Library 2 machines to do the conversion rendered the display list comparatively unnecessary, and the inclusion of object handling routines in Graphics Library 2 was largely for compatibility with Graphics Library 1. With a 68010-based Graphics Library 2 machine, plain immediate mode is roughly one quarter to one third as fast as display list mode due to function call overhead. You can eliminate this overhead with the use of the fast immediate mode macros. On a 68020-based system, even the function call overhead is usually insignificant, since the ability of the Geometry Engines to consume data dominates performance, rather than the ability of the processor to feed them data.

Although the Graphics Library display lists are easier to use and Geometry Engine's speed in traversing existing display lists is very good, there are a number of reasons to bypass the Graphics Library's object handling routines on *any* Graphics Library 2 machine:

- **The Graphics Library's display lists are too general**

Since the Graphics Library has to gear itself for all users, it uses strategies that cover any sort of object (i.e., any legal routine in any order at any time), at the expense of degraded speed, added size, or unnecessary complexity. You can optimize performance if you know in advance what your objects contain and bypass the Graphics Library. For example, if you know you are going to do a great number of moves and draws, you could store the points in an array, and step through the array using the fast immediate mode macros to display them. Remember that if you edit this sort of object, you are responsible for managing the array. For instance, if a series of moves and draws in the middle of the array is replaced by a longer series of moves and draws, the contents of the array must be shifted down to make room. Nonetheless, tailored code is generally more optimal than generic code.

- **Editing objects is slow**

Editing is slow because the data structures used in objects are complex. Some calls generate system calls to allocate more memory. Object and tag addresses are kept in hash tables and if there are a lot of objects, collisions in the hash table can make hash table creation and subsequent traversal

slow. The Graphics Library's object deletions and insertions can result in wasted space; although objects are compacted when this happens, compaction is also a slow process.

- **Editing fragments memory**

Editing also has a tendency to fragment memory, touching several virtual memory pages. Arrays inhabit contiguous memory, so using arrays to store objects will touch a minimum of pages. When you are not planning to edit, use of the Graphics Library display lists is efficient, although not necessarily desirable–the overhead of the hash tables, etc., that the Graphics Library maintains is still present in memory.

- **Keep only one copy of the data**

You cannot read a Graphics Library display list. If you intend to use the data in a Graphics Library display list for some other purpose (e.g., for computing centroids), you must make a copy of the data. If you maintain your own display list, you can avoid this redundancy by using the data in your display list for purposes other than display. This saves both space and maintenance overhead. As in the simple example above, altering geometry (the array) in effect alters the display list and accessing elements of the array, in effect, reads the display list. Furthermore, the Graphic Library display lists store only geometric information. Extra data, such as electrical information in a circuit design system or type of material in a mechanical CAD system, must be stored in a second display list. By writing your own display list, you have the option of storing nongeometric information with geometric information.

- **Better control–conditional display lists**

You can implement many features such as conditional display more efficiently with user-defined display lists. For example, you can make filled polygons unfilled by setting a flag, and then checking the flag when traversing the list of polygons. If you use conventional display lists, this would require an edit for each polygon.

- **Allows overlap of computing and graphics**

Calling large objects in Graphics Library display lists makes the processor spin in a buzz loop when the pipe is full; this does not make optimal use of the processor. You can design a program that loads the pipe and then does other work if you are familiar with the pipe and know where it tends to bottleneck with various routines. It is always possible to take advantage of

processor idle time when clearing the screen or when drawing large filled polygons (16 ms for a **clear()** on both the 68010 and 68020 workstations). With a 68020, 100 to 200 instructions can execute between the time the pipe is *almost* filled to the time it is empty. Note that actually filling the Geometry Engine buffer causes the processor to wait until the pipe empties to some threshhold; you must put enough in the pipe to allow some time, but not enough to fill the pipe.

Approach loading the pipe and doing concurrent processing with great care. Take into account whether limited real memory or fragmented memory could cause you to page when you perform this additional work. You can monitor paging to some extent with the *scan rate* field of **vmstat(1)**. Note that accessing the disk or the net, or making system calls triggers context switches, so do not attempt these procedures for filling idle time, except for **clear()** and when drawing large polygons

G.3 Example -- Defining Your Own Display List

The following example shows how to use the fast immediate macros to create a display list. It edits a grid to design 16x16 pixel in a scaled down version. As it stands, it has no practical use, but could be used as part of a program to design cursor glyphs or patterns. This example is not designed for speed, but for simplicity to demonstrate the subject of this paper, i.e., one would probably avoid picking because it involves unnecessary overhead. You could check for inclusion in various squares using the position of the grid on the screen in a much more efficient manner. It is worth noting that you can use picking with fast immediate mode and that it is not limited to conventional display list usage. Here the array *grid* is the display list. Changing values in *grid* effectively edits the display list.

```
/*
 *  Sample program to demonstrate user-defined display
 *  lists
 *  Program allows editing of a 16x16 grid of squares.
 *
 *  Note grid[][] IS the display list
 */

#define UNIX
```

```
#define IP2      /* 3000 series */
#define UC4
#define DC4

#include <gl.h>
#include <device.h>
#include <gl2/addrs.h>
#include <gl2/immed.h>
#include <gl2/gltypes.h>
#include <gl2/glerror.h>

#define BACKGRND  6
#define CURSOR    4
#define OUTLINE1  8
#define OUTLINE2 15

Colorindex grid[16][16];

 /* dummy main -- Because you can't call ginit and im_setup
        in same function */

main()
{
        register Scoord row, column;
        short i,j,basex,basey;

  /* start with blank grid */

        for(row=0;row<16;row++) {
                for(column=0;column<16;column++) {
                        grid[row][column] = BLACK;
                }
        }

        ginit();
        doublebuffer();
        gconfig();

        mapcolor(CURSOR,0,175,255);     /* cursor color */
        setcursor(0,4,7);

        mapcolor(BACKGRND,255,212,45); /* background color */

  /*
   *    Set up grid outline and make it permanent
   *    with writemask.
   *    Also set up color map  so that when WHITE is
   *    written over outline, it will still
   *    draw in the outline color
   */

        writemask(OUTLINE1);
        mapcolor(OUTLINE1,168,168,168); /* outline color */
```

```
        mapcolor(OUTLINE2,168,168,168); /* same when
                                           overwritten
                                           with WHITE */
        color(OUTLINE1);

        for(i=0;i<2;i++) {

                basex = 272;
                basey = 144;

                for(j=0;j<=16;j++) {
                        move2i(basex,basey);
                        draw2i(basex+480,basey);
                        basey += 30;
                }

                basex = 272;
                basey = 144;

                for(j=0;j<=16;j++) {
                        move2i(basex,basey);
                        draw2i(basex,basey+480);
                        basex += 30;
                }
                swapbuffers();
        }

  /* reset writemask for dynamic display */

        writemask(7);


  /* center and enlarge grid */

        translate(512.,384.,0.);
        scale(30.,30.,1.);
        translate(-8.,-8.,0.);

        main1();

        gexit();
}


main1()
{
        im_setup;
        im_color(BACKGRND);
        clear();

        while(!getbutton(RIGHTMOUSE)) {

                edit();       /* edit display list */
```

```
                traverse();  /* traverse display list */
        }

}

edit()
{
        short           buffer[50],i;
        long            numnames,numpicked;
        int             x,y;
        static Boolean  firstsquare;
        Colorindex      newcol;
        register Scoord row, column;

        im_setup;

        firstsquare = TRUE;

        while(getbutton(MIDDLEMOUSE)) {

                im_pushmatrix();

                pick(buffer,50);

        /* restate any and all transformations */

                ortho2(-0.5,1024.5,-0.5,767.5);
                im_translate(512.,384.,0.);
                im_scale(30.,30.,1.);
                im_translate(-8.,-8.,0.);

                pickit();

                numpicked = endpick(buffer);
                im_popmatrix();

                if(numpicked) {
                        if (firstsquare) {

                             if(grid[buffer[1]][buffer[2]])
                                     newcol = 0;
                             else
                                     newcol = 7;

                             firstsquare = FALSE;
                        }

                        grid[buffer[1]][buffer[2]]=newcol;

                        traverse();
                }
        }
```

```
        /* LEFTMOUSE --> clear grid */

        if(getbutton(LEFTMOUSE)) {
                for(row=0;row<16;row++) {
                        for(column=0;column<16;column++)
                                grid[row][column] = BLACK;
                }
        }

        firstsquare = TRUE;
}


traverse()
{
        register Scoord row, column;
        im_setup;

        im_pushmatrix();
        ortho2(-0.5,1023.5,-0.5,767.5);
        im_translate(100.,100.,0.);
        im_scale(5.,5.,1.);

 /* draw large grid */

        drawit();

        im_popmatrix();

 /* draw small grid */

        drawit();

        swapbuffers();

        im_color(BACKGRND);
        clear();        /* get clear going */
}

pickit() {

        register Scoord row, column;
        im_setup;

        for(row=0;row<16;row++) {

                im_loadname((short)row);
                im_pushname((short)row);

                for(column=0;column<16;column++) {

                        im_loadname((short)column);
```

```
                        im_pmv2i(row,column);
                        im_pdr2i(row+1,column);
                        im_pdr2i(row+1,column+1);
                        im_pdr2i(row,column+1);
                        im_pdr2i(row,column);
                        im_pclos();
                }
                im_popname();
        }
}

drawit() {

        register Scoord row, column;
        im_setup;

        for(row=0;row<16;row++) {
                for(column=0;column<16;column++) {
                        im_color(grid[row][column]);

                        im_pmv2i(row,column);
                        im_pdr2i(row+1,column);
                        im_pdr2i(row+1,column+1);
                        im_pdr2i(row,column+1);
                        im_pdr2i(row,column);
                        im_pclos();
                }
        }
}
```

G.4 Counting Instructions

It is often helpful to look at the assembly code your program generates to see if any inefficiencies exist and to locate opportunities for optimization. To look at an assembly listing in C, you can use the **-S** option to **cc(1)** and look at the resulting instructions does not guarantee increased performance because the 68020 can generally keep up with the pipe. The key is to spend as little time as possible loading the pipe and make maximum use of the processor while the pipe empties. **This makes the macros difficult to benchmark, because so many factors influence the benefits gained by their use.**

G.5 Conclusions

- The fast immediate mode macros are not always appropriate. Creating your own display lists with them requires considerable work that would not be necessary if you were to use the Graphics Library object handling routines.
- Fast immediate mode generates two to five times fewer assembly instructions; processes can be that much faster if you take advantage of the free cycles that fast immediate mode makes available.
- Display list mode is roughly as fast immediate mode, but you do not gain any of the benefits associated with defining your own display lists. These advantages include editing speed, flexibility, ability to read the display list, and the opportunity to overlap computing and graphics.
- The fast immediate mode macros defy benchmarks, except in specific cases. Since the 68020 can generally keep up with the pipe, with or without macros, performance benefits are achieved through overlapping of processing and graphics. Overlap is hard to quantify, except in specific situations. To benefit from overlap requires timing tests and much trial and error.

Appendix H: Using the Image Library

The Image Library is a set of routines that are specifically designed to aid in the manipulation of IRIS screen images. Images may be sent to files, where they can be subsequently read from and written to on a pixel-by-pixel basis. Uses for these routines include on-screen image manipulation, as well as image preparation for dumps to printers.

H.1 Image Files

Image files are used to store screen dumps, black and white images, color images, and colormaps. The IRIS stores values of the screen pixels comprising these images in 1-, 2-, or 3-dimensional arrays of either one- or two-byte unsigned integers. The range of pixel values is 0 - 255 for one-byte representation and 0 - 65534 for two-byte representation.

H.2 Opening and Closing an Image File

Image files may be opened for either reading or writing with the routine `iopen` and may be closed with the routine `iclose`. `iopen` returns a pointer to the structure `IMAGE`. Once an image file has been opened, this pointer is passed as an argument in all subsequent function calls referencing the image, in the same manner as the UNIX function `fopen`. In the following code examples, this argument is called `image`.

H.2.1 iopen - Open an Image File

```
IMAGE *iopen(file, mode [, type, dim, xsize, ysize, zsize])
char *file;
register char *mode;
unsigned int type, dim, xsize, ysize, zsize;
```

`iopen` opens an image file for either reading or writing and returns a pointer to `IMAGE` in the same manner as the UNIX Standard I/O Library function `fopen`. A return value of 0 means the attempt to open a file has failed.

Opening a File for Reading

To open an image file for reading, call `iopen`, specifying the file `name` and a `mode` of **"r"**. To determine the dimensions of the image, reference the `xsize, ysize,` and `zsize` fields of the structure `IMAGE`, through the pointer returned by `iopen`. Other information may be found in the `IMAGE` fields `name, colormap, type, max,` and `min`. See the include file *image.h* for the structure definition of `IMAGE`.

Opening a File for Writing

To open an image file for writing, call `iopen`, specifying the file `name`, a `mode` of **"w"**, followed by the `type`, the `dimension`, and the `xsize, ysize` and `zsize` of the image.

There are four image file `type` descriptors. The type descriptor given in the call to `iopen` determines whether the opened file's pixel values will be represented by one- or two-byte integers, and whether the image file will be run-length encoded or stored verbatim. The four descriptors are:

`RLE(1), RLE(2), VERBATIM(1),` and `VERBATIM(2).`

An image file may be of `dimension` 1, 2, or 3, representing a single row of pixels, an array of rows (i.e., a two-dimensional image) or an array of 2 dimensional images. A color image, for example, consists of 3 layers (one layer each for red, green, and blue color components) and is represented by

a 3-dimensional image file that is `xsize` by `ysize` by 3, while a black-and-white image has one layer and is represented by a 2-dimensional image file that is `xsize` by `ysize`.

H.2.2 iclose - Close an Image File

```
iclose(image)
register IMAGE  *image;
```

Closes an image file that was open for reading or writing. All output is flushed to the image file opened by `iopen`, and the output file is closed.

H.3 Reading from and Writing to Image Files

The following functions allow pixel values to be transferred to and from an image file. These functions provide an interface to an image file that is independent of the the file's pixel representation and encoding scheme.

H.3.1 putrow - Write a Row of Pixels from Buffer to Image File

```
putrow(image,buffer,y,z)
register IMAGE  *image;
unsigned short  *buffer;
unsigned        y, z;
```

Writes a row of pixels from a buffer to the specified image file. The buffer should be an array of `shorts` containing the pixel values. If the image file represents pixels values with single bytes, only the lower 8 bits of each `short` is stored in the image file. The row of the image being written to is given by `y`, and must have a value greater than or equal to 0 and less than the image's ysize. `z` indicates the image layer being written to. Numbering of layers begins with 0.

H.3.2 getrow - Read a Row of Pixels from Image File to Buffer

```
getrow(image,buffer,y,z)
register IMAGE *image;
unsigned short *buffer;
register unsigned int y, z;
```

Reads a row of pixels from the specified image file to a buffer. The buffer should be an array of shorts as for `putrow`. The row of the image being read from is given by `y`, while `z` indicates which layer of the image is being read from. The restrictions on arguments `y` and `z` in `getrow` are the same as in `putrow`.

H.4 Miscellaneous Functions

H.4.1 isetname - Name an Image File

```
isetname(image,name)
IMAGE *image;
char *name;
```

Copies the character string `name` into the name field of the image file. `name` must be no longer than 79 characters.

H.4.2 isetcolormap - Interpret Pixel Values

```
isetcolormap(image,colormap)
IMAGE *image;
int colormap;
```

Indicates how the pixels in an image file should be interpreted. The argument `colormap` may be one of following four values:

`CM_NORMAL`, `CM_DITHERED`, `CM_SCREEN`, and `CM_COLORMAP`.

`CM_NORMAL` is the default, and indicates that the pixels are to be interpreted as intensity values, with 0 representing black and the maximum value representing white. `CM_DITHERED` indicates that the range of pixel values is 0 to 255, and that these values index into a color map that has 3 bits for red 3 bits for green and 2 bitsfor blue. `CM_SCREEN` indicates that the pixels were copied from the screen and must be transformed by a color map to be meaningful. `CM_COLORMAP` indicates that the pixels in the image file represent a color map.

H.4.3 scrsave - Save Rectangular Region of Screen to Image File

```
scrsave(name,mapflag,left,right,bottom,top)
char *name;
long mapflag;
Screencoord left,right,bottom,top;
```

Saves a screen of any mode; RGB, single-, and double-buffer (it will save the back buffer in double-buffer mode). `name` is the name of the image file to which the screen region is saved. `mapflag` is a flag which when set causes the color map to be saved to the file `name`.*map* in addition to saving the screen region. `mapflag` is ignored in RGB mode. `left`, `right`, `bottom`, and `top` specify which region of the screen to save. `scrsave` returns a zero (0) value upon success or a non-zero value if the routine fails.

H.5 An Example

The following example shows how to open an image file and read its contents.

```
/*
 *   readimage -
 *      Read an image file but don't do anything with it!
 *
 *                                  Paul Haeberli - 1984
 *
 */
#include "image.h"

short rowbuf[4096];

main(argc,argv)
int argc;
char **argv;
{
    register IMAGE *image;
    register int i, y, ysize;
    register int z, zsize;

    if( argc<2 ) {
        fprintf(stderr,"usage: readimage infile0);
        exit(1);
    }
    if( (image=iopen(argv[1],"r")) == NULL ) {
        fprintf(stderr,"readimage: can't open
        input file %s",argv[1]);
        exit(1);
    }
    ysize = image->ysize;
    zsize = image->zsize;
    for(z=0; z<zsize; z++)
        for(y=0; y<ysize; y++)
            getrow(image,rowbuf,y,z);
}
```

Silicon Graphics, Inc.

Date _____

Your name _____

Title _____

Department _____

Company _____

Address _____

_____

_____

Phone _____

COMMENTS

Manual title and version _____

_____

Please list any errors, inaccuracies, or omissions you have found in this manual

_____

_____

_____

_____

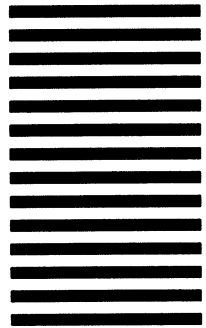
Please list any suggestions you may have for improving this manual

_____

_____

_____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 45 MOUNTAIN VIEW, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Silicon Graphics, Inc.
Attention: Technical Publications
2011 Stierlin Road
Mountain View, CA 94043-1321