
C Edition

Document Version 4.0

Document Number 007-1203-040
Technical Publications:
Lorrie Williams
Melissa Heinrich
Claudia Lohnes
Kevin Walsh

Engineering:
Kurt Akeley
Herb Kuta

© Copyright 1990, Silicon Graphics, Inc. - All rights reserved

This document contains proprietary information of Silicon Graphics, Inc. The contents of this document may not be disclosed to third parties, copied or duplicated in any form, in whole or in part, without the prior written permission of Silicon Graphics, Inc.

U.S. Government Limited Rights
Use, duplication or disclosure of the technical data contained in this document by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013, and/or similar or successor clauses in the FAR, or the DOD or NASA FAR Supplement. Unpublished rights reserved under Copyright laws of the United States. Contractor/manufacturer is Silicon Graphics Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94039-7311.

C Edition
Document Version 4.0
Document Number 007-1203-040

Silicon Graphics, Inc.
Mountain View, California

IRIS is a registered trade mark of Silicon Graphics, Inc. IRIX, Power Series, IRIS-4D, Personal IRIS, Geometry Link, Geometry Partners, Geometry Engine, and Geometry Accelerator are trademarks of Silicon Graphics, Inc. IBM® is a trademark of International Business Machines Corporation.
NAME

intro – description of routines in the Graphics Library and Distributed Graphics Library

OVERVIEW

This manual is the reference manual for the routines of the Graphics Library (GL) and the Distributed Graphics Library (DGL). For a more tutorial introduction to the GL and DGL, see the Graphics Library Programmer's Guide and the “Using the GL/DGL Interfaces” section of the 4Sight Programmer's Guide.

In general, all routines in the GL are supported in the DGL. However, in some routines there are minor differences. In addition, some routines (dglopen and dglclose) are supported in the DGL but not the GL. Where there is a difference for a routine, it is noted on its manual page.

The manual pages are available on-line. To view them, use the IRIX command:

```
man routine-name <Enter>
```

HOW A MANUAL PAGE IS ORGANIZED

A manual page provides the specification of a GL or DGL routine. Because these pages are intended as on-line reference material, they tend to be terse. A page is divided into a number of sections:

NAME

lists the name of the routine or routines described by the manual page.

C SPECIFICATION

lists the type declarations for the routine and its parameters.

PARAMETERS

describes the parameters of the routine.

FUNCTION RETURN VALUE

describes what the routine returns if it is a function.
DESCRIPTION

describes how to use the routine.

SEE ALSO

lists related routines or other sources of information.

EXAMPLE

gives an example of how the routine is used.

NOTES

highlights information concerning the limitations of the routine and
differences in its behavior on the various IRIS-4D models.

BUGS

describes deviations from the specified behavior that may be fixed in
a future release.

HEADER FILES

There are three header files in /usr/include/gl that you should probably
include in code that calls routines from the Graphics Library. The files
are gl.h, get.h, and device.h.

TYPE DECLARATIONS

We have constructed type declarations for C wherever they add to the
readability of the code. Here are the type definitions as found in
<gl/gl.h>:

```
#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 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;
typedef unsigned short Linestyle;
typedef unsigned short Cursor[16];
typedef unsigned short Pattern16[PATTERN_16_SIZE];
typedef unsigned short Pattern32[PATTERN_32_SIZE];
typedef unsigned short Pattern64[PATTERN_64_SIZE];
typedef struct {
    unsigned short offset;
    Byte w,h;
    char xoff,yoff;
    short width;
} Fontchar;
typedef long Object;
typedef long Tag;
typedef long Offset;
acbuf — operate on the accumulation buffer
acsiz.e — specify the number of bitplanes per color component in the accumulation buffer
addtopup — adds items to an existing pop-up menu
afunction — specify alpha test function
arc, arci, archs — draw a circular arc
arcf, arcfi, arcfi — draw a filled circular arc
attachcursor — attaches the cursor to two valuators
backbuffer — enable and disable drawing to the back buffer
backface — turns backfacing polygon removal on and off
bbox2, bbox2i, bbox2s — culls and prunes to bounding box and minimum pixel radius
bgnclosedline — delimit the vertices of a closed line
bgnline — delimit the vertices of a line
bgnpoint — delimit the interpretation of vertex routines as points
bgnpolygon — delimit the vertices of a polygon
bgnqstrip — delimit the vertices of a quadrilateral strip
bgnnsurface — delimit a NURBS surface definition
bgnntmesh — delimit the vertices of a triangle mesh
bgntrim — delimit a NURBS surface trimming loop
blankscreen — controls screen blanking
blanktime — sets the screen blanking timeout
blendfunction — computes a blended color value for a pixel
blink — changes a color map entry at a selectable rate
blkqread — reads multiple entries from the queue
c3f, c3i, c3s, c4f, c4i, c4s — sets the RGB (or RGBA) values for the current color vector
callfunc – calls a function from within an object
 calleeobj – draws an instance of an object
 charstr – draws a string of raster characters on the screen
 chunksize – specifies minimum object size in memory
 circ, circi, circs – outlines a circle
 circf, circfi, circfs – draws a filled circle
 clear – clears the viewport
 clearhitcode – sets the hitcode to zero
 clipplane – specify a plane against which all geometry is clipped
 clkon, clkoff – control keyboard click
 closeobj – closes an object definition
 cmode – sets color map mode as the current mode.
 cmov, cmovi, cmove, cmov2, cmov2i, cmov2s – updates the current
 character position
 color, colorf – sets the color index in the current draw mode
 compactify – compacts the memory storage of an object
 concave – allows the system to draw concave polygons
 cpack – specifies RGBA color with a single packed 32-bit integer
 crv – draws a curve
 crvn – draws a series of curve segments
 curorigin – sets the origin of a cursor
 curson, cursoff – control cursor visibility by window
 curstype – defines the type and/or size of cursor
 curvebasis – selects a basis matrix used to draw curves
 curveit – draws a curve segment
 curveprecision – sets number of line segments used to draw a curve segment
cyclemap – cycles between color maps at a specified rate
czclear – clears the color bitplanes and the z-buffer simultaneously
dbtext – sets the dial and button box text display
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 a tag from the current open object
depthcue – turns depth-cue mode on and off
dgliclose – closes the DGL server connection
dglopen – opens a DGL connection to a graphics server
dopup – displays the specified pop-up menu
doublebuffer – sets the display mode to double buffer mode
draw, drawi, draws, draw2, draw2i, draw2s – draws a line
drawmode – selects which GL framebuffer is drawable
editobj – opens an object definition for editing
endclosedline – delimit the vertices of a closed line
endfeedback – control feedback mode
endfullscrn – ends full-screen mode
endline – delimit the vertices of a line
endpick – turns off picking mode
endpoint – delimit the interpretation of vertex routines as points
endpolygon – delimit the vertices of a polygon
endpupmode – obsolete routine
endqstrip – delimit the vertices of a quadrilateral strip
endselect – turns off selecting mode
endsurface – delimit a NURBS surface definition
endtmesh – delimit the vertices of a triangle mesh
endtrim – delimit a NURBS surface trimming loop
feedback – control feedback mode
finish – blocks until the Geometry Pipeline is empty
fogvertex – specify fog density for per-vertex atmospheric effects
font – selects a raster font for drawing text strings
foreground – prevents a graphical process from being put into the background
freenup – deallocates a menu
frontbuffer – enable and disable drawing to the front buffer
frontface – turns front-facing polygon removal on and off
fudge – specifies fudge values that are added to a graphics window
fullscrn – allows a program write to the entire screen
gammaramp – defines a color map ramp for gamma correction
gbegin – create a window that occupies the entire screen
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 back-facing polygons will appear
getbuffer – indicates which buffers are enabled for writing
getbutton – returns the state of a button
getcode – 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 — obsolete routine
getdescender — returns the character characteristics
getdev — reads a list of valuators at one time
getdisplaymode — returns the current display mode
getdrawmode — returns the current drawing mode
getfont — returns the current raster font number
getgdesc — gets graphics system description
getgpos — gets the current graphics position
getheight — returns the maximum character height in the current raster font
gethitcode — returns the current hitcode
getlsbackup — has no function in the current system
getlsrepeat — returns the linestyle repeat count
getlstyle — returns the current linestyle
getlinewidth — returns the current linewidth
getmap — returns the number of the current color map
getmatrix — returns a copy of a transformation matrix
getmcolor — gets a copy of the RGB values for a color map entry
getmmode — returns the current matrix mode
getmonitor — returns the type of the current display monitor
getnurbsproperty — returns the current value of a trimmed NURBS surfaces display property
getopenobj — returns the identifier of the currently open object
getorigin – returns the position of a graphics window
getothermonitor – obsolete routine
getpattern – returns the index of the current pattern
getplanes – returns the number of available bitplanes
getport – obsolete routine
getresetls – returns the state of linestyle reset mode
getscrbox – read back the current computed screen bounding box
getscrmask – returns the current screen mask
getshade – obsolete routine
getsize – returns the size of a graphics window
getsm – returns the current shading model
getvaluator – returns the current state of a valuator
getvideo – get video hardware registers
getviewport – gets a copy of the dimensions of the current viewport
getwritemask – returns the current writemask
getwscrn – returns the screen upon which the current window appears
getzbuffer – returns whether z-buffering is on or off
gexit – exits graphics
gfflush – flushes the DGL client buffer
ginit – create a window that occupies the entire screen
glcompat – controls compatibility modes
greset – resets graphics state
gRGBcolor – gets the current RGB color values
gRGBcursor – obsolete routine
gRGBmask – returns the current RGB writemask
gselect – puts the system in selecting mode
gsync - waits for a vertical retrace period
gversion - returns graphics hardware and library version information
iconszie - specifies the icon size of a window
icontitle - assigns the icon title for the current graphics window.
imakebackground - registers the screen background process
initnames - initializes the name stack
ismex - obsolete routine
isobj - returns whether an object exists
isqueued - returns whether the specified device is enabled for queuing
istag - returns whether a tag exists in the current open object
keepaspect - specifies the aspect ratio of a graphics window
lampon, lampoff - control the keyboard display lights
linesmooth - specify antialiasing of lines
linewidth - specifies width of lines
imbind - selects a new material, light source, or lighting model
imcolor - change the effect of color commands while lighting is active
imdef - defines or modifies a material, light source, or lighting model
loadmatrix - loads a transformation matrix
loadname - loads a name onto the name stack
logicop - specifies a logical operation for pixel writes
lookat - defines a viewing transformation
lrectread - reads a rectangular array of pixels into CPU memory
lrectwrite - draws a rectangular array of pixels into the frame buffer
lRGBRange - sets the range of RGB colors used for depth-cueing
lsbackup - controls whether the ends of a line segment are colored
lsetdepth - sets the depth range
Ishaderange – sets range of color indices used for depth-cueing
Isrepeat – sets a 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 the maximum size of a graphics window
minsize – specifies the minimum size of a graphics window
mmode – sets the current matrix mode
move, movei, moves, move2, move2i, move2s – moves the current graphics position to a specified point
mswapbuffers – swap multiple framebuffers simultaneously
multimap – organizes the color map as a number of smaller maps
multimatrix – premultiplies the current transformation matrix
n3f – specifies a normal
newpup – allocates and initializes a structure for a new menu
newtag – creates a new tag within an object relative to an existing tag
nmode – specify renormalization of normals
noborder – specifies a window without any borders
noise – filters valuator motion
noport – specifies that a program does not need screen space
normal – obsolete routine
nurbscurve – controls the shape of a NURBS trimming curve
nurbssurface – controls the shape of a NURBS surface
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, ortho2 – define an orthographic projection transformation
overlay – allocates bitplanes for display of overlay colors
pagecolor – sets the color of the textport background
passsthrough – passes a single token through the Geometry Pipeline
patch – draws a surface patch
patchbasis – sets current basis matrices
patchcurves – sets the number of curves used to represent a patch
patchprecision – sets the precision at which curves are drawn in a patch
pclos – closes a filled polygon
pdr, pdri, pdrs, pdr2, pdr2i, pdr2s – specifies the next point of a polygon
perspective – defines a perspective projection transformation
pick – puts the system in picking mode
picksize – sets the dimensions of the picking region
pixmode – specify pixel transfer mode parameters
pmv, pmvi, pmvs, pmv2, pmv2i, pmv2s – specifies the first point of a polygon
pnt, pnti, pnts, pnt2, pnt2i, pnt2s – draws a point
pntsMOOTH – specify antialiasing of points
polarview – defines the viewer’s position in polar coordinates
polf, polfi, polfs, polf2, polf2i, polf2s – draws a filled polygon
poly, polyi, polys, poly2, poly2i, poly2s – outlines a polygon
polymode – control the rendering of polygons
polysMOOTH – specify antialiasing of polygons
popattributes – pops the attribute stack
popmatrix – pops the transformation matrix stack
popname – pops a name off the name stack
popviewport – pops the viewport stack
prefposition – specifies the preferred location and size of a graphics window
prefsize – specifies the preferred size of a graphics window
pupmode – obsolete routine
pushattributes – pushes down the attribute stack
pushmatrix – pushes down the transformation matrix stack
pushname – pushes a new name on the name stack
pushviewport – pushes down the viewport stack
pwlcurve – describes a piecewise linear trimming curve for NURBS surfaces
qcontrol – administers event queue
qdevice – queues a device
qenter – creates an event queue entry
qgetfd – returns the file descriptor of the event queue
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 rational curve
rcrvn – draws a series of curve segments
rdr, rdri, rdrs, rdr2, rdr2i, rdr2s – relative draw
readpixels – returns values of specific pixels
readRGB – gets values of specific pixels
readsource – sets the source for pixels that various routines read
rect, recti, rects – outlines a rectangular region
rectcopy – copies a rectangle of pixels with an optional zoom
rectf, rectfi, rectfs – fills a rectangular area
rectread – reads a rectangular array of pixels into CPU memory
rectwrite – draws a rectangular array of pixels into the frame buffer
rectzoom – specifies the zoom for rectangular pixel copies and writes
resetls – controls the continuity of linestyles
reshapeviewport – sets the viewport to the dimensions of the current graphics window
RGBcolor – sets the current color in RGB mode
RGBcursor – obsolete routine
RGBmode – sets a rendering and display mode that bypasses the color map
RGBrange – obsolete routine
RGBwritemask – grants write access to a subset of available bitplanes
ringbell – rings the keyboard bell
rmv, rmvi, rmvs, rmv2, rmv2i, rmv2s – relative move
rotate, rot – rotate graphical primitives
rpatch – draws a rational surface patch
rpdr, rp dri, rpd rs, rp dr2, rp dr2i, rp dr2s – relative polygon draw
rpmv, rpmvi, rpmvs, rpmv2, rpmv2i, rpmv2s – relative polygon move
sbox, sboxi, sboxs – draw a screen-aligned rectangle
sboxf, sboxfi, sboxfs – draw a filled screen-aligned rectangle
scale – scales and mirrors objects
sclear – clear the stencil planes to a specified value
scrbox – control the screen box
screenspace — map world space to absolute screen coordinates
scrmask — defines a rectangular screen clipping mask
scrnattach — attaches the input focus to a screen
scrnselect — selects the screen upon which new windows are placed
scrsubdivide — subdivide lines and polygons to a screen-space limit
setbell — sets the duration of the beep of the keyboard bell
setcursor — sets the cursor characteristics
setdblights — sets the lights on the dial and button box
setdepth — obsolete routine
setlinestyle — selects a linestyle pattern
setmap — selects one of the small color maps provided by multimap mode
setmonitor — sets the monitor type
setnurbsproperty — sets a property for the display of trimmed NURBS surfaces
setpattern — selects a pattern for filling polygons and rectangles
setpup — sets the display characteristics of a given pop up menu entry
setshade — obsolete routine
setvaluator — assigns an initial value and a range to a valuator
setvideo — set video hardware registers
shademodel — selects the shading model
shaderange — obsolete routine
singlebuffer — writes and displays all bitplanes
smoothline — obsolete routine
spclos — obsolete routine
splf, splfi, splfs, splf2, splf2i, splf2s — draws a shaded filled polygon
stencil — alter the operating parameters of the stencil
stensize - specify the number of bitplanes to be used as stencil planes
stepunit - specifies that a graphics window change size in discrete steps
strwidth - returns the width of the specified text string
subpixel - controls the placement of point, line, and polygon vertices
swapbuffers - exchanges the front and back buffers of the normal framebuffer
swapinterval - defines a minimum time between buffer swaps
swapmesh - toggles the triangle mesh register pointer
swinopen - creates a graphics subwindow
swritemask - specify which stencil bits can be written
t2d, t2f, t2i, t2s - specify a texture coordinate
tevbind - selects a texture environment
tevdef - defines a texture mapping environment
texbind - selects a texture function
texdef2d - convert a 2-dimensional image into a texture
texgen - specify automatic generation of texture coordinates
textcolor - sets the color of text in the textport
textinit - initializes the textport
textport - positions and sizes the textport
tie - ties two valuators to a button
tpon, tpoff - control the visibility of the textport
translate - translates graphical primitives
underlay - allocates bitplanes for display of underlay colors
unqdevice - disables the specified device from making entries in the event queue
v2d, v2f, v2i, v2s, v3d, v3f, v3i, v3s, v4d, v4f, v4i, v4s - transfers a 2-D, 3-D, or 4-D vertex to the graphics pipe
videocmd – initiates a command transfer sequence on an optional video peripheral
viewport – allocates an area of the window for an image
winattach – obsolete routine
winclose – closes the identified graphics window
winconstraints – binds window constraints to the current window
winedepth – measures how deep a window is in the window stack
window – defines a perspective projection transformation
winget – returns the identifier of the current graphics window
winmove – moves the current graphics window by its lower-left corner
winopen – creates a graphics window
winpop – moves 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 graphics window behind all other windows
winset – sets the current graphics window
wintitle – adds a title bar to the current graphics window
wmpack – specifies RGBA writemask with a single packed integer
writemask – grants write permission to bitplanes
writepixels – paints a row of pixels on the screen
writeRGB – paints a row of pixels on the screen
xfpt, xfpti, xfpts, xfpt2, xfpt2i, xfpt2s, xfpt4, xfpt4i, xfpt4s – multiplies a point by the current matrix in feedback mode
zbuffer – enable or disable z-buffer operation in the current framebuffer
zclear – initializes the z-buffer of the current framebuffer
zdrow – enables or disables drawing to the z-buffer
zfunction – specifies the function used for z-buffer comparison by the current framebuffer
zsource – selects the source for z-buffering comparisons
zwritemask – specifies a write mask for the z-buffer of the current framebuffer
NAME

acbuf – operate on the accumulation buffer

C SPECIFICATION

void acbuf(op, value)
long op;
float value;

PARAMETERS

op expects one of six symbolic constants:

AC_CLEAR: The red, green, blue, and alpha accumulation buffer contents are all set to value (rounded to the nearest integer). value is clamped to the range of a 16-bit signed integer.

AC_ACCUMULATE: Pixels are taken from the current readsource bank (front, back, or zbuffer). Their red, green, blue, and alpha components are each scaled by value. The resulting 16-bit/component pixels are added to the pixels already present in the accumulation buffer. The range of value is -255.996 through 255.996. Arguments outside this range are clamped to it. Accumulated values are NOT clamped to the signed 16-bit range of the accumulation buffer. Thus overflow is avoided only by limiting the range of accumulation operations.

AC_CLEAR_ACCUMULATE: An efficient combination command whose effect is to first clear the accumulation buffer contents to zero, then add as per AC_ACCUMULATE. Ranges and clamping are as per AC_ACCUMULATE.

AC_RETURN: Pixels are taken from the accumulation buffer. Their red, green, blue, and alpha components are each scaled by value. The resulting 8-bit/component pixels are then written to the currently enabled drawing buffers (front, back, or zbuffer). All special pixel operations (zbuffer, blendfunction, logicrop, stencil, texture mapping, etc.) are ignored during this transfer. Destination values are simply replaced. The operation is limited by the current viewport and screenmask, however. The range of value is 0.0 through 1.0. Arguments outside this range are
clamped to it. After being scaled by value, color components are clamped to the range 0 through 255 before being written to the enabled drawing buffers.

**AC_MULT:** The red, green, blue, and alpha components of each accumulation buffer pixel are scaled by value.

**AC_ADD:** value is added to each red, green, blue, and alpha component of each pixel in the accumulation buffer.

value expects a float point value. op determines how value is used.

**DESCRIPTION**

The accumulation buffer is a bank of 64-bit pixels, 16 bits each for red, green, blue, and alpha, that is mapped 1-to-1 with screen pixels. Pixel images stored in the normal framebuffer (typically generated from geometric data) can be added to the accumulation buffer. These pixels are scaled during the transfer by a floating-point value (of limited range and resolution). Later, the accumulated image can be returned to the normal frame buffer, again while being scaled.

Effects such as antialiasing (of points, lines, and polygons), motion-blur, and depth-of-field can be created by accumulating images generated with different transformation matrixes. Predictable effects are possible only when subpixel mode is TRUE (see subpixel).

**readsource** mode is shared with other pixel read operations, including lrectread and rectcopy. rectzoom, however, has no effect on accumulation operation.

All accumulation buffer operations are limited to the area of the current screenmask, which itself is limited to the current viewport.

The accumulation buffer is a part of the normal framebuffer. acbuf should be called only while draw mode is NORMALDRAW, and while the normal framebuffer is in RGB mode.

**SEE ALSO**

acsiz, drawmode, subpixel, scrmask
NOTES

An error is reported, and no action is taken, if accumulate is called while acsize is zero.
NAME

acsiz – specify the number of bitplanes per color component in the accumulation buffer

C SPECIFICATION

void acsiz(planes)
long planes;

PARAMETERS

planes specifies the number of bitplanes to be reserved for each color component in the accumulation buffer. Accepted values are 0 (default) and 16.

DESCRIPTION

Rendered images are accumulated (see acbuf) into a framebuffer with more than 8 bits per color component. acsiz specifies the size of the accumulation buffer. You must call gconfig after acsiz to activate the new size specification.

By default the accumulation buffer size is zero, meaning that images cannot be accumulated.

The 16-bit per component accumulation buffer is signed; it therefore supports accumulated values in the range -32768 through 32767.

SEE ALSO

acbuf, drawmode, gconfig

NOTE

This routine is available only in immediate mode.

The accumulation buffer is available only in the normal framebuffer. acsiz should be called only while draw mode is NORMALDRAW.

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support the accumulation buffer. Use getgdesc to determine what support is available for accumulation buffering.
NAME

addtopup — adds items to an existing pop-up menu

C SPECIFICATION

```c
void addtopup(pup, str, arg)
long pup;
String str;
long arg;
```

PARAMETERS

- **pup** expects the menu identifier of the menu to which you want to add. The menu identifier is the returned function value of the menu creation call to either newpup or defpup functions.

- **str** expects a pointer to the text that you want to add as a menu item. In addition, you have the option of pairing an "item type" flag with each menu item. There are seven menu item type flags:
  
  - **%t** marks item text as the menu title string.
  - **%F** invokes a routine for every selection from this menu except those marked with a %n. You must specify the invoked routine in the arg parameter. The value of the menu item is used as a parameter of the executed routine. Thus, if you select the third menu item, the system passes 3 as a parameter to the function specified by %F.
  - **%f** invokes a routine when this particular menu item is selected. You must specify the invoked routine in the arg parameter. The value of the menu item is passed as a parameter of the routine. Thus, if you select the third menu item, the system passes 3 as a parameter to the routine specified by %f. If you have also used the %F flag within this menu, then the result of the %f routine is passed as a parameter of the %F routine.
  - **%l** adds a line under the current entry. You can use this as a visual cue to group like entries together.
%m  pops up a menu whenever this menu item is selected. You must provide the menu identifier of the new menu in the arg parameter.

%n  like %f, this flag invokes a routine when the user selects this menu item. However, %n differs from %f in that it ignores the routine (if any) specified by %F. The value of the menu item is passed as a parameter of the executed routine. Thus, if you select the third menu item, the system passes 3 as a parameter to the function specified by %f.

%xn assigns a numeric value to this menu item. This values overrides the default position-based value assigned to this menu item (e.g., the third item is 3). You must enter the numeric value as the n part of the text string. Do not use the arg parameter to specify the numeric value.

NOTE: If you use the vertical bar delimiter, "|", you can specify multiple menu items in a text string. However, because there is only one arg parameter, the text string can contain no more than one item type that references the arg parameter.

arg  expects the command or submenu that you want to assign to the menu item. You can have only one arg parameter for each call to addtopup.

DESCRIPTION

addtopup adds items to the bottom of an existing pop-up menu. You can build a menu by using a call to newpup to create a menu, followed by a call to addtopup for each menu item that you want to add to the menu. To activate and display the menu, submit the menu to dopup.
EXAMPLE

This example creates a menu with a submenu:

```c
submenu = newpup();
addtopup(submenu, "rotate %f", dorota);
addtopup(submenu, "translate %f", dotran);
addtopup(submenu, "scale %f", doscal);
menu = newpup();
addtopup(menu, "sample %t", 0);
addtopup(menu, "persp", 0);
addtopup(menu, "xform %m", submenu);
addtopup(menu, "greset %f", greset);
```

Because neither the "sample" menu title nor the "persp" menu item refer to the arg parameter, you can group "sample", "persp", and "xform" in a single call:

```c
addtopup(menu, "sample %t | persp | xform %m", submenu);
```

SEE ALSO

defpup, dopup, freepup, newpup

NOTES

This routine is available only in immediate mode.

When using the Distributed Graphics Library (DGL), you can not call other DGL routines within a function that is called by a popup menu, i.e. a function given as the argument to a %f or %F item type.
NAME

afunction – specify alpha test function

C SPECIFICATION

void afunction(ref, func)
long ref, func;

PARAMETERS

ref expects a reference value with which to compare source alpha at each pixel. This value should be in the range 0 through 255.

func expects one of two flags specifying the alpha comparison function: AF_NOTEQUAL and AF_ALWAYS (the default).

DESCRIPTION

afunction makes the drawing of pixels conditional on the relationship of the incoming alpha value to a reference constant value. It is typically used to avoid updating either the color or the z field of a framebuffer pixel when the incoming pixel is completely transparent. Arguments ref and func specify the conditions under which the pixel will be drawn. The incoming (source) alpha value is compared to ref with function func, and if the comparison passes, the incoming pixel is drawn (conditional on subsequent z-buffer tests). Thus afunction can be called with arguments 0, AF_NOTEQUAL to defeat drawing of completely transparent pixels. This assumes that incoming alpha is proportional to pixel coverage, as it is when either pointsmooth or linesmooth is being used.

afunction testing follows scan conversion, texture mapping, and stencil operation, but preceeds all other pixel tests. Thus, if the test fails, neither the color nor zbuffer contents will be modified. afunction operates on all pixel writes, including those resulting from the scan conversion of points, lines, and polygons, and from pixel write and copy operations. afunction does not affect screen clear operation, however.
SEE ALSO

blendfunction

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support afuction. Use getgdesc to determine what support is available for afuction.

BUGS

On IRIS-4D VGX models afuction cannot be enabled while stencil is being used. Also, ref must be 0.
NAME

arc, arci, arcs – draw a circular arc

C SPECIFICATION

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

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

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

All of the routines named above are functionally the same. They differ only in the type assignments of their parameters.

PARAMETERS

x       expects the x coordinate of the center of the arc. The center of the arc is the center of the circle that would contain the arc.

y       expects the y coordinate of the center of the arc. The center of the arc is the center of the circle that would contain the arc.

radius  expects the length of the radius of the arc. The radius of the arc is the radius of the circle that would contain the arc.

startang expects the measure of the start angle of the arc. The start angle of the arc is measured from the positive x-axis.

endang  expects the measure of the end angle of the arc. The end angle of the arc is measured from the positive x-axis.

DESCRIPTION

arc draws an unfilled circular arc in the x-y plane (z = 0). To draw an arc in a plane other than the x-y plane, define the arc in the x-y plane and then rotate or translate the arc.
An arc is drawn as a sequence of line segments, and therefore inherits all properties that affect the drawing of lines. These include the current color, writemask, line width, stipple pattern, shade model, line antialiasing mode, and subpixel mode. The stipple pattern is initialized to bit zero of the current linestyle before the arc is drawn, then shifted continuously through the segments of the arc.

An arc is defined in terms of the circle that contains it. All references to the radius and center of the arc refer to the radius and center of the circle that contains the arc. The angle swept out by the arc is the angle from the start angle counter-clockwise to the end angle.

The start and end angles are defined relative to the positive x-axis. (To speak more precisely, because the arc might not be centered on the origin, the start and end angles are defined relative to the right horizontal radius of the circle containing the arc). Positive values for an angle indicate a counter-clockwise rotation from the horizontal. Negative values indicate a clockwise rotation from the horizontal.

The basic unit of angle measure is a tenth of a degree. The value 900 indicates an angle of 90 degrees in a counter-clockwise direction from the horizontal. Thus, an arc that spans from a start angle of 10 degrees \((\text{startang} = 100)\) to an end angle of 5 degrees \((\text{endang} = 50)\) is almost a complete circle.

After \texttt{arc} executes, the graphics position is undefined.

**SEE ALSO**

\texttt{arcf}, \texttt{bgnclosedline}, \texttt{circ}, \texttt{crvn}, \texttt{linewidth}, \texttt{linesmooth}, \texttt{lsrepeat}, \texttt{scrsdivide}, \texttt{setlinestyle}, \texttt{shademodel}, \texttt{subpixel}

**BUGS**

When the line width is greater than 1, small notches will appear in arcs, because of the way wide lines are implemented.
NAME

arcf, arcfi, arcfs – draw a filled circular arc

C SPECIFICATION

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

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

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

All of the routines named above are functionally the same. They differ
only in the type assignments of their parameters.

PARAMETERS

x expects the x coordinate of the center of the filled arc. The
center of the filled arc is the center of the circle that would
contain the arc.

y expects the y coordinate of the center of the filled arc. The
center of the filled arc is the center of the circle that would
contain the arc.

radius expects the length of the radius of the filled arc. The radius of
the filled arc is the radius of the circle that would contain the
filled arc.

startang expects the measure (in tenths of a degree) of the start angle of
the filled arc. The start angle of the filled arc is measured relative
to the positive x-axis.

endang expects the measure (in tenths of a degree) of the end angle of
the filled arc. The end angle of the filled arc is measured relative
to the positive x-axis.
DESCRIPTION

arcf draws a filled circular arc in the x-y plane \((z = 0)\). The filled area is bound by the arc and by the start and end radii. To draw an arc in a plane other than the x-y plane, define the arc in the x-y plane and then rotate or translate the arc.

An arc is drawn as a single polygon, and therefore inherits all properties that affect the drawing of polygons. These include the current color, writemask, fill pattern, shade model, polygon antialiasing mode, polygon scan conversion mode, and subpixel mode. Front-face and back-face elimination work correctly with filled arcs, which are front-facing when viewed from the positive \(z\) half-space.

A filled arc is defined in terms of the circle that contains it. All references to the radius and the center of the filled arc refer to the radius and center of the circle that contains the filled arc. The angle swept out by the filled arc is the angle from the start angle counter-clockwise to the end angle.

The start and end angles are defined relative to the positive x-axis. (To speak more precisely, because the arc might not be centered on the origin, the start and end angles are defined relative to the right horizontal radius of the circle containing the arc). Positive values for an angle indicate a counter-clockwise rotation from the horizontal. Negative values indicate a clockwise rotation from the horizontal.

The basic unit of angle measure is a tenth of a degree. The value 900 indicates an angle of 90 degrees in a counter-clockwise direction from the horizontal. Thus, a filled arc that spans from a start angle of 10 degrees \((\text{startang} = 100)\) to an end angle of 5 degrees \((\text{endang} = 50)\) is almost a complete filled circle.

After arcf executes, the graphics position is undefined.

SEE ALSO

arc, backface, bgnpolygon, cirm, frontface, polymode, polysmooth, scrsubdivide, setpattern, shademodel, subpixel
NAME

attachcursor – attaches the cursor to two valuators

C SPECIFICATION

void attachcursor(vx, vy)
Device vx, vy;

PARAMETERS

vx expects the valuator device number for the device that controls the horizontal location of the cursor. By default, vx is MOUSEX.

vy expects the valuator device number for the device that controls the vertical location of the cursor. By default, vy is MOUSEY.

DESCRIPTION

attachcursor attaches the cursor to the movement of two valuators. Both vx and vy are valuator device numbers. (See Appendix A, Valuators, for a list of device numbers.) The values at vx and vy determine the cursor position in screen coordinates. Every time the values at vx or vy change, the system redraws the cursor at the new coordinates.

SEE ALSO

noise, tie

NOTE

This routine is available only in immediate mode.
NAME

backbuffer, frontbuffer – enable and disable drawing to the back or front buffer

C SPECIFICATION

void backbuffer(b)
Boolean b;

void frontbuffer(b)
Boolean b;

PARAMETERS

b expects either TRUE or FALSE.
TRUE enables updating in the back/front bitplane buffer.
FALSE turns off updating in the back/front bitplane buffer.

DESCRIPTION

The IRIS framebuffer is divided into four separate GL framebuffers: pop-up, overlay, underlay, and normal. Three of these framebuffers, overlay, underlay, and normal, can be configured in double buffer mode. When so configured, a framebuffer includes two color bitplane buffers: one visible bitplane buffer, called the front buffer, and one non-visible bitplane buffer, called the back buffer. The commands swapbuffers and mswapbuffers interchange the front and back buffer assignments.

By default, when a framebuffer is configured in double buffer mode, drawing is enabled in the back buffer, and disabled in the front buffer. frontbuffer and backbuffer enable and disable drawing into the front and back buffers, allowing the default to be overridden. Its is acceptable to enable neither front nor back, either front or back, or both front and back simultaneously. Note, for example, that z-buffer drawing continues to update the z-buffer with depth values when neither the front buffer nor the back buffer is enabled for drawing.

frontbuffer and backbuffer state is maintained separately for each of the overlay, underlay, and normal framebuffers. Calls to these routines affect the framebuffer that is currently active, based on the current drawmode.
backbuffer is ignored when the currently active framebuffer is in single buffer mode. frontbuffer is also ignored when the currently active framebuffer is in single buffer mode, unless zdraw is enabled for that framebuffer (see zdraw).

After each call to gconfig, backbuffer is enabled and frontbuffer is disabled.

SEE ALSO
drawmode, doublebuffer, getbuffer, gconfig, singlebuffer, swapbuffers, zdraw

NOTE
Only VGX graphics support double buffer operation in the overlay and underlay framebuffers.
NAME

backface – turns back-facing polygon removal on and off

C SPECIFICATION

void backface(b)
  Boolean b;

PARAMETERS

b  expects either TRUE or FALSE.
  TRUE suppresses the display of back-facing filled polygons.
  FALSE allows the display of back-facing filled polygons.

DESCRIPTION

backface allows or suppresses the display of back-facing filled polygons. If your programs represent solid objects as collections of polygons, you can use this routine to remove hidden surfaces. This routine works best for simple convex objects that do not obscure other objects.

A back-facing polygon is defined as a polygon whose vertices are in clockwise order in screen coordinates. When back-facing polygon removal is on, the system displays only polygons whose vertices are in counter-clockwise order. For complicated objects, this routine alone may not remove all hidden surfaces. To remove hidden surfaces for more complicated objects or groups of objects, your routine needs to check the relative distances of the object from the viewer (z values). (See “Hidden Surface Removal” in the Graphics Library Programming Guide.)

SEE ALSO

zbuffer

NOTES

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 backface to do the opposite of what is intended.
On IRIS-4D B and G models backface does not work well when a polygon shrinks to the point where its vertices are coincident. Under these conditions, the routine cannot determine the orientation of the polygon and so displays the polygon by default.
NAME
bbox2, bbox2i, bbox2s – culls and prunes to bounding box and minimum pixel radius

C SPECIFICATION

void bbox2(xmin, ymin, x1, y1, x2, y2)
Screecoord xmin, ymin;
Coord x1, y1, x2, y2;
void bbox2i(xmin, ymin, x1, y1, x2, y2)
Screecoord xmin, ymin;
Icoord x1, y1, x2, y2;
void bbox2s(xmin, ymin, x1, y1, x2, y2)
Screecoord xmin, ymin;
Scoord x1, y1, x2, y2;

All of the above routines are functionally the same. They differ only in the declaration types of their parameters.

PARAMETERS

xmin expects the width, in pixels, of the smallest displayable feature.
ymin expects the height, in pixels, of the smallest displayable feature.
x1 expects the x coordinate of a corner of the bounding box.
y1 expects the y coordinate of a corner of the bounding box.
x2 expects the x coordinate of a corner of the bounding box. The corner referenced by this parameter must be diagonally opposite the corner referenced by the x1 and y1 parameters.
y2 expects the y coordinate of a corner of the bounding box. The corner referenced by this parameter must be diagonally opposite the corner referenced by the x1 and y1 parameters.

DESCRIPTION

bbox2 performs the graphical functions known as culling and pruning. Culling prevents the system from drawing objects that are less than the minimum feature size (xmin and ymin). Pruning prevents the system
from drawing objects that lie completely outside the viewport.

To determine whether or not to cull an object, \texttt{bbox2} tests whether or not the display of a rectangle the size of the bounding box is smaller than the minimum feature size. To determine whether or not to prune an object, \texttt{bbox2} tests whether or not the bounding box is completely outside the viewport.

Call \texttt{bbox2} within the definition for an object, just after the call to \texttt{makeobj}. If the object must be pruned or culled, the remainder of the object definition is ignored.

\section*{SEE ALSO}

\texttt{makeobj}

\section*{NOTES}

This routine does not function in immediate mode.

This routine is not a free test. If you use \texttt{bbox2} too freely, your performance can suffer. Reserve \texttt{bbox2} for complicated object definitions only.
NAME

gnclosedline, endclosedline – delimit the vertices of a closed line

C SPECIFICATION

void gnclosedline()
void endclosedline()

PARAMETERS

none

DESCRIPTION

gnclosedline marks the start of a group of vertex routines that you want interpreted as points on a closed line. Use endclosedline to mark the end of the vertex routines that are part of the closed line.

A closed line draws a line segment from one vertex on the list to the next vertex on the list. When the system reaches the end of the vertex list, it draws a line that connects the last vertex to the first vertex. All segments use the current linestyle, which is reset prior to the first segment and continues through subsequent segments. To specify a vertex, use the v routine.

Between gnclosedline and endclosedline, you can issue only the following Graphics Library routines: c, color, cpack, lmbind, lmcolor, lndef, n, RGBcolor, t, and v. Within a closed line, you should use lndef and lmbind only to respecify materials and their properties. If the color changes between a pair of vertices, the color of the line segment will be constant if the current shading model is FLAT and interpolated if the current shading model is GOURAUD. In color map mode, the colors vary through the color map; to get reasonable results, the color map should contain a ramp.

There is no limit to the number of vertices that can be specified between gnclosedline and endclosedline. After endclosedline, the system draws a line from the final vertex back to the initial vertex, and the current graphics position is left undefined.
By default line vertices are forced to the nearest pixel center prior to scan conversion. Line accuracy is improved when this coercion is defeated with the `subpixel` command. Subpixel vertex positioning is especially important when lines are scan converted with antialiasing enabled (see `linesmooth`).

`begnclosedline/endclosedline` are the same as `begnline/endline`, except they connect the last vertex to the first.

**EXAMPLE**

The code fragment below draws the outline of a triangle. Lines use the current linestyle, which is reset prior to the first vertex and continues through all subsequent vertices.

```c
begnclosedline();
v3f(vert1);
v3f(vert2);
v3f(vert3);
endclosedline();
```

**SEE ALSO**

`begnline`, `c`, `linesmooth`, `linewidth`, `lsrepeat`, `scrsdivide`, `setlinestyle`, `shademodel`, `subpixel`, `v`

**BUGS**

On the IRIS-4D B and G models, and on the Personal Iris without Turbo Graphics, if the color changes between a pair of vertices, the color of the line segment will be constant regardless of the current shading model.

On the IRIS-4D GT and GTX models, if the color changes between a pair of vertices, the color of the line segment will be interpolated regardless of the current shading model.
NAME

bgnline, endline – delimit the vertices of a line

C SPECIFICATION

void bgnline()
void endline()

PARAMETERS

none

DESCRIPTION

Vertices specified after bgnline and before endline are interpreted as endpoints of a series of line segments. Use the v routine to specify a vertex. The first vertex connects to the second; the second connects to the third; and so on until the next-to-last vertex connects to the last one. The last vertex does not connect to the first vertex. Use bgnclosedline to connect the first and last points. All segments use the current linestyle, which is reset prior to the first segment and continues through subsequent segments.

Between bgnline and endline, you can issue only the following Graphics Library routines: c, color, cpack, lmind, lmcolor, lmdef, n, RGBcolor, t, and v. lmdef and lmind can be used to respecify only materials and their properties. If the color changes between a pair of vertices, the color of the line segment will be constant if the current shading model is FLAT and interpolated if the current shading model is GOURAUD. In color map mode, the colors vary through the color map; to get reasonable results, the color map should contain a ramp.

There is no limit to the number of vertices that can be specified between bgnline and endline. After endline, the current graphics position is undefined.

By default line vertices are forced to the nearest pixel center prior to scan conversion. Line accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when lines are scan converted with antialiasing enabled (see linesmooth).
SEE ALSO

bgnclsdline, c, linesmooth, linewidth, lsrepeat, scrssubdivide, setlinestyle, shademodel, subpixel, v

BUGS

On the IRIS-4D B and G models, and on the Personal Iris without Turbo Graphics, if the color changes between a pair of vertices, the color of the line segment will be constant regardless of the current shading model.

On the IRIS-4D GT and GTX models, if the color changes between a pair of vertices, the color of the line segment will be interpolated regardless of the current shading model.
NAME

**bgnpoint, endpoint** – delimit the interpretation of vertex routines as points

C SPECIFICATION

```c
void bgnpoint()
void endpoint()
```

PARAMETERS

*none*

DESCRIPTION

**bgnpoint** marks the beginning of a list of vertex routines that you want interpreted as points. Use the **endpoint** routine to mark the end of the list. For each vertex, the system draws a one-pixel point into the frame buffer. Use the **v** routine to specify a vertex.

Between **bgnpoint** and **endpoint**, you can issue only the following Graphics Library routines: **c**, **color**, **cpack**, **Imbind**, **Imcolor**, **lmdef**, **n**, **RGBcolor**, **t**, and **v**. Use **lmdef** and **Imbind** to respecify only materials and their properties.

There is no limit to the number of vertices that can be specified between **bgnpoint** and **endpoint**.

By default points are forced to the nearest pixel center prior to scan conversion. This coercion is defeated with the **subpixel** command. Subpixel point positioning is important only when points are scan converted with antialiasing enabled (see **pntsmooth**).

After **endpoint**, the current graphics position is the most recent vertex.

SEE ALSO

**c**, **pntsmooth**, **subpixel**, **v**
NAME

bgnpolygon, endpolygon – delimit the vertices of a polygon

C SPECIFICATION

void bgnpolygon()
void endpolygon()

PARAMETERS

none

DESCRIPTION

Vertices specified after bgnpolygon and before endpolygon form a single polygon. The polygon can have no more than 256 vertices. Use the v subroutine to specify a vertex. Self-intersecting polygons (other than four-point bowties) may render incorrectly. Likewise, concave polygons may not render correctly if you have not called concave(TRUE).

Between bgnpolygon and endpolygon, you can issue only the following Graphics Library subroutines: c, color, cpack, Imbind, Imcolor, Imdef, n, RGBcolor, t, and v. Use Imdef and Imbind to respecify only materials and their properties.

By default polygon vertices are forced to the nearest pixel center prior to scan conversion. Polygon accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when polygons are scan converted with antialiasing enabled (see polysmooth).

After endpolygon, the current graphics position is undefined.

SEE ALSO

backface, c, concave, frontface, polymode, polysmooth, scrsubdivide, setpattern, shademodel, subpixel, v
NOTES

If you want to use the backface or frontface routines, specify the vertices in counter-clockwise order.

Although calling concave(TRUE) will guarantee that all polygons will be drawn correctly, on the IRIS-4D B and G models, and on the Personal Iris, doing so cause their performance to be degraded.
NAME

bgnqstrip, endqstrip – delimit the vertices of a quadrilateral strip

C SPECIFICATION

void bgnqstrip()
void endqstrip()

DESCRIPTION

Vertices specified between bgnqstrip and endqstrip are used to define a strip of quadrilaterals. The graphics pipe maintains three vertex registers. The first, second, and third vertices are loaded into the registers, but no quadrilateral is drawn until the system executes the fourth vertex routine. Upon executing the fourth vertex routine, the system draws a quadrilateral through the vertices, then replaces the two oldest vertices with the third and fourth vertices.

For each new pair of vertex routines, the system draws a quadrilateral through two new vertices and the two older stored vertices, then replaces the older stored vertices with the two new vertices.

Between bgnqstrip and endqstrip you can issue the following Graphics Library routines: c, color, cpack, lmbind, lmcolor, lmdef, n, RGBcolor, t, and v. Use lmdef and lmbind only to respecify materials and their properties.

If you want to use backface, you should specify the vertices of the first quadrilateral in counter-clockwise order. All quadrilaterals in the strip have the same rotation as the first quadrilateral in a strip, so that back-facing works correctly.

There is no limit to the number of vertices that can be specified between bgnqstrip and endqstrip. The result is undefined, however, if an odd number of vertices are specified, or if fewer than four vertices are specified.

By default quadrilateral vertices are forced to the nearest pixel center prior to scan conversion. Quadrilateral accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when quadrilaterals are scan converted with antialiasing enabled (see polysmooth).
After **endqstrip**, the current graphics position is undefined.

**EXAMPLE**

For example, the code sequence:

```c
bgnqstrip();
v3f(zero);
v3f(one);
v3f(two);
v3f(three);
v3f(four);
v3f(five);
v3f(six);
v3f(seven);
endqstrip();
```

draws three quadrilaterals: (0,1,2,3), (2,3,4,5), and (4,5,6,7). Note that the vertex order required by quadrilateral strips matches the order required by the equivalent triangle mesh. The vertices above, when places between **bgnmesh** and **endmesh** calls, draws six triangles: (0,1,2), (1,2,3), (2,3,4), (3,4,5), (4,5,6), and (5,6,7).

**SEE ALSO**

backface, c, concave, frontface, polymode, polysmooth, scrsdivide, setpattern, shademodel, subpixel, v

**NOTE**

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support quadrilateral strips. Use **getgdesc** to determine whether quadrilateral strips are supported.

IRIS-4D VGX models use vertex normals to improve the shading quality of quadrilaterals, regardless of whether lighting is enabled.
NAME

bgsurface, endsurface – delimit a NURBS surface definition

C SPECIFICATION

void bgsurface()
void endsurface()

PARAMETERS

none

DESCRIPTION

Use bgsurface to mark the beginning of a NURBS (Non-Uniform Rational B-Spline) surface definition. After you call bgsurface, call the routines that define the surface and that provide the trimming information. To mark the end of a NURBS surface definition, call endsurface.

Within a NURBS surface definition (between bgsurface and endsurface) you may use only the following Graphics Library subroutines: nurbssurface, bgntrim, endtrim, nurbbscurve, and pwllcurve. The NURBS surface definition must consist of exactly one call to nurbssurface to define the shape of the surface. In addition, this call may be preceded by calls to nurbssurface that specify how texture and color parameters vary across the surface. The call(s) to nurbssurface may be followed by a list of one or more trimming loop definitions (to define the boundaries of the surface). Each trimming loop definition consists of one call to bgntrim, one or more calls to either pwllcurve or nurbbscurve, and one call to endtrim.

The system renders a NURBS surface as a polygonal mesh, and calculates normal vectors at the corners of the polygons within the mesh. Therefore, your program should specify a lighting model if it uses NURBS surfaces. If your program uses no lighting model, all the interesting surface information is lost. When using a lighting model, use Imdef and Imbind to define or modify materials and their properties.
EXAMPLE

The following code fragment draws a NURBS surface trimmed by two closed loops. The first closed loop is a single piecewise linear curve (see `pwlcurve`), and the second closed loop consists of two NURBS curves (see `nurbscurve`) joined end to end:

```c
bgnsurf;
  nurbsSurface(. . .);
  bgntrim();
    pwlcurve(. . .);
  endtrim();
  bgntrim();
    nurbsCurve(. . .);
    nurbsCurve(. . .);
  endtrim();
endsurf;
```

SEE ALSO

`nurbsSurface`, `bgntrim`, `nurbsCurve`, `pwlcurve`, `setnurbsproperty`, `getnurbsproperty`
NAME

*bgnmesh, endtmesh* – delimit the vertices of a triangle mesh

C SPECIFICATION

```c
void bgnmesh()
void endtmesh()
```

PARAMETERS

*none*

DESCRIPTION

Vertices specified between *bgnmesh* and *endtmesh* are used to define a mesh of triangles. The graphics pipe maintains two vertex registers. The first and second vertices are loaded into the registers, but no triangle is drawn until the system executes the third vertex routine. Upon executing the third vertex routine, the system draws a triangle through the vertices, then replaces the older of the register vertices with the third vertex.

For each new vertex routine, the system draws a triangle through the new vertex and the stored vertices, then (by default) replaces the older stored vertex with the new vertex. If you want the system to replace the more recent of the stored vertices, call *swaptmesh* prior to calling *v*.

Between *bgnmesh* and *endtmesh* you can issue the following Graphics Library routines: *c, color, cpack, lmbind, lmcolor, lmdef, n, RGBcolor, swaptmesh, t, and v*. Use *lmdef* and *lmbind* only to respecify materials and their properties.

If you want to use *backface*, you should specify the vertices of the first triangle in counter-clockwise order. All triangles in the mesh have the same rotation as the first triangle in a mesh so that backfacing works correctly.

There is no limit to the number of vertices that can be specified between *bgnmesh* and *endtmesh*. 
By default triangle vertices are forced to the nearest pixel center prior to scan conversion. Triangle accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when triangles are scan converted with antialiasing enabled (see polysmooth).

After endtmesh, the current graphics position is undefined.

EXAMPLE

For example, the code sequence:

```c
bgntmesh();
v3f(zero);
v3f(one);
v3f(two);
v3f(three);
endtmesh();
```

draws two triangles, (zero,one,two) and (one,two,three), while the code sequence:

```c
bgntmesh();
v3f(zero);
v3f(one);
swaptmesh();
v3f(two);
v3f(three);
endtmesh();
```

draws two triangles, (zero,one,two) and (zero,two,three). There is no limit to the number of times that swaptmesh can be called.

SEE ALSO

backface, c, concave, frontface, polymode, polysmooth, scrsubdivide, setpattern, shademodel, subpixel, swaptmesh, v
NAME

`bgntrim, endtrim` - delimit a NURBS surface trimming loop

C SPECIFICATION

```c
void bgntrim()
void endtrim()
```

PARAMETERS

`none`

DESCRIPTION

Use `bgntrim` to mark the beginning of a definition for a trimming loop. Use `endtrim` to mark the end of a definition for a trimming loop. A trimming loop is a set of oriented curves (forming a closed curve) that defines boundaries of a NURBS surface. You include these trimming loop definitions in the definition of a NURBS surface.

The definition for a NURBS surface may contain many trimming loops. For example, if you wrote a definition for NURBS surface that resembled a rectangle with a hole punched out, the definition would contain two trimming loops. One loop would define the outer edge of the rectangle. The other trimming loop would define the hole punched out of the rectangle. The definitions of each of these trimming loops would be bracketed by a `bgntrim/endtrim` pair.

The definition of a single closed trimming loop may consist of multiple curve segments, each described as a piecewise linear curve (see `pwlcurve`) or as a single NURBS curve (see `nurbscurve`), or as a combination of both in any order. The only Graphics library calls that can appear in a trimming loop definition (between a call to `bgntrim` and a call to `endtrim`) are `pwlcurve` and `nurbscurve`.

In the following code fragment, we define a single trimming loop that consists of one piecewise linear curve and two NURBS curves:
bgntrim();
pwlcurve(. . .);
nurbscurve(. . .);
nurbscurve(. . .);
endtrim();

The area of the NURBS surface that the system displays is the region in the domain to the left of the trimming curve as the curve parameter increases. Thus, the resultant visible region of the NURBS surface is inside for a counter-clockwise trimming loop and outside for a clockwise trimming loop. So for the rectangle mentioned earlier, the trimming loop for the outer edge of the rectangle should run counter-clockwise, and the trimming loop for the hole punched out should run clockwise.

If you use more than one curve to define a single trimming loop, the curve segments must form a closed loop (i.e., the endpoint of each curve must be the starting point of the next curve, and the endpoint of the final curve must be the starting point of the first curve). If the endpoints of the curve are sufficiently close together but not exactly coincident, the system coerces the them to match. If the endpoints are not sufficiently close, the system generates an error message and ignores the entire trimming loop.

If a trimming loop definition contains multiple curves, the direction of the curves must be consistent (i.e., the inside must be to the left of the curves). Nested trimming loops are legal as long as the curve orientations alternate correctly. If no trimming information is given for a NURBS surface, the entire surface is drawn.

SEE ALSO

bgnnsurface, nurbsssurface, nurbscurve, pwlcurve, setnurbsproperty, getnurbsproperty
NAME

blankscreen – controls screen blanking

C SPECIFICATION

void blankscreen(b)
Boolean b;

PARAMETERS

b expects TRUE or FALSE.
TRUE stops display and turns screen black.
FALSE restores the display.

DESCRIPTION

blankscreen turns screen refresh on and off. It affects the screen on which the current window is displayed.

NOTE

This routine is available only in immediate mode.

SEE ALSO

blanktime
NAME

blanktime — sets the screen blanking timeout

C SPECIFICATION

void blanktime(count)
long count;

PARAMETERS

count expects the number of graphics timer events after which to blank the current screen. The frequency of graphics timer events is returned by the getgdesc inquiry GD_TIMERHZ.

DESCRIPTION

By default, a screen blanks (turns black) after the system receives no input for 10 minutes. This protects the monitor. Use blanktime to change the amount of time the system waits before it blanks a screen. It affects the screen on which the current window is displayed.

To calculate the value of count, simply multiply the desired blanking latency period (in seconds) by getgdesc(GD_TIMERHZ).

You can disable screen blanking by calling this routine with a count of zero.

NOTE

This routine is available only in immediate mode.

SEE ALSO

blankscreen, getgdesc
NAME

blendfunction – computes a blended color value for a pixel

C SPECIFICATION

void blendfunction(sfactr, dfactr)
long sfactr, dfactr;

PARAMETERS

sfactr Expects a symbolic constant from the list below that identifies the blending factor by which to scale contribution from source pixel RGBA (red, green, blue, alpha) values. Blending factors use RGBA values converted to fractions of the maximum value 255. To improve performance, conversion calculations are approximate. However, 0 converts exactly to 0.0, and 255 converts exactly to 1.0.

| BF_ZERO      | 0         |
| BF_ONE       | 1         |
| BF_DC        | (destination RGBA)/255 |
| BF_MDC       | 1 – (destination RGBA)/255 |
| BF_SA        | (source alpha)/255 |
| BF_MSA       | 1 – (source alpha)/255 |
| BF_DA        | (destination alpha)/255 |
| BF_MDA       | 1 – (destination alpha)/255 |
| BF_MIN_SA_MDA| min(BF_SA, BF_MDA) |

dfactr Expects a symbolic constant from the list below that identifies the blending factor by which to scale contribution from destination pixel RGBA values.

| BF_ZERO      | 0         |
| BF_ONE       | 1         |
| BF_SC        | (source RGBA)/255 |
| BF_MSC       | 1 – (source RGBA)/255 |
| BF_SA        | (source alpha)/255 |
| BF_MSA       | 1 – (source alpha)/255 |
BF_DA
BF_MDA

(destination alpha)/255
1 – (destination alpha)/255

DESCRIPTION

In RGB mode, the system draws pixels using a function that blends the incoming (source) RGBA values with the RGBA values that are already in the framebuffer (the destination values). Most often, blending is simple: the source RGBA values replace the destination RGBA values of the pixel.

In some cases, however, simple replacement of framebuffer values is not appropriate. Two such cases are transparency and antialiasing. To be blended properly, transparent objects must be rendered back-to-front (i.e. drawn in order from the farthest object to the nearest object) with a blend function of (BF_SA, BF_MSA). As can be seen from the equations below, this function scales the incoming color components by the incoming alpha value, and scales the framebuffer contents by one minus the incoming alpha value. Thus incoming (source) alpha is correctly thought of as a material opacity, ranging from 1.0 (completely opaque) to 0.0 (completely transparent). Note that this transparency calculation does not require the presence of alpha bitplanes in the framebuffer.

Suggestions for appropriate blend functions for antialiasing are given on the pntsmooth and linesmooth manual pages. Other less obvious applications are also possible. For example, if the red component in the framebuffer is first cleared to all zeros, and then each primitive is drawn with red set to 1 and a blend function of (BF_ONE, BF_ONE), the red component of each pixel in the framebuffer will contain the count of the number of times that pixel was drawn.

To determine the blended RGBA values of a pixel when drawing in RGB mode, the system uses the following functions:

\[
R_{\text{destination}} = \min(255, ((R_{\text{source}} \times sfactr) + (R_{\text{destination}} \times dfactr)))
\]
\[
G_{\text{destination}} = \min(255, ((G_{\text{source}} \times sfactr) + (G_{\text{destination}} \times dfactr)))
\]
\[
B_{\text{destination}} = \min(255, ((B_{\text{source}} \times sfactr) + (B_{\text{destination}} \times dfactr)))
\]
\[
A_{\text{destination}} = \min(255, ((A_{\text{source}} \times sfactr) + (A_{\text{destination}} \times dfactr)))
\]
When the blend function is set to \((\text{BF\_ONE}, \text{BF\_ZERO})\), the default values, the equations reduce to simple replacement:

\[
\begin{align*}
R_{\text{destination}} &= R_{\text{source}} \\
G_{\text{destination}} &= G_{\text{source}} \\
B_{\text{destination}} &= B_{\text{source}} \\
A_{\text{destination}} &= A_{\text{source}}
\end{align*}
\]

Fill rate may be increased substantially when blending is disabled in this manner.

Polygon antialiasing (see \texttt{polysmooth}) is sometimes optimized when the blendfunction \((\text{BF\_MIN\_SA\_MDA}, \text{BF\_ONE})\) is used. Source factor \texttt{BF\_MIN\_SA\_MDA}, which should be used only with destination factor \texttt{BF\_ONE}, has the side effect of slightly modifying the blending arithmetic:

\[
\begin{align*}
R_{\text{destination}} &= \min (255, \ ((R_{\text{source}} \times s\text{factr}) + R_{\text{destination}})) \\
G_{\text{destination}} &= \min (255, \ ((G_{\text{source}} \times s\text{factr}) + G_{\text{destination}})) \\
B_{\text{destination}} &= \min (255, \ ((B_{\text{source}} \times s\text{factr}) + B_{\text{destination}})) \\
A_{\text{destination}} &= s\text{factr} + A_{\text{destination}}
\end{align*}
\]

This special blend function accumulates pixel contributions until the pixel is fully specified, then allows no further changes. Destination alpha bitplanes, which must be present for this blend function to operate correctly, store the accumulated coverage.

It is intended that the \texttt{destination} values on the left and the right of the above equations be the same framebuffer locations. However, when multiple destination buffers are specified (using \texttt{frontbuffer}, \texttt{backbuffer}, and \texttt{zdraw}) only a single location can be read and used on the right side of the equation. By default, the destination RGBA values are read from the front buffer in single buffer mode and from the back buffer in double buffer mode. If the front buffer is not enabled in single buffer mode, the RGBA values are taken from the z-buffer. If the back buffer is not enabled in double buffer mode, the RGBA values are taken from the front buffer (if possible) or from the z-buffer.

Blending is available with or without z-buffer mode. When blendfunction is set to any value other than \((\text{BF\_ONE}, \text{BF\_ZERO})\), logiop is forced to \texttt{LO\_SRC}.
SEE ALSO

cpack, linesmooth, logicop, pntsmooth, polysmooth

NOTES

This subroutine is available only in immediate mode.

Blending factors BF_DA, BF_MDA, and BF_MIN_SA_MDA are not supported on machines without alpha bitplanes. Blend factor BF_MIN_SA_MDA is supported only on VGX graphics systems.

This subroutine does not function on IRIS-4D B or G models or on the Personal Iris. Use getgdesc(GDBLEND) to determine whether blending hardware is available.

BUGS

Blending works properly only in RGB mode. In color map mode, the results are unpredictable.

On some IRIS-4D GT and GTX models, while copying rectangles with blending active, readsource also specifies the bank from which destination color and alpha are read (overriding the blendfunction setting).

IRIS-4D VGX models do not clamp color values generated by the special blending function BF_MIN_SA_MDA,BF_ONE to 255. Instead, color values are allowed to wrap. This will be corrected in the next release.
NAME

blink – changes a color map entry at a selectable rate

C SPECIFICATION

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

PARAMETERS

- **rate** expects the number of vertical retraces per blink. On the standard monitor, there are 60 vertical retraces per second.
- **i** expects an index into the current color map. The color defined at that index is the color that is blinked (alternated).
- **red** expects the red value of the alternate color that blinks against the color selected from the color map by the **i** parameter.
- **green** expects the green value of the alternate color that blinks against the color selected from the color map by the **i** parameter.
- **blue** expects the blue value of the alternate color that blinks against the color selected from the color map by the **i** parameter.

DESCRIPTION

**blink** alternates the color located at index **i** in the current color map with the color defined by the parameters **red**, **green**, and **blue**. The rate at which the two colors are alternated is set by the **rate** parameter. The maximum number of color map entries that can be blinking simultaneously on a screen is returned by the **getgdesc** inquiry **GD_NBLINKS**.

The length of time between retraces varies according to the monitor used. On the standard monitor, there are 60 retraces per second, so a **rate** of 60 would cause the color to change once every second.

To terminate blinking and restore the original color for a single color map entry, call **blink** for that entry with **rate** set to 0.
To terminate all blinking colors simultaneously, call `blink` with `rate` set to $-1$. When `rate` is $-1$, the other parameters are ignored.

SEE ALSO

getgdesc, mapcolor

NOTE

This routine is available only in immediate mode.
NAME

`blkqread` – reads multiple entries from the queue

C SPECIFICATION

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

PARAMETERS

- `data` expects a pointer to the buffer that is to receive the queue information.
- `n` expects the number of elements in the buffer.

FUNCTION RETURN VALUE

The returned value of the function is the number of 16 bit words of data actually read into the `data` buffer. Note that this number will be twice the number of complete queue entries read, because each queue entry consists of two 16 bit words.

DESCRIPTION

`blkqread` reads multiple entries from the input queue and stores them in the array pointed to by `data`. This function fills the `data` buffer with paired values (a device number and the value of that device).

SEE ALSO

`qread`

NOTE

This routine is available only in immediate mode.
NAME

c3f, c3i, c3s, c4f, c4i, c4s – sets the RGB (or RGBA) values for the current color vector

C SPECIFICATION

void c3s(cv)
short cv[3];

void c3i(cv)
long cv[3];

void c3f(cv)
float cv[3];

void c4s(cv)
short cv[4];

void c4i(cv)
long cv[4];

void c4f(cv)
float cv[4];

The subroutines above are functionally the same but declare their parameters differently.

PARAMETER

**cv** For the c4 routines, this parameter expects a four element array containing RGBA (red, green, blue, and alpha) values. If you use the c3 routines, this parameter expects a three element array containing RGB values.

Array components 0, 1, 2, and 3 are red, green, blue, and alpha, respectively. Floating point RGBA values range from 0.0 through 1.0. Integer RGBA values range from 0 through 255. Values that exceed the upper limit are clamped to it. Values that exceed the lower limit are not clamped, and therefore result in unpredictable operation.
DESCRIPTION

c4 sets the red, green, blue, and alpha color components of the currently active GL framebuffer, one of normal, popup, overlay, or underlay (see drawmode). c3 sets red, green, and blue to the specified values, and sets alpha to the maximum value. The current framebuffer must be in RGB mode (see RGBmode) for the c command to be applicable. Most drawing commands copy the current RGBA color components into the color bitplanes of the current framebuffer. Color components are retained in each draw mode, so when a draw mode is re-entered, red, green, blue, and alpha are reset to the last values specified in that draw mode.

Integer color component values range from 0, specifying no intensity, through 255, specifying maximum intensity. Floating point color component values range from 0.0, specifying no intensity, through 1.0, specifying maximum intensity.

It is an error to call c while the current framebuffer is in color map mode.

The color components of all framebuffers in RGB mode are set to zero when gconfig is called.

SEE ALSO
cpack, drawmode, Imcolor, gRGBcolor

NOTE

These routines can also be used to modify the current material while lighting is active (see Imcolor). Note that clamping to 1.0 is disabled in this case.

Because only the normal framebuffer currently supports RGB mode, c should be called only while draw mode is NORMALDRAW. Use getgdesc to determine whether RGB mode is available in draw mode NORMALDRAW.
NAME

callfunc – calls a function from within an object

C SPECIFICATION

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

PARAMETERS

fctn     expects a pointer to a function.
nargs    expects the number of arguments, excluding itself, that the
         function pointed to by fctn is to be called with.

arg1, arg2, ..., argn
         expect the arguments to the function pointed to by fctn.

DESCRIPTION

callfunc is used to call an arbitrary function from within an object.
When callfunc executes in the object, the function call
(*fctn)(nargs, arg1, arg2, ..., argn) is made.

NOTE

This routine does not function in the Distributed Graphics Library
(DGL), and we advise against its use in new development.
NAME

callobj – draws an instance of an object

C SPECIFICATION

void callobj(obj)
Object obj;

PARAMETERS

obj expects the object identifier of the object that you want to draw.

DESCRIPTION

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

Global state attributes are not saved before a call to callobj. Thus, 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.

Likewise, the object may execute transformations that change the matrix stack, so you may want to use pushmatrix and popmatrix to restore the state of the matrix stack.

SEE ALSO

makeobj, popattributes, pushattributes, pushmatrix, popmatrix
NAME

`charstr` — draws a string of raster characters on the screen

C SPECIFICATION

```c
void charstr(str)
String str;
```

PARAMETERS

`str` expects a pointer to the string you want to draw.

DESCRIPTION

`charstr` draws a string of text 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 width is added to 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.

SEE ALSO

`cmov`, `defrasterfont`, `font`, `strwidth`
NAME

chunksize – specifies minimum object size in memory

C SPECIFICATION

void chunksize(chunk)
long chunk;

PARAMETERS

chunk Expect the minimum memory size to allocate for an object. As
you add objects to a display list, chunk is the unit size (in bytes)
by which the memory allocated to the display list grows.

DESCRIPTION

chunksize specifies the minimum object memory size. You can call it
only once after graphics initialization and before the first makeobj.

If you do not use this function, the system assumes a chunk size of 1020
bytes. This is usually more than large enough. Therefore, you generally
need to use chunksize only if your application is running up against the
memory limits, and you know that 1020 bytes per object is too much.

But be careful, if chunksize is set too small, complex objects (e.g.,
multi-sided polygons) will not display. Each object in a display list
must fit entirely into a single chunk. Some experimentation may be
necessary to determine the optimal chunksize for an application.

SEE ALSO

compactify, makeobj

NOTE

This routine is available only in immediate mode.
NAME

circ, circi, circs – outlines a circle

C SPECIFICATION

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

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

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

The routines above are functionally the same. However, the type declarations for the coordinates differ.

PARAMETERS

x expects the x coordinate of the center of the circle specified in world coordinates.

y expects the y coordinate of the center of the circle specified in world coordinates.

radius expects the length of the radius of the circle.

DESCRIPTION

circ draws an unfilled circle in the x-y plane with z assumed to be zero. To create a circle that does not lie in the x-y plane, draw the circle in the x-y plane, then rotate and/or translate the circle. Note that circles rotated outside the 2-D x-y plane appear as ellipses.

A circle is drawn as a sequence of line segments, and therefore inherits all properties that affect the drawing of lines. These include the current color, writemask, line width, stipple pattern, shade model, line antialiasing mode, and subpixel mode. The stipple pattern is initialized to bit zero of the current linestyle before the circle is drawn, then shifted continuously through the segments of the circle.
After `circ` executes, the graphics position is undefined.

**SEE ALSO**

arc, bgnclosedline, circf, crvn, linewidth, linesmooth, lsrepeat, scrsubdivide, setlinestyle, shademodel, subpixel

**BUGS**

When the line width is greater than 1, small notches will appear in circles, because of the way wide lines are implemented.
NAME

circf, circfi, circfs – draws a filled circle

C SPECIFICATION

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

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

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

The routines above are functionally the same even though the type
declarations for the coordinates differ.

PARAMETERS

x        expects the x coordinate of the center of the filled circle
         specified in world coordinates.

y        expects the y coordinate of the center of the filled circle
         specified in world coordinates.

radius   expects the length of the radius of the filled circle.

DESCRIPTION

circf draws a filled circle in the x-y plane \((z = 0)\). To draw a circle in a
plane other than the x-y plane, define the circle in the x-y plane and then
rotate or translate the circle. Note that filled circles rotated outside the
2-D x-y plane appear as filled ellipses.

A circle is drawn as a single polygon, and therefore inherits all prop-
ties that affect the drawing of polygons. These include the current color,
writemask, fill pattern, shade model, polygon antialiasing mode, polygon
scan conversion mode, and subpixel mode. Front-face and back-face elimina-
tion work correctly with filled circles, which are front-facing when viewed from the positive z half-space.
After `circf` executes, the graphics position is undefined.

SEE ALSO

`arcf`, `backface`, `bgnpolygon`, `circ`, `frontface`, `polymode`, `polysmooth`, `scrsdivide`, `setpattern`, `shademodel`, `subpixel`
NAME

clear – clears the viewport

C SPECIFICATION

void clear()

PARAMETERS

none

DESCRIPTION

clear sets the bitplane area of the viewport to the current color. Multiple bitplane buffers can be cleared simultaneously using the backbuffer, frontbuffer, and zdraw commands. Current polygon fill pattern and writemask affect the operation of clear. The screen mask, when it is set to a subregion of the viewport, bounds the cleared region. Alpha function, blend function, logical operation, stenciling, texture mapping, and z buffering, however, are ignored by clear. Stencil and z buffer contents are not affected by clear (except in the special case of zdraw).

Like other drawing commands, clear operates on the currently active framebuffer, one of normal, popup, overlay, or underlay, based on the current draw mode (see drawmode).

After clear executes, the graphics position is undefined.

SEE ALSO

afunction, backbuffer, blendfunction, czclear, drawmode, frontbuffer, logicop, scrmask, setpattern, stencil, texbind, zbuffer, zdraw

NOTE

On the IRIS-4D B, G, GT, GTX, and VGX models, clear runs faster when the window is completely unobscured.

On the Personal Iris, clear runs faster when the visible window area consists of four or fewer rectangular regions.
NAME

clearhitcode — sets the hitcode to zero

C SPECIFICATION

void clearhitcode()

PARAMETERS

none

DESCRIPTION

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

SEE ALSO

gethitcode, gselect, pick

NOTES

This routine is available only in immediate mode.

This routine only functions on IRIS-4D B and G models, and therefore we advise against its use in new development.
NAME

clipplane – specify a plane against which all geometry is clipped

C SPECIFICATION

void clipplane(index, mode, params)
long index, mode;
float params[];

PARAMETERS

index expects an integer in the range 0 through 5, indicating which of
the 6 clipping planes is being modified.

mode expects one of three tokens:

CP_DEFINE: use the plane equation passed in params to
define a clipplane. The clipplane is neither enabled nor dis-
abled.

CP_ON: enable the (previously defined) clipplane.

CP_OFF: disable the clipplane. (default)

params expects an array of 4 floats that specify a plane equation. A
plane equation is usually thought of as a 4-vector [A,B,C,D].
In this case, A is the first component of the params array, and
D is the last. A 4-component vertex array (see v4f) can be
passed as a plane equation, where vertex X becomes A, Y
becomes B, etc.

DESCRIPTION

Geometry is always clipped against the boundaries of a 6-plane frustum
in x, y, and z. clipplane allows the specification of additional planes, not
necessarily perpendicular to the x, y, or z axes, against which all
geometry is clipped. Up to 6 additional planes can be specified.
Because the resulting clipping region is always the intersection of the
(up to) 12 half-spaces, it is always convex.
**clipplane** specifies a half-space using a 4-component plane equation. When it is called with mode CP_DEFINE, this object-coordinate plane equation is transformed to eye-coordinates using the inverse of the current ModelView matrix.

A defined clipplane is then enabled by calling `clipplane` with the CP_ON argument, and with arbitrary values passed in `params`. While drawing after a clipplane has been defined and enabled, each vertex is transformed to eye-coordinates, where it is dotted with the transformed clipping plane equation. Eye-coordinate vertexes whose dot product with the transformed clipping plane equation is positive or zero are in, and require no clipping. Those eye-coordinate vertexes whose dot product is negative are clipped. Because `clipplane` clipping is done in eye-coordinates, changes to the projection matrix have no effect on its operation.

By default all six clipping planes are undefined and disabled. The behavior of an enabled but undefined clipplane is undefined.

**NOTES**

IRIS-4D models G, GT, and GTX, and the Personal Iris, do not implement `clipplane`. Use `getgdesc` to determine whether user-defined clipping planes are supported.

`clipplane` cannot be used while `mmode` is MSINGLE.

A point and a normal are converted to a plane equation in the following manner:
point = [Px,Py,Pz]

normal = [Nx]
         [Ny]
         [Nz]

plane equation = |A|
                |B|
                |C|
                |D|

A = Nx
B = Ny
C = Nz
D = -[Px,Py,Pz] dot [Nx]
         |Ny|
         |Nz|
NAME

clkon, clkoff – control keyboard click

C SPECIFICATION

void clkon()
void clkoff()

PARAMETERS

none

DESCRIPTION

clkon and clkoff control the keyboard click.

SEE ALSO

lampon, ringbell, setbell

NOTE

This routine is available only in immediate mode.
NAME

closeobj – closes an object definition

C SPECIFICATION

void closeobj();

PARAMETERS

none

DESCRIPTION

closeobj closes an open object definition. Use makeobj to open a
definition for 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 ses-

If no object is open, closeobj is ignored.

SEE ALSO

editobj, makeobj

NOTE

This routine is available only in immediate mode.
NAME

cmode – sets color map mode as the current mode.

C SPECIFICATION

void cmode()

PARAMETERS

none

DESCRIPTION

cmode instructs the system to treat color as a 1-component entity in the currently active drawmode. The single color component is used as an index into a table of RGB color values called the color map. Because color map mode is the default value for all GL framebuffers, it can be called in any of the framebuffer drawmodes (NORMALDRAW, PUP-DRAW, OVERDRAW, and UNDERDRAW). To return the normal framebuffer to color map mode, however, you must call cmode while in drawmode NORMALDRAW. You must call gc locals for cmode to take effect.

While in color map mode, a framebuffer is configured to store a single color index at each pixel location. The framebuffer is displayed by continually translating color indices into RGB triples using the framebuffer’s color map, a table of index-to-RGB mappings. The red, green, and blue components stored in the color map are used (after correction for monitor non-linearity) to directly control the color guns of the monitor. Colors and writemasks must be specified using color map-compatible commands such as color, colorf, and writemask.

Many advanced rendering features, such as texture mapping, polygon antialiasing, and fog, are available only in RGB mode. Color map mode lighting, while functional, is substantially less robust than its RGB mode counterpart.

Since cmode is the default, you do not have to call it unless the normal framebuffer was previously set to RGB mode.
SEE ALSO

color, drawmode, gconfig, getdisplaymode, getgdesc, multimap, onemap, RGBmode, writemask

NOTE

Color map mode is available in all framebuffers of all hardware configurations. getgdesc can be used to determine how many bitplanes in each of the normal, popup, overlay, and underlay framebuffers are available in both single and double buffered color map mode.

This routine is available only in immediate mode.
NAME

cmov, cmovi, cmovs, cmov2, cmov2i, cmov2s – updates the current character position

C SPECIFICATION

void cmov(x, y, z)
Coord x, y, z;
void cmovi(x, y, z)
Icoord x, y, z;
void cmovs(x, y, z)
Scoord x, y, z;
void cmov2(x, y)
Coord x, y;
void cmov2i(x, y)
Icoord x, y;
void cmov2s(x, y)
Scoord x, y;

All of the above functions are functionally the same except for the type declarations of the parameters. In addition the cmov2* routines assume a 2-D point instead of a 3-D point.

PARAMETERS

x expects the x location of the point (in world coordinates) to which you want to move the current character position.

y expects the y location of the point (in world coordinates) to which you want to move the current character position.

z expects the z location of the point (in world coordinates) to which you want to move the current character position. (This parameter not used by the 2-D subroutines.)

DESCRIPTION

cmov moves the current character position to a specified point (just as move sets the current graphics position). 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
NAME

color, colorf – sets the color index in the current draw mode

C SPECIFICATION

void color(c)
Colorindex c;

void colorf(c)
float c;

PARAMETERS

c expects an index into the current color map.

DESCRIPTION

color sets the color index of the currently active GL framebuffer, one of normal, popup, overlay, or underlay (see drawmode). The current framebuffer must be in color map mode (see cmode) for the color command to be applicable. Most drawing commands copy the current color index into the color bitplanes of the current framebuffer. color is retained in each draw mode, so when a draw mode is re-entered, color is reset to the last value specified in that draw mode.

color values range from 0 through $2^n-1$, where $n$ is the number of bitplanes available in the current draw mode. $n$ can be ascertained by calling getplanes while in the desired draw mode, or by calling getgdesc at any time. Color indices larger than $2^n-1$ are clamped to $2^n-1$; color indices less than zero yield undefined results.

The color displayed by a given color index is determined by the current color map (see mapcolor.) Each draw mode has its own color map.

colorf is identical to color, except that it expects a floating point color index. Before the color is written into display memory, it is rounded to the nearest integer value. When drawing with the GOURAUD shading model, machines that iterate color indices with fractional precision yield more precise shading results using colorf than with color. The results of color and colorf are indistinguishable when drawing with FLAT shading.
It is an error to call `color` or `colorf` while the current framebuffer is in RGB mode.

The color indices of all framebuffers in color map mode are set to zero when `geonfig` is called.

**SEE ALSO**

drawmode, getcolor, mapcolor, writemask

**NOTE**

IRIS-4D B, G, GT, and GTX models do not iterate color with fractional precision, nor do early serial numbers of the Personal Iris. Use `getgdesc(GD_CIFRACT)` to determine whether fractional color index iteration is supported.
NAME

compactify – compacts the memory storage of an object

C SPECIFICATION

void compactify(obj)
Object obj;

PARAMETERS

obj expects the object identifier for the object you want to compact.

DESCRIPTION

When you modify an open object definition (using the object editing routines), the memory storage for the object definition can become fragmented. A call to compactify can make a fragmented object definition occupy a continuous section of memory.

Although you can call compactify to explicitly compact an object, it is rarely necessary because a call to closeobj automatically calls compactify, when the object definition becomes too fragmented. (After you edit an object, you must always call closeobj.)

Because compactify, requires a significant amount of time, do not call it unless storage space is critical and you cannot tolerate even the small amount of fragmentation allowed by closeobj.

SEE ALSO

closeobj, chunksize

NOTE

This routine is available only in immediate mode.
NAME

concave — allows the system to draw concave polygons

C SPECIFICATION

void concave(b)
Boolean b;

PARAMETERS

b expects either TRUE or FALSE.

TRUE tells the system to expect concave polygons.

FALSE tells the system to expect no concave polygons. This is the default.

DESCRIPTION

concave tells the system whether or not to expect concave polygons. If you try to draw a concave polygon while the system does not expect it, the results are unpredictable. Although calling concave(TRUE) guarantees that all non-selfintersecting polygons will be drawn correctly, the performance of non-concave polygons is reduced on some machines. Polygons whose edges intersect each other are never guaranteed to be drawn correctly.

In all cases, performance is optimized when concave polygons are decomposed into convex pieces before being passed to a GL drawing routine.

SEE ALSO

bgnpolygon

BUG

IRIS-4D GT and GTX models always expect concave polygons, regardless of the value of the concave flag.
NAME

cpack – specifies RGBA color with a single packed 32-bit integer

C SPECIFICATION

void cpack(pack)
unsigned long pack;

PARAMETERS

pack expects a packed integer containing the RGBA (red, green, blue, alpha) values you want to assign as the current color. Expressed in hexadecimal, the format of the packed integer is 0xaabbggrr, where:

\begin{align*}
aa & \text{ is the alpha value,} \\
bb & \text{ is the blue value,} \\
gg & \text{ is the green value, and} \\
rr & \text{ is the red value.} \\
\end{align*}

RGBA component values range from 0 to 0xFF (255).

DESCRIPTION

cpack sets the red, green, blue, and alpha color components of the currently active GL framebuffer, one of normal, popup, overlay, or underlay (see drawmode). The current framebuffer must be in RGB mode (see RGBmode) for the cpack command to be applicable. Most drawing commands copy the current RGBA color components into the color bitplanes of the current framebuffer. Color components are retained in each draw mode, so when a draw mode is re-entered, red, green, blue, and alpha are reset to the last value specified in that draw mode.

Color component values range from 0, specifying no intensity, through 255, specifying maximum intensity. For example, cpack(0xFF004080) sets red to 0x80 (half intensity), green to 0x40 (quarter intensity), blue to 0 (off), and alpha to 0xFF (full intensity).
It is an error to call \texttt{cpack} while the current framebuffer is in color map mode.

The color components of all framebuffers in RGB mode are set to zero when \texttt{gconfig} is called.

\textbf{SEE ALSO}

\texttt{c, drawmode, gRGBcolor, lmcolor}

\textbf{NOTE}

\texttt{cpack} can also be used to modify the current material while lighting is active (see \texttt{lmcolor}).

Because only the normal framebuffer currently supports RGB mode, \texttt{cpack} should be called only while draw mode is \texttt{NORMALDRAW}. Use \texttt{getgdesc} to determine whether RGB mode is available in draw mode \texttt{NORMALDRAW}.
NAME

`crv` – draws a curve

C SPECIFICATION

```c
void crv(points)
Coord points[4][3];
```

PARAMETERS

`points` expects an array containing the four points that define the curve. The routine expects 3-D points (x, y, and z coordinates for each point).

DESCRIPTION

`crv` draws a cubic spline curve segment (defined by the four submitted points) according to the current curve basis and precision.

The curve segment is approximated by a sequence of straight lines. All lines use the current linestyle, which is reset prior to the first line and continues through subsequent lines. Other line modes, including depthcuing, line width, and line antialiasing, also apply to the lines generated by `crv`.

After `crv` executes, the graphics position is undefined.

SEE ALSO

`crvn`, `curvebasis`, `curveprecision`, `defbasis`, `depthcue`, `linesmooth`, `linewidth`, `rcrv`, `rcrvn`, `setlinestyle`
NAME

crvn – draws a series of curve segments

C SPECIFICATION

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

PARAMETERS

geom expects a matrix of 3-D points.
n expects the number of points in the matrix referenced by geom.

DESCRIPTION

crvn draws a series of cubic spline segments using the current basis and precision. The control points determine the shapes of the curve segments and are used sequentially four at a time.

For example, if there are six control points, there are three possible sequential selections of four control points. Thus, crvn draws three curve segments: the first using control points 0,1,2,3; the second using control points 1,2,3,4; and the third using control points 2,3,4,5.

If the current basis is a B-spline, a Cardinal spline, or a basis with similar properties, the curve segments are joined end to end and appear as a single curve.

Each curve segment is approximated by a sequence of straight lines. All lines use the current linestyle, which is reset prior to the first line and continues through subsequent lines. Other line modes, including depthcueing, line width, and line antialiasing, also apply to the lines generated by crvn.

After crvn executes, the graphics position is undefined.

SEE ALSO

crv, curvebasis, curveprecision, defbasis, depthcue, linesmooth, linewidth, rcrv, rcrvn, setlinestyle
NAME

curorigin – sets the origin of a cursor

C SPECIFICATION

void curorigin(n, xorigin, yorigin)
short n, xorigin, yorigin;

PARAMETERS

n        expects an index into the cursor table created by defcursor.

xorigin  expects the x distance of the origin relative to the lower left
corner of the cursor.

yorigin  expects the y distance of the origin relative to the lower left
corner of the cursor.

DESCRIPTION

curorigin sets the origin of a cursor. The origin is the point on the cur-
sor 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. curorigin does not take effect until
you call setcursor.

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

SEE ALSO

attachcursor, defcursor, setcursor

NOTE

This routine is available only in immediate mode.
NAME

curon, cursoff – control cursor visibility by window

C SPECIFICATION

void curson()
void cursoff()

PARAMETERS

none

DESCRIPTION

curon and cursoff control the visibility of the cursor in the current window. The default is curson.

Use getcursor to find out if the cursor is visible.

SEE ALSO

getcursor

NOTE

This routine is available only in immediate mode.

BUG

On the Personal IRIS, cursor visibility is a global resource. The calls curson and cursoff control cursor visibility regardless of its position on the screen. If a process turns off the cursor, it will remain off until that process is killed or the cursor is turned back on by a call to curson.
NAME

curstype – defines the type and/or size of cursor

C SPECIFICATION

void curstype(typ)
long typ;

PARAMETERS

type expects one of five values that describe the cursor:

C16X1: the default, a 16x16 bitmap cursor of no more than one color.
C16X2: a 16x16 bitmap cursor of no more than three colors.
C32X1: a 32x32 bitmap cursor of no more than one color.
C32X2: a 32x32 bitmap cursor of no more than three colors.
CCROSS: a cross-hair cursor.

DESCRIPTION

curstype defines the type and size of a cursor. After you call curstype call defcursor to specify the glyph’s bitmap and to assign a numeric name to it.

The cross-hair cursor is formed with a horizontal line and a vertical line (each 1 pixel wide) that extend completely across the screen. Its origin (15,15) is at the intersection of the two lines. It is a single-color cursor whose color is mapped by the color index returned by the getgdesc inquiry GD_CROSSHAIR_CINDEX.

SEE ALSO

defcursor, curorigin, getgdesc

NOTES

This routine is available only in immediate mode.
Cursor types C16X2 and C32X2 are not available on systems where the getgdesc inquiry GD_BITS_CURSOR returns 1.
NAME

curvebasis – selects a basis matrix used to draw curves

C SPECIFICATION

void curvebasis(basid)
short basid;

PARAMETERS

basid expects the basis identifier of the basis matrix you want to use
when drawing a curve. (You must have previously called
defbasis to assign a basis identifier to a basis matrix.)

DESCRIPTION

curvebasis selects a basis matrix (by its basis identifier) as the current
basis matrix to draw curve segments. The basis matrix determines how
the system uses the control points when drawing a curve. Depending on
the basis matrix, the system draws bezier curves, cardinal spline curves,
b-spline curves and others. The system does not restrict you to a limited
set of basis matrices. You can define basis matrices to match whatever
constraints you want to place on the curve.

SEE ALSO

crv, crvn, curveprecision, defbasis
NAME

curveit — draws a curve segment

C SPECIFICATION

void curveit(iterationcount)
short iterationcount;

PARAMETERS

iterationcount expects the number of times you want to iterate

DESCRIPTION

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

SEE ALSO

crv
NAME

curveprecision – sets number of line segments used to draw a curve segment

C SPECIFICATION

void curveprecision(nsegments)
short nsegments;

PARAMETERS

nsegments expects the number of line segments to use when drawing a curve segment.

DESCRIPTION

curveprecision sets the number of line segments used to draw a curve. Whenever crv, crvn, rcrv, or rcrvn execute, a number of straight line segments approximate each curve segment. The greater the value of nsegments, the smoother the curve appears, but the longer the drawing time.

SEE ALSO

crv, crvn, curvebasis, rcrv, rcrvn
NAME

cyclemap – cycles between color maps at a specified rate

C SPECIFICATION

void cyclemap(duration, map, nxtmap)
short duration, map, nxtmap;

PARAMETERS

duration expects the number of vertical traces before switching to the
  map named by nxtmap.

map expects the number of the map to use before completing the
  number of vertical sweeps specified by duration.

nxtmap expects the number of the map to use after completing the
  number of vertical sweeps specified by duration.

DESCRIPTION

When the system is inmultimap mode, cyclemap allows you to switch
from one color map to another after a specified duration. In multimap
mode there are 16 color maps, numbered 0-15. You can use cyclemap
within a loop if you want to cycle through more than one map.

EXAMPLE

The code fragment sets up multimap mode and cycle between two maps,
leaving map 1 on for ten vertical retraces and map 3 on for five retraces.

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

SEE ALSO

    blink, gconfig, multimap
NOTE

This routine is available only in immediate mode and cannot be used in onemap mode.
NAME

czclear – clears the color bitplanes and the z-buffer simultaneously

C SPECIFICATION

void czclear(cval, zval)
unsigned long cval;
long zval;

PARAMETERS

cval  expects the color to which you want to clear the color bitplanes.
zval  expects the depth value to which you want to clear the z-buffer.

DESCRIPTION

czclear sets the color bitplanes in the area of the viewport to cval, and the
z buffer bitplanes in the area of the viewport to zval. Multiple color bit-
plane buffers can be cleared simultaneously using the backbuffer and
frontbuffer commands. The screen mask, when it is set to a subregion
of the viewport, bounds the cleared region. Most other drawing modes,
including alpha function, blend function, logical operation, polygon fill
pattern, stenciling, texture mapping, writemask, and z buffering, have no
effect on the operation of czclear. The current color does not change.

Because only the normal framebuffer includes a z buffer, czclear should
be called only while draw mode is NORMALDRAW.

In RGB mode, the cval parameter expects a packed integer of the same
format used by cpack, namely 0xaaggbrr, where rr is the red value, bb
the blue value, gg the green value, and aa is the alpha value. In color
map mode this parameter expects an index into the current color map, so
only up to 12 of the least-significant bits are significant.

The valid range of the zval parameter depends on the graphics hardware,
where the minimum is the value returned by getgdesc(GD_ZMIN) and
the maximum is the value returned by getgdesc(GD_ZMAX). It is
unaffected by the state of the GLCZRANGEMAP compatibility mode
(see glcompat).
After `czclear` executes, the graphics position is undefined.

SEE ALSO

afunction, blendfunction, clear, cpack, getgdesc, glcompat, logicop, scrmask, setpattern, stencil, texbind, wmpack, writemask, zbuffer, zclear, zfuction

NOTES

Whenever you need to clear both the z-buffer and the color bitplanes to constant values at the same time, use `czclear`. A simultaneous clear will take place if circumstances allow it. There is never a penalty in calling `czclear` over calling `clear` and `zclear` sequentially.

IRIS-4D GT and GTX models can do a simultaneous clear only under the following circumstances:

- In RGB mode, the 24 least significant bits of `cval` (red, green, and blue) must be identical to the 24 least significant bits of `zval`.
- In color map mode, the 12 least significant bits of `cval` must be identical to the 12 least significant bits of `zval`.

IRIS-4D VGX models always clear color and z bitplanes banks sequentially, regardless of the values of `cval` and `zval`.

On the Personal Iris, you can speed up `czclear` by as much as a factor of four for common values of `zval` if you call `zfuction` in conjunction with it such that one of the following conditions are met:

<table>
<thead>
<tr>
<th>zval</th>
<th>zfuction</th>
</tr>
</thead>
<tbody>
<tr>
<td>getgdesc(GD_ZMIN)</td>
<td>ZF_GREATER or ZF_GEQUAL</td>
</tr>
<tr>
<td>getgdesc(GD_ZMAX)</td>
<td>ZF_LESS or ZF_LEQUAL</td>
</tr>
</tbody>
</table>

BUGS

IRIS-4D G models always clear their z-buffers to `GD_ZMAX`, regardless of the value passed to `czclear`.
NAME
dbtext – sets the dial and button box text display

C SPECIFICATION
void dbtext(str)
String str;

PARAMETERS
str expects a pointer to a text string of no more than eight characters: digits, spaces, and uppercase letters only.

DESCRIPTION
dbtext places up to eight characters of text into the text display on the dial and button box.

SEE ALSO
setdblights

NOTES
This routine is available only in immediate mode.
As might be expected, this routine does not function if you use the dial and button box without a text display.
NAME

defbasis – defines a basis matrix

C SPECIFICATION

void defbasis(id, mat)
short id;
Matrix mat;

PARAMETERS

id expects the basis matrix identifier you want to assign to the matrix at mat.

mat expects the matrix to which you want to assign the basis matrix identifier, id.

DESCRIPTION

defbasis assigns a basis matrix identifier to a basis matrix. The basis matrix is used by the routines that generate curves and patches. Use the basis matrix identifier in subsequent calls to curvebasis and patchbasis.

SEE ALSO

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

NOTE

This routine is available only in immediate mode.
NAME

defcursor – defines a cursor glyph

C SPECIFICATION

void defcursor(n, curs)
short n;
unsigned short *curs;

PARAMETERS

n expects the constant you want to assign as a cursor name. By
default, an arrow is defined as cursor 0 and cannot be redefined.

curs expects the bitmap for the cursor you want to define. The bitmap
can be 16x16 or 32x32 and either one or two layers deep. This
parameter is ignored for cross-hair cursors.

DESCRIPTION

defcursor defines a cursor glyph with the specified name and bitmap.
Call curstype prior to calling defcursor to set the type and size of cur-
sor it defines. The name parameter n is used to identify the cursor glyph
to other cursor routines. A subsequent call to defcursor with the same
value of n will replace the current definition of the cursor with the new
one.

By default, the cursor origin of a bitmap cursor is at (0,0), its lower-left
corner, and the cursor origin of a cross-hair cursor is at (15,15), the
intersection of its two lines. Use curorigin to set the cursor origin to
somewhere else. The cursor origin is the position controlled by valua-
tors attached to the cursor, and is also the position pick uses for the
picking region.

SEE ALSO

curorigin, curstype, getcursor, getgdesc, pick, setcursor

NOTES

This routine is available only in immediate mode.
Some models do not support two-layer cursor bitmaps. Use the getgdesc inquiry GD_BITS_CURSOR to determine how many layers are supported.
NAME
deflinestyle — defines a linestyle

C SPECIFICATION

void deflinestyle(n, ls)
short n;
Linestyle ls;

PARAMETERS

n expects the constant that you want to use as an identifier for the
linestyle described by ls. This constant is used as an index into a
table of linestyles. By default, index 0 contains the pattern 0xFFFF,
which draws solid lines and cannot be redefined.

ls expects a 16-bit pattern to use as a linestyle. This pattern is stored in
the linestyle table at index n. You can define up to $2^{16}$ distinct
linestyles.

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. To replace a linestyle, respecify the previous index.

SEE ALSO
defcursor, defpattern, defrasterfont, getlstyle, lsrepeat, setlinestyle

NOTES

This routine is available only in immediate mode.

On the Personal Iris, there is a performance penalty for drawing non-
solid lines; there is no penalty on the other IRIS-4D models.
NAME

defpattern – defines patterns

C SPECIFICATION

    void defpattern(n, size, mask)
    short n, size;
    unsigned short mask[];

PARAMETERS

    n    expects the constant that you want to use as an identifier for the
         pattern described by mask. This constant is used as an index into
         a table of patterns. By default, pattern 0 is a 16X16 solid pattern
         that cannot be changed.

    size  expects the size of the pattern: 16, 32, or 64 for a 16×16-, 32×32-, or 64×64-bit pattern, respectively.

    mask expects an array of 16-bit integers that form the actual bit pattern. The system stores the pattern in a pattern table at index n. The pattern is described from left to right and bottom to top, just as characters are described in a raster font.

DESCRIPTION

defpattern allows you to define an arbitrary pattern and assign it an identifier. You can later reference this pattern in other routines via its identifier. Patterns are available to all windows when using multiple windows.

Patterns affect the filling of polygons, including rectangles, arcs, and circles, as well as polygons specified with individual vertices. Patterns have no effect on the scan conversion of points, lines, or characters, or on pixel write or copy operations.

When a pattern is active (see setpattern) it is effectively replicated across the entire screen, with the edges of pattern tiles aligned to the left and bottom edges of the screen. Bit 15 of each 16-bit description word is leftmost, and words are assembled left to right, then bottom to top, to form each pattern square. Pixels on the screen that correspond to zeros in the pattern remain unmodified during scan conversion of polygons. No changes are made to any bitplane bank of a protected pixel.
SEE ALSO

deflinestyle, defrasterfont, getpattern, setpattern

NOTES

This routine is available only in immediate mode.

Some machines do not support 64x64 patterns. Call getgdesc(GD_PATSIZE_64) to determine the availability of 64x64 patterns.

On the Personal Iris there is a performance penalty for non-solid patterns.
NAME

defpup – defines a menu

C SPECIFICATION

long defpup(str [, args ... ])
String str;
long args;

PARAMETERS

str expects a pointer to the text that you want to add as a menu item. In addition, you have the option of pairing an "item type" flag with each menu item. There are seven menu item type flags:

%t marks item text as the menu title string.

%F invokes a routine for every selection from this menu except those marked with a %n. You must specify the invoked routine in the arg parameter. The value of the menu item is used as a parameter of the executed routine. Thus, if you select the third menu item, the system passes 3 as a parameter to the function specified by %F.

%f invokes a routine when this particular menu item is selected. You must specify the invoked routine in the arg parameter. The value of the menu item is passed as a parameter of the routine. Thus, if you select the third menu item, the system passes 3 as a parameter to the routine specified by %f. If you have also used the %F flag within this menu, then the result of the %f routine is passed as a parameter of the %F routine.

%l adds a line under the current entry. This is useful in providing visual clues to group like entries together.

%m pops up a menu whenever this menu item is selected. You must provide the menu identifier of the new menu in the arg parameter.
%n like %f, this flag invokes a routine when the user selects this menu item. However, %n differs from %f in that it ignores the routine (if any) specified by %F. The value of the menu item is passed as a parameter of the executed routine. Thus, if you select the third menu item, the system passes 3 as a parameter to the function specified by %f.

%xn assigns a numeric value to this menu item. This value overrides the default position-based value assigned to this menu item (e.g., the third item is 3). You must enter the numeric value as the n part of the text string. Do not use the arg parameter to specify the numeric value.

args an optional set of arguments. Each argument expects the command or submenu that you want to assign to this menu item. You can use as many args parameters as you need.

FUNCTION RETURN VALUE

The returned value for the function is the menu identifier of the menu just defined.

DESCRIPTION

defpup defines a pop-up menu under the window manager and returns a positive menu identifier as the function value.

EXAMPLES

Examples best illustrate the use of the item types.

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

defines a pop-up menu with title menu 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:

String str = "menu2 %t %f|1 %n%1|2 %m|3 %f|4 %x234";

menu2 = defpup(str, menufunc, submenu, func);

defines a menu with title menu2 and four items with a line under the first one. Invoked by:

menuval = dopup(menu2);

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.

SEE ALSO

addtopup, dopup, freepup, newpup

NOTES

This routine is available only in immediate mode.
When using the Distributed Graphics Library (DGL), you can not call other DGL routines within a function that is called by a popup menu, i.e. a function given as the argument to a %f or %F item type.
NAME

defrasterfont – defines a raster font

C SPECIFICATION

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

PARAMETERS

n expects the constant that you want to use as the identifier for this raster font. This constant is used as an index into a font table. The default font, 0, is a fixed-pitch font with a height of 16 and width of 9. Font 0 cannot be redefined.

ht expects the maximum height (in pixels) for a character.

nc expects the number of characters in this font.

chars expects an array of character description structures of type Fontchar. The Fontchar structure is defined in <gl/gl.h> as:

typedef struct {
    unsigned short offset;
    Byte w, h;
    signed char xoff, yoff;
    short width;
} Fontchar;

offset expects the element number of raster at which the bitmap for this character starts. The element numbers start at zero.

w expects the number of columns in the bitmap that contain set bits (character width).

h expects the number of rows in the bitmap of the character (including ascender and descender).

xoff expects bitmap columns between the start of the character’s bitmap and the start of the character.
yoff expects the number rows between the character’s baseline and the bottom of the bitmap. For characters with descenders (e.g., g) this value is a negative number. For characters that rest entirely on the baseline, this value is zero.

width expects the pixel width for the character. This value tells the system how far to space after drawing the character. (This value is added to the character position.)

nr expects the number of 16-bit integers in raster.

raster expects a one-dimensional array that contains all the bit maps (masks) for the characters in the font. Each element of the array is a 16-bit integer and the elements are ordered left to right, bottom to top. When interpreting each element, the bits are left justified within the character’s bounding box.

The maximum row width for a single bitmap is not limited to the capacity of a single 16-bit integer array element. The rows of a bitmap may span more than one array element. However, each new row in the character bitmap must start with its own array element. Likewise, each new character bitmap must start with its own array element. The system reads the row width and starting location for a character bitmap from the structures in the chars array.

DESCRIPTION

defrasterfont defines a raster font. The hardest part of creating a new raster font is generating a bit map for each character. You may want to write a graphically oriented tool for creating the bitmaps expected by raster.

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. Patterns, cursors, and fonts are available to all windows when using multiple windows.

SEE ALSO

charstr, cmove, font, getcpos, getdescender, getfont, getheight, strwidth
NOTE

This routine is available only in immediate mode.
NAME

delobj – deletes an object

C SPECIFICATION

void delobj(obj)
Object obj;

PARAMETERS

obj expects the object identifier of the object that you want to delete.

DESCRIPTION

delobj deletes an object. Deleting an object frees most of its display list storage; the object identifier remains undefined until you create a new object for that identifier. The system ignores calls to delete objects that don’t exist.

SEE ALSO

compactify, makeobj

NOTE

This routine is available only in immediate mode.
NAME

deltag – deletes a tag from the current open object

C SPECIFICATION

void deltag(t)
Tag t;

PARAMETERS

\( t \) expects the tag that you want to delete.

DESCRIPTION

deltag deletes the specified tag from the object currently open for editing. You cannot delete the special tags STARTTAG and ENDTAG.

SEE ALSO

editobj, maketag

NOTE

This routine is available only in immediate mode.
NAME

depthcue – turns depth-cue mode on and off

C SPECIFICATION

void depthcue(mode)
Boolean mode;

PARAMETERS

mode expects either TRUE or FALSE.
TRUE turns depthcue mode on.
FALSE turns depthcue mode off.

DESCRIPTION

depthcue turns depth-cue mode on or off. If depth-cue mode is on, all lines, points, characters, and polygons are drawn depth-cued. This means the z values and the range of color values specified by lshaderange or lRGBrange determine the color of the lines, points, characters, or polygons. The z values, whose range is set by lsetdepth, are mapped linearly into the range of color values. In this mode, lines that vary greatly in z value span the range of colors specified by lshaderange or lRGBrange.

In color index mode, the color map entries specified by lshaderange should be loaded with a series of colors that gradually increase or decrease in intensity.

SEE ALSO

lRGBrange, lsetdepth, lshaderange
NAME
dgIClose – closes the DGL server connection

C SPECIFICATION

void dgIClose(sid)
long sid;

PARAMETERS

sid expects the identifier of the server you want to close. If sid is negative, then all graphics server connections are closed. Server identifiers are returned by dglopen.

DESCRIPTION
dgIClose closes the connection to the graphics server associated with the server identifier sid, killing the Distributed Graphics Library (DGL) server process and all its windows. If sid is negative, then all graphics server connections are closed. Call dgIClose after gexit or when the graphics server is no longer needed. Closing the connection frees up resources on the graphics server.

After a connection is closed, there is no current graphics window and no current graphics server. Calling any routines other than dglopen, dgIClose or routines that take graphics window identifiers as input parameters will result in an error.

SEE ALSO
dglopen

4Sight User’s Guide, “Using the GL/DGL Interfaces”.

NOTE
This routine is available only in immediate mode.
NAME
dglopen – opens a DGL connection to a graphics server

C SPECIFICATION

long dglopen(svname, type)
String svname;
long type;

PARAMETERS

svname expects a pointer to the name of the graphics server to which
you want to open a connection.

For a successful connection, the username on the server must
be equivalent (in the sense of rlogin(1C)) to the originating
account; no provision is made for specifying a password. The
remote username used is the same as the local username unless
you specify a different remote username. To specify a dif-
f erent remote username, the svname string should use the for-
m at username@servername.

For DECnet connections, if the server account has a password,
this password must be specified using the format
username password@servername. This password is used only
for opening the DECnet connection; the two accounts must
still be equivalent in the rlogin sense.

type expects a symbolic constant that specifies the kind of connec-
tion. There are three defined constants for this parameter:

DGLLOCAL indicates a direct connection to the local graph-
ics hardware.

DGLTSOCKET indicates a remote connection via TCP/IP.

DGL4DDN indicates a remote connection via DECnet.

FUNCTION RETURN VALUE

If the connection succeeds, the returned value of the function is a non-
negative integer, serverid, that identifies the graphics server. If the con-
nection failed, the returned value for the function is a negative integer.
The absolute value of a negative returned value is either a standard error value (defined in `<errno.h>`) or one of several error returns associated specifically with `dglopen`:

- **ENOMEM**  
  *type* is not a valid connection type.

- **EACCESS**  
  login incorrect or permission denied.

- **EMFILE**  
  too many graphics connections are currently open.

- **EBUSY**  
  only one DGLLOCAL connection allowed.

- **ENOPROTOOPT**  
  DGL service not found in `/etc/services`.

- **ERANGE**  
  invalid or unrecognizable number representation.

- **EPROTONOSUPPORT**  
  DGL version mismatch.

- **ESRCH**  
  the window manager is not running on the server.

**DESCRIPTION**

`dglopen` opens a Distributed Graphics Library (DGL) connection to a graphics server (*svname*). After a connection is open, all graphics input and output are directed to that connection. Graphics input and output continue to be directed to the connection until either the connection is closed, another connection is opened or a different connection is selected. A different connection can be selected by calling a subroutine that takes a graphics window identifier as an input parameter, e.g. `winset`. The server connection associated with that graphics window identifier becomes the current connection. To close a DGL connection, call `dglclose` with the server identifier returned by `dglopen`.

**SEE ALSO**

dglclose, finish, gflush, winopen, winset
rlogin(1C) in the *IRIS-4D User's Reference Manual*

*4Sight User's Guide*, “Using the GL/DGL Interfaces”.

Version 4.0 - 2 - April 1990
NOTES

This routine is available only in immediate mode.

This routine is available in both the DGL and GL library. However, only a **DGLLOCAL** connection type is supported by the GL library.
NAME

dopup – displays the specified pop-up menu

C SPECIFICATION

long dopup(pup)
long pup;

PARAMETERS

pup expects the identifier of the pop-up menu you want to display.

FUNCTION RETURN VALUE

The returned value of the function is the value of the item selected from the pop-up menu. If the user makes no menu selection, the returned value of the function is −1.

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.

When you first define the menu (using defpup or addtopup) you specify the list of menu entries and their corresponding actions. See addtopup for details.

SEE ALSO

addtopup, defpup, freepup, newpup

NOTE

This routine is available only in immediate mode.
NAME

doublebuffer – sets the display mode to double buffer mode

C SPECIFICATION

void doublebuffer()

PARAMETERS

none

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.

In double buffer mode, gconfig calls frontbuffer(OFF) and backbuffer(ON).

SEE ALSO

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

NOTE

This routine is available only in immediate mode.
NAME

draw, drawi, draws, draw2, draw2i, draw2s – draws a line

C SPECIFICATION

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

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

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

    void draw2(x, y)
    Coord x, y;

    void draw2i(x, y)
    Icoord x, y;

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

All of the above functions are functionally the same except for the type declarations of the parameters. In addition the draw2* routines assume a 2-D point instead of a 3-D point.

PARAMETERS

    x    expects the x coordinate of the point to which you want to draw a line segment.

    y    expects the y coordinate of the point to which you want to draw a line segment.

    z    expects the z coordinate of the point to which you want to draw a line segment. (Not used by 2-D subroutines.)

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 depthcue 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**

bgnline, endline, move, v

**NOTE**

draw should not be used in new development. Rather, lines should be drawn using the high-performance v commands, surrounded by calls to bgnline and endline.
NAME

drawmode – selects which GL framebuffer is drawable

C SPECIFICATION

void drawmode(mode)
  long mode;

PARAMETERS

mode expects the identifier of the framebuffer to which GL drawing
commands are to be directed:

NORMALDRAW, which sets operations for the normal color
and z buffer bitplanes.

OVERDRAW, which sets operations for the overlay bitplanes.

UNDERDRAW, which sets operations for the underlay bit-
planes.

PUPDRAW, which sets operations for the pop-up bitplanes.

CURSORDRAW, which sets operations for the cursor.

DESCRIPTION

The IRIS physical framebuffer is divided into 4 separate GL frame-
buffers: pop-up, overlay, normal, and underlay. drawmode specifies
which of these four buffers is currently being controlled and modified by
GL drawing and mode commands. Because drawmode cannot be set to
multiple framebuffers, GL drawing commands affect only one of the
four GL framebuffers at a time.

The way that GL modes interact with drawmode is both complex and
significant to the GL programmer. For example, each framebuffer
maintains its own current color and its own color map, but linewidth is
shared among all framebuffers. In general, modes that determine what
is to be drawn into the framebuffers are shared; modes that control
framebuffer resources are either multiply specified, or specified only for
the normal framebuffer.
A separate version of each of the following modes is maintained by each GL framebuffer. These modes are modified and read back based on the current draw mode:

- backbuffer
- cmode
- color or RGBcolor
- doublebuffer
- frontbuffer
- mapcolor (a separate color map per framebuffer)
- readsource
- RGBmode
- singlebuffer
- writemask or RGBwritemask

The following modes currently affect only the operation of the normal framebuffer. They must therefore be modified only while draw mode is NORMALDRAW. As features are added to the GL, these modes may become available in other draw modes. When this happens, a separate mode will be maintained for each draw mode.

- acsiz
- blink
- cyclemap
- multimap
- onemap
- setmap
- stencil
- stensize
- swritemask
- zbuffer
- zdraw
- zfunction
- zsource
- zwritemask

All other modes, including matricies, viewports, graphics and character positions, lighting, and many primitive rendering options, are shared by the four GL framebuffers.
Draw mode CURSOR_DRAW differs from the others. True bitplanes for the cursor do not exist; there is no current color or writemask in this drawing mode. However, the cursor does have its own color map, and when in this mode, mapcolor and getmcolor access it.

SEE ALSO

acsizex, cmode, c, color, cpack, gconfig, getcolor, getmcolor, getwritemask, mapcolor, overlay, stencil, underlay, wmpack, writemask

NOTE

This routine is available only in immediate mode.

PUPDRAW mode is provided for compatibility, its use is discouraged.

Some GL modes that are shared by all draw modes are not implemented by the popup, overlay, or underlay framebuffers. For example, the Personal Iris does not do Gouraud shading in these framebuffers. It is important for the programmer to explicitly disable modes that are shared, but not desired, when in draw modes other than NORMALDRAWX. Otherwise the code may function differently on different platforms.
NAME

editobj – opens an object definition for editing

C SPECIFICATION

void editobj(obj)
Object obj;

PARAMETERS

obj expects object identifier for object definition you want to edit.

DESCRIPTION

editobj opens an object definition for editing. The system maintains an
editing pointer that initially points to the end of the definition. The sys-
tem appends all new routines at that pointer location until you call
closeobj or until you call a routine that repositions the editing pointer,
such as objdelete, objinsert, or objreplace.

Usually, you need not be concerned about memory allocation. Objects
grow and shrink automatically as routines are added and deleted. (See
chunksize.)

If you call editobj for an undefined object identifier, the system displays
an error message.

SEE ALSO

compactify, objdelete, objinsert, objreplace, chunksize

NOTE

This routine is available only in immediate mode.
NAME
bgnclosedline, endclosedline – delimit the vertices of a closed line

C SPECIFICATION
void bgnclosedline()
void endclosedline()

PARAMETERS
none

DESCRIPTION
bgnclosedline marks the start of a group of vertex routines that you want interpreted as points on a closed line. Use endclosedline to mark the end of the vertex routines that are part of the closed line.

A closed line draws a line segment from one vertex on the list to the next vertex on the list. When the system reaches the end of the vertex list, it draws a line that connects the last vertex to the first vertex. All segments use the current linestyle, which is reset prior to the first segment and continues through subsequent segments. To specify a vertex, use the v routine.

Between bgnclosedline and endclosedline, you can issue only the following Graphics Library routines: c, color, cpack, lmbind, lmcolor, lmdc, n, RGBcolor, t, and v. Within a closed line, you should use lmdc and lmbind only to respecify materials and their properties. If the color changes between a pair of vertices, the color of the line segment will be constant if the current shading model is FLAT and interpolated if the current shading model is GOURAUD. In color map mode, the colors vary through the color map; to get reasonable results, the color map should contain a ramp.

There is no limit to the number of vertices that can be specified between bgnclosedline and endclosedline. After endclosedline, the system draws a line from the final vertex back to the initial vertex, and the current graphics position is left undefined.
By default line vertices are forced to the nearest pixel center prior to scan conversion. Line accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when lines are scan converted with antialiasing enabled (see linesmooth).

bgnclosedline/endclosedline are the same as bgnline/endline, except they connect the last vertex to the first.

EXAMPLE

The code fragment below draws the outline of a triangle. Lines use the current linestyle, which is reset prior to the first vertex and continues through all subsequent vertices.

bgnclosedline();
v3f(vert1);
v3f(vert2);
v3f(vert3);
endclosedline();

SEE ALSO

bgnline, c, linesmooth, linewidth, lsrepeat, scrsubdivide, setlinestyle, shademodel, subpixel, v

BUGS

On the IRIS-4D B and G models, and on the Personal Iris without Turbo Graphics, if the color changes between a pair of vertices, the color of the line segment will be constant regardless of the current shading model.

On the IRIS-4D GT and GTX models, if the color changes between a pair of vertices, the color of the line segment will be interpolated regardless of the current shading model.
NAME

feedback, endfeedback – control feedback mode

C SPECIFICATION

Personal Iris and IRIS-4D VGX:

```c
void feedback(buffer, size)
float buffer[];
long size;
long endfeedback(buffer)
float buffer[];
```

Other models:

```c
void feedback(buffer, size)
short buffer[];
long size;
long endfeedback(buffer)
short buffer[];
```

PARAMETERS

- **buffer** expects a buffer into which the system writes the feedback output from the Geometry Pipeline. On the Personal Iris and the IRIS-4D VGX, the output consists of 32-bit floating point values; on the other IRIS-4D models, the output consists of 16-bit integer values. Be sure you declare your buffer appropriately.

- **size** expects the maximum number of buffer elements into which the system will write feedback output.

FUNCTION RETURN VALUE

The return value of **endfeedback** is the actual number of elements of **buffer** that were written. The system will not write more than **size** elements, even when the amount of feedback exceeds it. You should assume that overflow has occurred whenever the return value is **size**.
DESCRIPTION

feedback puts the system in feedback mode. In feedback mode, the system retains the output of the Geometry Pipeline rather than sending it to the rendering subsystem. endfeedback turns off feedback mode and returns the feedback output in buffer. This information is typically a description of a vertex, and is machine specific. For information for interpreting the returned buffer, see the "Feedback" chapter of the Graphics Library Programming Guide.

NOTE

These routines are available only in immediate mode.
NAME

endfullscrn – ends full-screen mode

C SPECIFICATION

void endfullscrn()

PARAMETERS

none

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

fullscrn

NOTE

This routine is available only in immediate mode.
NAME

bgnline, endline – delimit the vertices of a line

C SPECIFICATION

void bgnline()

void endline()

PARAMETERS

none

DESCRIPTION

Vertices specified after bgnline and before endline are interpreted as endpoints of a series of line segments. Use the v routine to specify a vertex. The first vertex connects to the second; the second connects to the third; and so on until the next-to-last vertex connects to the last one. The last vertex does not connect to the first vertex. Use bgnclosedline to connect the first and last points. All segments use the current linestyle, which is reset prior to the first segment and continues through subsequent segments.

Between bgnline and endline, you can issue only the following Graphics Library routines: c, color, cpack, Imbind, Imcolor, Imdef, n, RGBcolor, t, and v. Imdef and Imbind can be used to respecify only materials and their properties. If the color changes between a pair of vertices, the color of the line segment will be constant if the current shading model is FLAT and interpolated if the current shading model is GOURAUD. In color map mode, the colors vary through the color map; to get reasonable results, the color map should contain a ramp.

There is no limit to the number of vertices that can be specified between bgnline and endline. After endline, the current graphics position is undefined.

By default line vertices are forced to the nearest pixel center prior to scan conversion. Line accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when lines are scan converted with antialiasing enabled (see linesmooth).
SEE ALSO
bgnclosedline, c, linesmooth, linewidth, lsrepeat, scrsubdivide, setlinestyle, shademodel, subpixel, v

BUGS
On the IRIS-4D B and G models, and on the Personal Iris without Turbo Graphics, if the color changes between a pair of vertices, the color of the line segment will be constant regardless of the current shading model.

On the IRIS-4D GT and GTX models, if the color changes between a pair of vertices, the color of the line segment will be interpolated regardless of the current shading model.
NAME

`endpick` — turns off picking mode

C SPECIFICATION

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

PARAMETERS

`buffer` expects a buffer into which to append the contents of the name stack when a drawing routine draws in the picking region. Before writing the contents of the name stack, the system appends the number of entries it is about to append. Thus, if the name stack contains the values, 5, 9, and 17; then `endpick` appends the values, 3, 5, 9, and 17, to `buffer`.

Because more than one drawing routine may have written in the picking region, it is possible for `buffer` to contain a number of readings from the name stack.

FUNCTION RETURN VALUE

The returned value for the function is the number of times `endpick` wrote the names stack to `buffer`.

If the returned function value is negative, then the buffer was too small to contain all the readings from the name stack.

DESCRIPTION

`endpick` turns off picking mode and writes the hits to a buffer.

SEE ALSO

`initnames`, `loadname`, `pick pushname`, `popname`

NOTE

This routine is available only in immediate mode.
NAME

bgnpoint, endpoint – delimit the interpretation of vertex routines as points

C SPECIFICATION

void bgnpoint()
void endpoint()

PARAMETERS

none

DESCRIPTION

bgnpoint marks the beginning of a list of vertex routines that you want interpreted as points. Use the endpoint routine to mark the end of the list. For each vertex, the system draws a one-pixel point into the frame buffer. Use the v routine to specify a vertex.

Between bgnpoint and endpoint, you can issue only the following Graphics Library routines: c, color, cpack, Imbind, lmcolor, lmdef, n, RGBcolor, t, and v. Use lmdef and lmbind to respecify only materials and their properties.

There is no limit to the number of vertices that can be specified between bgnpoint and endpoint.

By default points are forced to the nearest pixel center prior to scan conversion. This coercion is defeated with the subpixel command. Subpixel point positioning is important only when points are scan converted with antialiasing enabled (see pntsmooth).

After endpoint, the current graphics position is the most recent vertex.

SEE ALSO

c, pntsmooth, subpixel, v
NAME

bgnpolygon, endpolygon – delimit the vertices of a polygon

C SPECIFICATION

void bgnpolygon()
void endpolygon()

PARAMETERS

none

DESCRIPTION

Vertices specified after bgnpolygon and before endpolygon form a single polygon. The polygon can have no more than 256 vertices. Use the v subroutine to specify a vertex. Self-intersecting polygons (other than four-point bowties) may render incorrectly. Likewise, concave polygons may not render correctly if you have not called concave(TRUE).

Between bgnpolygon and endpolygon, you can issue only the following Graphics Library subroutines: c, color, cpack, lmbind, lmcolor, lmdef, n, RGBcolor, t, and v. Use lmdef and lmbind to respecify only materials and their properties.

By default polygon vertices are forced to the nearest pixel center prior to scan conversion. Polygon accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when polygons are scan converted with antialiasing enabled (see polysmooth).

After endpolygon, the current graphics position is undefined.

SEE ALSO

backface, c, concave, frontface, polymode, polysmooth, scrsubdivide, setpattern, shademodel, subpixel, v
NOTES

If you want to use the **backface** or **frontface** routines, specify the vertices in counter-clockwise order.

Although calling **concave(TRUE)** will guarantee that all polygons will be drawn correctly, on the IRIS-4D B and G models, and on the Personal Iris, doing so cause their performance to be degraded.
NAME

pupmode, endpupmode – obsolete routines

C SPECIFICATION

void pupmode()
void endpupmode()

PARAMETERS

none

DESCRIPTION

These routines are obsolete. Although pupmode/endpupmode continue to function (to provide backwards compatibility) all new development should use drawmode to access the pop-up menu bitplanes.

SEE ALSO

drawmode
NAME

bgnqstrip, endqstrip – delimit the vertices of a quadrilateral strip

C SPECIFICATION

void bgnqstrip()
void endqstrip()

DESCRIPTION

Vertices specified between bgnqstrip and endqstrip are used to define a strip of quadrilaterals. The graphics pipe maintains three vertex registers. The first, second, and third vertices are loaded into the registers, but no quadrilateral is drawn until the system executes the fourth vertex routine. Upon executing the fourth vertex routine, the system draws a quadrilateral through the vertices, then replaces the two oldest vertices with the third and fourth vertices.

For each new pair of vertex routines, the system draws a quadrilateral through two new vertices and the two older stored vertices, then replaces the older stored vertices with the two new vertices.

Between bgnqstrip and endqstrip you can issue the following Graphics Library routines: c, color, cpack, lmbind, lmcolor, lmdef, n, RGBcolor, t, and v. Use lmdef and lmbind only to respecify materials and their properties.

If you want to use backface, you should specify the vertices of the first quadrilateral in counter-clockwise order. All quadrilaterals in the strip have the same rotation as the first quadrilateral in a strip, so that back-facing works correctly.

There is no limit to the number of vertices that can be specified between bgnqstrip and endqstrip. The result is undefined, however, if an odd number of vertices are specified, or if fewer than four vertices are specified.

By default quadrilateral vertices are forced to the nearest pixel center prior to scan conversion. Quadrilateral accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when quadrilaterals are scan converted with antialiasing enabled (see polysmooth).
After **endqstrip**, the current graphics position is undefined.

**EXAMPLE**

For example, the code sequence:

```c
bgnqstrip();
v3f(zero);
v3f(one);
v3f(two);
v3f(three);
v3f(four);
v3f(five);
v3f(six);
v3f(seven);
endqstrip();
```

draws three quadrilaterals: (0,1,2,3), (2,3,4,5), and (4,5,6,7). Note that the vertex order required by quadrilateral strips matches the order required by the equivalent triangle mesh. The vertices above, when places between **bgnntmesh** and **endntmesh** calls, draws six triangles: (0,1,2), (1,2,3), (2,3,4), (3,4,5), (4,5,6), and (5,6,7).

**SEE ALSO**

backface, c, concave, frontface, polymode, polysmooth, scrsubdivide, setpattern, shademodel, subpixel, v

**NOTE**

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support quadrilateral strips. Use **getgdesc** to determine whether quadrilateral strips are supported.

IRIS-4D VGX models use vertex normals to improve the shading quality of quadrilaterals, regardless of whether lighting is enabled.
endselect

NAME

endselect – turns off selecting mode

C SPECIFICATION

long endselect(buffer)
short buffer[];

PARAMETERS

buffer expects a buffer into which to write hits.

FUNCTION RETURN VALUE

The returned function values is the number of hits made while selection mode was active. Each time there is a hit, the system writes the name stack to buffer.

If the value returned is negative, the buffer is not large enough to hold all the hits that occurred.

DESCRIPTION

endselect turns off selection mode. The buffer stores any hits generated by drawing routines 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.

SEE ALSO

gselect, loadname, initnames pushname, popname

NOTE

This routine is available only in immediate mode.
NAME

begnsurface, endsurface – delimit a NURBS surface definition

C SPECIFICATION

void begnsurface()
void endsurface()

PARAMETERS

none

DESCRIPTION

Use begnsurface to mark the beginning of a NURBS (Non-Uniform Rational B-Spline) surface definition. After you call begnsurface, call the routines that define the surface and that provide the trimming information. To mark the end of a NURBS surface definition, call endsurface.

Within a NURBS surface definition (between begnsurface and endsurface) you may use only the following Graphics Library subroutines: nurbsurface, bgntrim, endtrim, nurbscurve, and pwlcurve. The NURBS surface definition must consist of exactly one call to nurbsurface to define the shape of the surface. In addition, this call may be preceded by calls to nurbsurface that specify how texture and color parameters vary across the surface. The call(s) to nurbsurface may be followed by a list of one or more trimming loop definitions (to define the boundaries of the surface). Each trimming loop definition consists of one call to bgntrim, one or more calls to either pwlcurve or nurbscurve, and one call to endtrim.

The system renders a NURBS surface as a polygonal mesh, and calculates normal vectors at the corners of the polygons within the mesh. Therefore, your program should specify a lighting model if it uses NURBS surfaces. If your program uses no lighting model, all the interesting surface information is lost. When using a lighting model, use lmdef and lmbind to define or modify materials and their properties.
EXAMPLE

The following code fragment draws a NURBS surface trimmed by two closed loops. The first closed loop is a single piecewise linear curve (see **pwlcurve**), and the second closed loop consists of two NURBS curves (see **nurbsscurve**) joined end to end:

```c
bgnssurface();
  nurbssurface(. . .);
  bgntrim();
    pwlcurve(. . .);
  endtrim();
  bgntrim();
    nurbscurve(. . .);
    nurbscurve(. . .);
  endtrim();
endsurface();
```

SEE ALSO

nurbssurface, bgntrim, nurbscurve, pwlcurve, setnurbsproperty, getnurbsproperty
NAME

bgnntmesh, endtmesh – delimit the vertices of a triangle mesh

C SPECIFICATION

void bgnntmesh()
void endtmesh()

PARAMETERS

none

DESCRIPTION

Vertices specified between bgnntmesh and endtmesh are used to define a mesh of triangles. The graphics pipe maintains two vertex registers. The first and second vertices are loaded into the registers, but no triangle is drawn until the system executes the third vertex routine. Upon executing the third vertex routine, the system draws a triangle through the vertices, then replaces the older of the register vertices with the third vertex.

For each new vertex routine, the system draws a triangle through the new vertex and the stored vertices, then (by default) replaces the older stored vertex with the new vertex. If you want the system to replace the more recent of the stored vertices, call swaptmesh prior to calling v.

Between bgnntmesh and endtmesh you can issue the following Graphics Library routines: c, color, cpack, lmbind, lmcolor, lmdef, n, RGBcolor, swaptmesh, t, and v. Use lmdef and lmbind only to respecify materials and their properties.

If you want to use backface, you should specify the vertices of the first triangle in counter-clockwise order. All triangles in the mesh have the same rotation as the first triangle in a mesh so that backfacing works correctly.

There is no limit to the number of vertices that can be specified between bgnntmesh and endtmesh.
By default triangle vertices are forced to the nearest pixel center prior to scan conversion. Triangle accuracy is improved when this coercion is defeated with the subpixel command. Subpixel vertex positioning is especially important when triangles are scan converted with antialiasing enabled (see polysmooth).

After endtmesh, the current graphics position is undefined.

EXAMPLE

For example, the code sequence:

```c
bgntmesh();
v3f(zero);
v3f(one);
v3f(two);
v3f(three);
endtmesh();
```

draws two triangles, (zero,one,two) and (one,two,three), while the code sequence:

```c
bgntmesh();
v3f(zero);
v3f(one);
swapmesh();
v3f(two);
v3f(three);
endtmesh();
```

draws two triangles, (zero,one,two) and (zero,two,three). There is no limit to the number of times that swapmesh can be called.

SEE ALSO

backface, c, concave, frontface, polymode, polysmooth, scrsubdivide, setpattern, shademodel, subpixel, swapmesh, v
NAME

bgntrim, endtrim – delimit a NURBS surface trimming loop

C SPECIFICATION

void bgntrim()
void endtrim()

PARAMETERS

none

DESCRIPTION

Use bgntrim to mark the beginning of a definition for a trimming loop. Use endtrim to mark the end of a definition for a trimming loop. A trimming loop is a set of oriented curves (forming a closed curve) that defines boundaries of a NURBS surface. You include these trimming loop definitions in the definition of a NURBS surface.

The definition for a NURBS surface may contain many trimming loops. For example, if you wrote a definition for NURBS surface that resembled a rectangle with a hole punched out, the definition would contain two trimming loops. One loop would define the outer edge of the rectangle. The other trimming loop would define the hole punched out of the rectangle. The definitions of each of these trimming loops would be bracketed by a bgntrim/endtrim pair.

The definition of a single closed trimming loop may consist of multiple curve segments, each described as a piecewise linear curve (see pwlcurve) or as a single NURBS curve (see nurbscurve), or as a combination of both in any order. The only Graphics library calls that can appear in a trimming loop definition (between a call to bgntrim and a call to endtrim) are pwlcurve and nurbscurve.

In the following code fragment, we define a single trimming loop that consists of one piecewise linear curve and two NURBS curves:
bgntrim();
pwlccurve(. . .);
nurbsscurve(. . .);
nurbsscurve(. . .);
endtrim();

The area of the NURBS surface that the system displays is the region in the domain to the left of the trimming curve as the curve parameter increases. Thus, the resultant visible region of the NURBS surface is inside for a counter-clockwise trimming loop and outside for a clockwise trimming loop. So for the rectangle mentioned earlier, the trimming loop for the outer edge of the rectangle should run counter-clockwise, and the trimming loop for the hole punched out should run clockwise.

If you use more than one curve to define a single trimming loop, the curve segments must form a closed loop (i.e., the endpoint of each curve must be the starting point of the next curve, and the endpoint of the final curve must be the starting point of the first curve). If the endpoints of the curve are sufficiently close together but not exactly coincident, the system coerces them to match. If the endpoints are not sufficiently close, the system generates an error message and ignores the entire trimming loop.

If a trimming loop definition contains multiple curves, the direction of the curves must be consistent (i.e., the inside must be to the left of the curves). Nested trimming loops are legal as long as the curve orientations alternate correctly. If no trimming information is given for a NURBS surface, the entire surface is drawn.

SEE ALSO

bgnsurface, nurbssurface, nurbsscurve, pwlcurve, setnurbsproperty, getnurbsproperty
NAME

feedback, endfeedback – control feedback mode

C SPECIFICATION

Personal Iris and IRIS-4D VGX:
void feedback(buffer, size)
float buffer[];
long size;
long endfeedback(buffer)
float buffer[];

Other models:
void feedback(buffer, size)
short buffer[];
long size;
long endfeedback(buffer)
short buffer[];

PARAMETERS

buffer expects a buffer into which the system writes the feedback output from the Geometry Pipeline. On the Personal Iris and the IRIS-4D VGX, the output consists of 32-bit floating point values; on the other IRIS-4D models, the output consists of 16-bit integer values. Be sure you declare your buffer appropriately.

size expects the maximum number of buffer elements into which the system will write feedback output.

FUNCTION RETURN VALUE

The return value of endfeedback is the actual number of elements of buffer that were written. The system will not write more than size elements, even when the amount of feedback exceeds it. You should assume that overflow has occurred whenever the return value is size.
DESCRIPTION

feedback puts the system in feedback mode. In feedback mode, the system retains the output of the Geometry Pipeline rather than sending it to the rendering subsystem. endfeedback turns off feedback mode and returns the feedback output in buffer. This information is typically a description of a vertex, and is machine specific. For information for interpreting the returned buffer, see the “Feedback” chapter of the Graphics Library Programming Guide.

NOTE

These routines are available only in immediate mode.
NAME

finish – blocks until the Geometry Pipeline is empty

C SPECIFICATION

void finish()

PARAMETERS

none

DESCRIPTION

finish forces all unsent commands down the Geometry Pipeline to the rendering subsystem followed by a final token. It blocks the calling process until an acknowledgement is returned from the rendering subsystem that the final token has been received.

SEE ALSO

gflush

NOTE

This routine is available only in immediate mode.
NAME

fogvertex – specify fog density for per-vertex atmospheric effects

C SPECIFICATION

void fogvertex(mode, params)
long mode;
float params[];

PARAMETERS

mode expects one of three valid symbolic constants:

FG_DEFINE: interpret params as a specification for fog density and color.

FG_ON: enable the previously defined fog calculation

FG_OFF: disable fog calculations (default)

params Expects an array of floats containing value settings. For
FG_DEFINE four floats are expected. They are density, red,
green, and blue. density specifies the (thickness) of the fog (or
haze). A value of 0.0 results in no fog. Increasing positive
values result in fog of increasing density. Values are normalized
such that a density of 1.0 results in the fog becoming com-
pletely opaque at a distance of 1.0 in eye-coordinates. red,
green, and blue specify the fog color in the range 0.0 through
1.0.

DESCRIPTION

The effects of atmosphere on shading are simulated by blending com-
puted object colors into the specified atmosphere color. The blend ratio
is an exponential function of the distance from the eye to the object.
This ratio is computed at each point, line, or polygon vertex, then inter-
polated across lines and polygons (regardless of the value of shademo-
del).
Calculation of the blend factor at each vertex uses the following equation:

\[ V_{\text{fog}} = e^{\ (5.5 \times \text{density}\times Z_{\text{eye}})} \]

Where:

- \( V_{\text{fog}} \) is the computed fog blending factor, ranging from 0 to 1.
- \( \text{density} \) is the fog density as specified when you call \fogvertex\( \text{(FG\_DEFINE, params)} \).
- \( Z_{\text{eye}} \) is the \( Z \) coordinate in eye space (always negative).

Vertex colors are first either Gouraud or flat shaded, then textured, before being blended with fog color. The pixel color/fog color blend is done with the following equation:

\[ C = C_p \times V_{\text{fog}} + C_f \times (1.0 - V_{\text{fog}}) \]

Where:

- \( V_{\text{fog}} \) is the computed fog blending factor, ranging from 0 to 1.
- \( C \) is the resulting color component (red, green, or blue).
- \( C_p \) is the incoming pixel color, already either Gouraud or flat shaded, and textured.
- \( C_f \) is the fog color component as specified when \fogvertex\( \text{(FG\_DEFINE, params)} \) is called.

Eye-coordinates exist between ModelView transformation and Projection transformation (see \texttt{mmode}). This space is right-handed, so visible vertices always have negative \( Z \) coordinates. Thus the \( V_{\text{fog}} \) equation always raises \( e \) to a negative power.

The projection matrix must either be specified with a GL call (\texttt{perspective}, \texttt{window}, or \texttt{ortho}), or have as its final column the values:

\[
\begin{vmatrix}
0 \\
0 \\
-1 \\
0 \\
\end{vmatrix}
\]

In all cases (including \texttt{ortho}) the viewer is considered to be at location \( 0,0,0 \), looking down the negative \( z \) axis.
SEE ALSO

gRGBcolor, mmode

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support fog. Use getgdesc to determine whether fog support is available.

The results of fog calculations are defined only while in RGB mode.
NAME

font - selects a raster font for drawing text strings

C SPECIFICATION

void font(fntnum)
short fntnum;

PARAMETERS

fntnum expects the font identifier, an index into the font table built by defrasterfont. If you specify a font number that is not defined, the system selects font 0.

DESCRIPTION

font selects the raster font that charstr uses when it draws a text string. This font remains in effect until you call font again. Font 0 is the default.

SEE ALSO

charstr, defrasterfont, getdescender, getfont, getheight, strwidth
NAME

foreground – prevents a graphical process from being put into the background

C SPECIFICATION

void foreground()

PARAMETERS

none

DESCRIPTION

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

SEE ALSO

winopen

NOTE

This routine is available only in immediate mode.
NAME

freepup – deallocates a menu

C SPECIFICATION

    void freepup(pup)
    long pup;

PARAMETERS

    pup expects the menu identifier of the pop-up menu that you want to
deallocate.

DESCRIPTION

    freepup deallocates a pop-up menu, freeing the memory reserved for its
data structures.

SEE ALSO

    defpup, addtopup, dopup, newpup

NOTE

    This routine is available only in immediate mode.
NAME

backbuffer, frontbuffer – enable and disable drawing to the back or front buffer

C SPECIFICATION

void backbuffer(b)
  Boolean b;
void frontbuffer(b)
  Boolean b;

PARAMETERS

b expects either TRUE or FALSE.

TRUE enables updating in the back/front bitplane buffer.

FALSE turns off updating in the back/front bitplane buffer.

DESCRIPTION

The IRIS framebuffer is divided into four separate GL framebuffers: pop-up, overlay, underlay, and normal. Three of these framebuffers, overlay, underlay, and normal, can be configured in double buffer mode. When so configured, a framebuffer includes two color bitplane buffers: one visible bitplane buffer, called the front buffer, and one non-visible bitplane buffer, called the back buffer. The commands swapbuffers and mswapbuffers interchange the front and back buffer assignments.

By default, when a framebuffer is configured in double buffer mode, drawing is enabled in the back buffer, and disabled in the front buffer. frontbuffer and backbuffer enable and disable drawing into the front and back buffers, allowing the default to be overridden. It is acceptable to enable neither front nor back, either front or back, or both front and back simultaneously. Note, for example, that z-buffer drawing continues to update the z-buffer with depth values when neither the front buffer nor the back buffer is enabled for drawing.

frontbuffer and backbuffer state is maintained separately for each of the overlay, underlay, and normal framebuffers. Calls to these routines affect the framebuffer that is currently active, based on the current drawmode.
backbuffer is ignored when the currently active framebuffer is in single buffer mode. frontbuffer is also ignored when the currently active framebuffer is in single buffer mode, unless zdraw is enabled for that framebuffer (see zdraw).

After each call to gconfig, backbuffer is enabled and frontbuffer is disabled.

SEE ALSO
drawmode, doublebuffer, getbuffer, gconfig, singlebuffer, swapbuffers, zdraw

NOTE
Only VGX graphics support double buffer operation in the overlay and underlay framebuffers.
NAME

frontface – turns frontfacing polygon removal on and off

C SPECIFICATION

void frontface(b)
Boolean b;

PARAMETERS

b expects either TRUE or FALSE.

TRUE suppresses the display of frontfacing filled polygons.
FALSE allows the display of frontfacing filled polygons.

DESCRIPTION

frontface allows or suppresses the display of frontfacing filled polygons. If your programs represent solid objects as collections of polygons, you can use this routine to expose hidden surfaces. This routine works best for simple convex objects that do not obscure other objects.

A frontfacing polygon is defined as a polygon whose vertices are in counter-clockwise order in screen coordinates. When frontfacing polygon removal is on, the system displays only polygons whose vertices are in clockwise order. For complicated objects, this routine alone may not expose all hidden surfaces. To expose hidden surfaces for more complicated objects or groups of objects, your routine needs to check the relative distances of the object from the viewer (z values). (See “Hidden Surface Removal” in the Graphics Library Programming Guide.)

If frontface and backface are asserted simultaneously, no filled polygons will be displayed.

SEE ALSO

backface, zbuffer
NOTE

On IRIS-4D G and B models frontface does not work well when a polygon shrinks to the point where its vertices are coincident. Under these conditions, the routine cannot determine the orientation of the polygon and so displays the polygon by default.

On all IRIS-4D models 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 frontface to do the opposite of what is intended.
NAME

fudge — specifies fudge values that are added to a graphics window

C SPECIFICATION

void fudge(xfudge, yfudge)
long xfudge, yfudge;

PARAMETERS

xfudge expects the number of pixels added in the x direction.
yfudge expects the number of pixels added in the y direction.

DESCRIPTION

fudge specifies fudge values that are added to the dimensions of a graphics window when it is sized. Typically, you use it to create interior window borders. Call fudge prior to calling winopen.

fudge is useful in conjunction with stepunit and keepaspect. With stepunit the window size for integers m and n is:

\[ \text{width} = \text{xunit} \times m + \text{xfudge} \]
\[ \text{height} = \text{yunit} \times n + \text{yfudge} \]

With keepaspect the window size is \((\text{width}, \text{height})\), where:

\[ (\text{width} - \text{xfudge}) \times \text{yaspect} = (\text{height} - \text{yfudge}) \times \text{xaspect} \]

SEE ALSO

keepaspect, stepunit, winopen

NOTE

This routine is available only in immediate mode.
NAME

fullscrn — allows a program write to the entire screen

C SPECIFICATION

void fullscrn()

PARAMETERS

none

DESCRIPTION

fullscrn allows a program write to the entire screen. It does this by eliminating the protections that normally prevent a graphics process from drawing outside of its current window. fullscrn calls viewport(0, getgdesc(GD_XPMAIN)-1, 0, getgdesc(GD_YPMAIN)-1) and ortho2 to set up an orthographic projection that maps world coordinates to screen coordinates. The current viewport and matrix state are not saved; it is the caller’s responsibility to do this.

fullscrn only affects graphics output; input focus management is unchanged.

SEE ALSO

endfullscrn, winopen

NOTES

This routine is available only in immediate mode.

Use fullscrn with caution or a sense of humor.
NAME

gammaramp – defines a color map ramp for gamma correction

C SPECIFICATION

    void gammaramp(r, g, b)
    short r[256], g[256], b[256]

PARAMETERS

r  expects an array of 256 elements. Each element contains a setting for the red electron gun.

g  expects an array of 256 elements. Each element contains a setting for the green electron gun.

b  expects an array of 256 elements. Each element contains a setting for the blue electron gun.

DESCRIPTION

    gammaramp supplies a level of indirection for all color map and RGB values. For example, before the system would turn on the red gun to setting 238, the system looks in a table at location 238 and uses the value it finds there instead of 238.

Thus, you can use this table to provide gamma correction, to equalize monitors with different color characteristics, or to modify the color warmth of the monitor. The default setting has r[i] = g[i] = b[i] = i. (So at location 238 of the red, green, and blue tables, you find the value 238.)

When the system is in RGB mode and draws an object, the system writes the actual red, green, and blue values to the bitplanes not the indirect values. However, the values that you see when the system draws the bitmap to the screen are the indirect values: r[red], g[green], b[blue] (where r,g,b are the arrays last specified by gammaramp).

Similarly, when the system is in color map mode and draws an object, the system knows that the true color of the object may be color i, but to determine the displayed color, the system finds the red, green, and blue values of color i and displays color i as r[red], g[green], b[blue].
SEE ALSO

color, cmode, mapcolor, RGBcolor

NOTES

This routine is available only in immediate mode.

On the IRIS-4D G, gamma correction in RGB mode uses the top 256 entries of the colormap.
NAME

`ginit`, `gbegin` – create a window that occupies the entire screen

C SPECIFICATION

```c
void ginit()
void gbegin()
```

PARAMETERS

`none`

DESCRIPTION

`ginit` creates a window that covers the entire screen, and initializes its graphics state to the the same values as would a `winopen` followed by a `greset`. It also sets the `MOUSEX` valuator to `getgdesc(GD_XPMAX)/2` with range 0 to `getgdesc(GD_XPMAX)`, and sets the `MOUSEY` valuato to `getgdesc(GD_YPMAX)/2` with range 0 to `getgdesc(GD_YPMAX)/2`. `gbegin` does the same, except it does not alter the color map.

These routines are a carry-over from the days before there was a window manager. Although they continue function, we recommend that all new development be designed to work with the window manager and to use `winopen`.

SEE ALSO

`greset`, `winopen`

NOTE

These routines are available only in immediate mode.
NAME

gconfig – reconfigures the system

C SPECIFICATION

void gconfig()

PARAMETERS

none

DESCRIPTION

gconfig sets the modes that you request.

You must call gconfig for acsizel cmodel doublebufferl multimapl onemapl overlayl RGBmode, singlebufferl stensizel and underlay to take effect. After a gconfig call, color for each draw mode is set to zero, and writemask for each draw mode is set to the number of biplanes available in that draw mode. The contents of the color map do not change.

gconfig resolves mode requests for all draw modes, regardless of the current draw mode.

SEE ALSO

acsize, cmode, drawmode, doublebuffer, multimap, onemap, overlay, RGBmode, singlebuffer, stensize, underlay

NOTE

This routine is available only in immediate mode.
NAME

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

C SPECIFICATION

   Object genobj()

PARAMETERS

   none

FUNCTION RETURN VALUE

   The returned value for this function is an object identifier.

DESCRIPTION

   genobj generates unique 31-bit integer numbers for use as object
   identifiers. Object identifiers can be up to 31 bits and must be unique
   within a program. Be careful 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 ques-
   tion, use isobj before using your own numbers.

SEE ALSO

   callobj, gentag, isobj, makeobj

NOTE

   This routine is available only in immediate mode.
NAME

ggentag – returns a unique integer for use as a tag

C SPECIFICATION

Tag gentag()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value for this function is a tag number.

DESCRIPTION

ggentag generates a unique integer to use as a tag. Tags must be unique within an object. Although ggentag generates unique tags, if you later define a tag with the same value, the first tag is lost.

SEE ALSO

genobj, istag

NOTE

This routine is available only in immediate mode.
NAME
getbackface — returns whether backfacing polygons will appear

C SPECIFICATION
long getbackface()

PARAMETERS
none

FUNCTION RETURN VALUE
The returned value for this function is either 0 or 1.
0 indicates that backfacing polygon removal is turned off.
1 indicates that backfacing polygon removal is enabled.

DESCRIPTION
getbackface returns the state of backfacing filled polygon removal mode. If backface removal is enabled, the system draws only those polygons that face the viewer.

SEE ALSO
backface

NOTE
This routine is available only in immediate mode.
NAME

getbuffer – indicates which buffers are enabled for writing

C SPECIFICATION

long getbuffer()

PARAMETERS

none

FUNCTION RETURN VALUE

Individual bits in the returned value indicate which buffers are enabled. The bits are named:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Buffer Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCKBUFFER</td>
<td>back buffer</td>
</tr>
<tr>
<td>FRNTBUFFER</td>
<td>front buffer</td>
</tr>
<tr>
<td>DRAWZBUFFER</td>
<td>zbuffer drawing</td>
</tr>
</tbody>
</table>

DESCRIPTION

getbuffer indicates which buffers are enabled for writing in double buffer mode.

SEE ALSO

backbuffer, doublebuffer, frontbuffer, zdraw

NOTE

This routine is available only in immediate mode.

The symbolic return values mentioned above are defined in <gl/get.h>.

Version 4.0

April 1990
NAME

getbutton – returns the state of a button

C SPECIFICATION

Boolean getbutton(num)
Device num;

PARAMETERS

num is the device number of the button you want to test.

FUNCTION RETURN VALUE

There are two possible return values for this function:

FALSE indicates that button num is up.

TRUE indicates that button num is down.

The return value is undefined if there was an error, e.g. num is not a button device.

DESCRIPTION

getbutton returns the state of button num.

NOTE

This routine is available only in immediate mode.
NAME

getcmmode – returns the current color map mode

C SPECIFICATION

Boolean getcmmode()

PARAMETERS

none

FUNCTION RETURN VALUE

There are two possible returned values for this function:

TRUE  indicates that onemap mode is active.
FALSE  indicates that multimap mode is active.

DESCRIPTION

getcmmode returns the current color map mode.

SEE ALSO

multimap, onemap

NOTE

This routine is available only in immediate mode.
NAME

getcolor – returns the current color

C SPECIFICATION

long getcolor()

PARAMETERS

none

FUNCTION RETURN VALUE

Returns an index into the color map.

DESCRIPTION

getcolor returns the current color for the current drawing mode. In NORMALDRAW, it is an index into the color map, and is meaningful in both single and double buffer modes. getcolor is ignored in RGB mode. In OVERDRAW mode, getcolor returns the color that is drawn into the overlay bitplanes, etc.

SEE ALSO

color, doublebuffer, drawmode, getmcolor, singlebuffer

NOTE

This routine is available only in immediate mode.
NAME

getcpos – returns the current character position

C SPECIFICATION

void getcpos(ix, iy)
short *ix, *iy;

PARAMETERS

ix  expects the pointer to the location at which to write the x coordinate
     of the current character position.

iy  expects the pointer to the location at which to write the y coordinate
     of the current character position.

DESCRIPTION

getcpos gets the current character position and writes it into the parameters. For purely historical reasons, the returned values are offset by the window origin; i.e. they are absolute screen coordinates.

SEE ALSO

charstr, cmov, getgpos

NOTE

This routine is available only in immediate mode.
NAME

gtcursor — returns the cursor characteristics

C SPECIFICATION

void gtcursor(index, color, wtm, b)
short *index;
Colorindex *color, *wtm;
Boolean *b;

PARAMETERS

index expects a pointer to the location into which the system writes the
index of the current cursor. The cursor index is an index into a
table of cursor bitmaps.

color is an obsolete parameter. It is retained for compatibility with
previous releases.

wtm is an obsolete parameter. It is retained for compatibility with
previous releases.

b expects a pointer to the location into which the system returns a
boolean indicating if the cursor in visible in the current window.

DESCRIPTION

gtcursor returns the index of the current cursor and a boolean value
indicating if the cursor is visible in the current window. (The cursor
will not be visible if cursoff has been called.)

SEE ALSO

curon, defcursor, setcursor

NOTE

This routine is available only in immediate mode.
NAME

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

C SPECIFICATION

Boolean getdcm()

PARAMETERS

none

FUNCTION RETURN VALUE

This function can return either of two possible values:
FALSE, indicating that the system is not in depth-cue mode.
TRUE, indicating that the system is in depth-cue mode.

DESCRIPTION

getdcm tells you whether or not the system is in depth-cue mode.

SEE ALSO

depthcue

NOTE

This routine is available only in immediate mode.
NAME

getdepth – obsolete routine

C SPECIFICATION

void getdepth(near, far)
Screencoord *near, *far;

PARAMETERS

near expects a pointer to the location into which the system should
write the distance of the near clipping plane.

far expects a pointer to the location into which the system should
write the distance of the far clipping plane.

DESCRIPTION

This routine is obsolete. It continues to function to provide backwards
compatibility, but only for depth values set with the obsolete routine
setdepth. It is not guaranteed to correctly return the depth values
passed to lsetdepth, even when they do not exceed 16 bits.

SEE ALSO

lsetdepth, setdepth

NOTE

This routine is available only in immediate mode.
NAME

getdescender — returns the character characteristics

C SPECIFICATION

long getdescender();

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the length (in pixels) of the longest descender in the current font.

DESCRIPTION

getdescender returns the maximum distance (in pixels) between the baseline of a character and the bottom of the bitmap for that character.

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

NOTE

This routine is available only in immediate mode.
NAME

getdev – reads a list of valuators at one time

C SPECIFICATION

void getdev(n, devs, vals)
long n;
Device devs[];
short vals[];

PARAMETERS

n expects the number of devices named in the devs array (no more than 128).

devs expects an array containing the device identifiers (device number constants, such as MOUSEX, BPADX, LEFTMOUSE, etc.) of the devices you want to read. This array can contain up to 128 devices.

vals expects the array into which you want the system to write the values read from the devices listed in the devs array. Each member in the vals array corresponds to a member of the devs array. Thus, the value at vals[3] was read from the device named in devs[3].

DESCRIPTION

getdev allows you to read as many 128 valuators and buttons (input devices) at one time.

SEE ALSO

getvaluator

NOTE

This routine is available only in immediate mode.
NAME

getdisplaymode – returns the current display mode

C SPECIFICATION

long getdisplaymode()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value for this function tells you which display mode is currently active.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Display Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMSINGLE</td>
<td>color map single buffer mode</td>
</tr>
<tr>
<td>DMDOUBLE</td>
<td>color map double buffer mode</td>
</tr>
<tr>
<td>DMRGB</td>
<td>RGB single buffer mode</td>
</tr>
<tr>
<td>DMRGBDOUBLE</td>
<td>RGB double buffer mode</td>
</tr>
</tbody>
</table>

DESCRIPTION

getdisplaymode returns the current display mode.

SEE ALSO

cmode, doublebuffer, RGBmode, singlebuffer

NOTE

This routine is available only in immediate mode.

The symbolic return values mentioned above are defined in <gl/get.h>.
NAME

getdrawmode – returns the current drawing mode

C SPECIFICATION

long getdrawmode()

PARAMETERS

none

FUNCTION RETURN VALUE

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Drawing Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMALDRAW</td>
<td>color planes</td>
</tr>
<tr>
<td>OVERDRAW</td>
<td>overlay planes</td>
</tr>
<tr>
<td>UNDERDRAW</td>
<td>underlay planes</td>
</tr>
<tr>
<td>PUPDRAW</td>
<td>pop-up planes</td>
</tr>
<tr>
<td>CURSORDRAW</td>
<td>cursor</td>
</tr>
</tbody>
</table>

DESCRIPTION

getdrawmode returns the current drawing mode. Use drawmode to set the drawing mode.

SEE ALSO

drawmode

NOTE

This routine is available only in immediate mode.
NAME
getfont – returns the current raster font number

C SPECIFICATION
long getfont()

PARAMETERS
none

FUNCTION RETURN VALUE
The returned value for this function is the index into the font table for
the current raster font.

DESCRIPTION
getfont returns the index of the current raster font.

SEE ALSO
defrasterfont, font

NOTE
This routine is available only in immediate mode.
NAME

getgdesc – gets graphics system description

C SPECIFICATION

    long getgdesc(inquiry)
    long inquiry;

PARAMETERS

    inquiry expects the characteristic about which you want to inquire.

FUNCTION RETURN VALUE

    The function returns the value of the requested characteristic, or -1, if
    the request is invalid or its value cannot be determined.

DESCRIPTION

    getgdesc allows you to inquire about characteristics of the currently
    selected screen. You can call getgdesc prior to graphics initialization.
    Therefore, its return values are unaltered by any commands issued after
    initialization.

    The symbolic names of the inquiries and their meanings are specified
    below:

Screen Boundary Inquiries

    GD_XMMAX
    GD_YMMAX
        Vertical and horizontal size of the screen in millimeters.

    GD_XPMAX
    GD_YPMAX
        Vertical and horizontal size of the screen in pixels.

    GD_ZMAX
    GD_ZMIN
        Maximum and minimum depth values that can be stored in the
        z-buffer of the normal framebuffer.
<table>
<thead>
<tr>
<th>Graphics Type</th>
<th>GD_ZMIN</th>
<th>GD_ZMAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL4D</td>
<td>-0x4000</td>
<td>0x3FFF</td>
</tr>
<tr>
<td>GL4DGT{,X}</td>
<td>0</td>
<td>0x7FFFFFF</td>
</tr>
<tr>
<td>GL4DPI{,2,T}</td>
<td>-0x800000</td>
<td>0x7FFFFFF</td>
</tr>
<tr>
<td>GL4DVGX</td>
<td>-0x800000</td>
<td>0x7FFFFFF</td>
</tr>
</tbody>
</table>

Framebuffer Depth Inquiries

**GD_BITS_ACBUF**
Number of bitplanes per color component in the hardware accumulation buffer, if one exists. Otherwise the number of bitplanes per color component in the software version of the accumulation buffer, if it is implemented. Otherwise 0.

**GD_BITS_ACBUF_HW**
Number of bitplanes per color component in the hardware accumulation buffer, if one exists. Otherwise 0.

**GD_BITS_CURSOR**
Number of bitplanes available in the cursor.

**GD_BITS_NORM,DBL_ALPHA**
Maximum number of bitplanes available in the normal framebuffer to store alpha in double buffered RGB mode.

**GD_BITS_NORM,DBL_CMODE**
Number of bitplanes available in the normal framebuffer to store the color index in double buffered color map mode.

**GD_BITS_NORM,DBL_MMAP**
Number of bitplanes available in the normal framebuffer to store the color index in double buffered multimap mode.

**GD_BITS_NORM,DBL_RED**
**GD_BITS_NORM,DBL_GREEN**
**GD_BITS_NORM,DBL_BLUE**
Number of bitplanes available in the normal framebuffer to store red, green, and blue in double buffered RGB mode. If any of these are 0, then double buffered RGB mode is not available.
GD_BITS_NORM_SNG_ALPHA
Maximum number of bitplanes available in the normal framebuffer to store alpha in single buffered RGB mode.

GD_BITS_NORM_SNG_CMODE
Maximum number of bitplanes available in the normal framebuffer to store the color index in single buffered color map mode.

GD_BITS_NORM_SNG_MMAP
Number of bitplanes available in the normal framebuffer to store the color index in single buffered multimap mode.

GD_BITS_NORM_SNG_RED
GD_BITS_NORM_SNG_GREEN
GD_BITS_NORM_SNG_BLUE
Number of bitplanes available in the normal framebuffer to store red, green, and blue in single buffered RGB mode. If any of these are 0, then single buffered RGB mode is not available.

GD_BITS_NORM_ZBUFFER
Maximum number of useful bitplanes in the z-buffer of the normal framebuffer. If 0, then there is no z-buffer.

GD_BITS_OVER_SNG_CMODE
Maximum number of bitplanes available in the overlay framebuffer to store the color index in single buffered color map mode.

GD_BITS_PUP_SNG_CMODE
Maximum number of bitplanes available in the popup framebuffer to store the color index in single buffered color map mode.

GD_BITS_STENCIL
Number of bitplanes available in the normal framebuffer for use as stencil bitplanes. 0 if stencil is not functional.

GD_BITS_UNDR_SNG_CMODE
Maximum number of bitplanes available in the underlay framebuffer to store the color index in single buffered color map mode.
Miscellaneous Inquiries

**GD_AFUCN**
1 if `afunction` is functional, 0 if it is not.

**GD_ALPHA_OVERUNDER**
1 if alpha bitplanes in the normal framebuffer can be allocated as color map bitplanes in the overlay or underlay framebuffers, 0 if they cannot.

**GD_BLEND**
1 if blending is supported in all framebuffers that support RGB mode, 0 otherwise. (See `blendfunc`.)

**GD_CIFRACT**
1 if fractional interpolation of color indices is supported in all framebuffers, 0 otherwise. (See `colorf`.)

**GD_CLIPPLANES**
1 if user-defined clipping planes are supported, 0 otherwise. (See `clipplane`.)

**GD_CROSSHAIR_CINDEX**
Color index whose color map entry controls the color of the cross-hair cursor.

**GD_DBG**
1 if the dial and button box routines are functional, 0 if they are not. Unlike most of the others, this inquiry is independent of the currently selected screen. (See `dbtext` and `setdblights`.)

**GD_DITHER**
1 if RGB mode pixels are dithered when rendering into a buffer with less than 24 bitplanes. 0 otherwise.

**GD_FOGVERTEX**
1 if `fogvertex` is functional, 0 if it is not.

**GD_FRAMEGRABBER**
1 if `readsource(SRC_FRAMEGRABBER)` is functional, 0 if it is not.

**GD_LIGHTING_TWOSIDE**
1 if the `TWOSIDE` lighting model attribute is functional, 0 if it is not. (See `Imdef`.)
GD_LINESMOOTH_CMODE
1 if antialiased lines are supported in the normal framebuffer in color map mode, 0 otherwise. (See linesmooth.)

GD_LINESMOOTH_RGB
1 if antialiased lines are supported in RGB mode in all framebuffers that support RGB mode, 0 otherwise. (See linesmooth.)

GD_LOGICOP
1 if logical operations are supported in all framebuffers, 0 otherwise. (See logicop.)

GD_NBLINKS
Maximum number of blinking color map entries on the selected screen. If the value is non-zero, it will be at least 20. (See blink.)

GD_NMMAPS
Number of smaller color maps available to the user in multimap mode. On some models, the highest-numbered color map is reserved for use by the system. (See setmap.)

GD_NSCRNS
Number of screens available on the system. Unlike most of the others, this inquiry is independent of the currently selected screen.

GD_NURBS_ORDER
Maximum order of a NURBS surface.

GD_NVERTEX_POLY
Maximum number of vertices in a single polygon. If there is no limit, then GD_NOLIMIT is returned.

GD_OVERUNDER_SHARED
1 if overlay and underlay planes are shared, 0 if both can be used simultaneously.

GD_PATSIZE_64
1 if 64×64 patterns are supported, 0 otherwise. (See defpattern.)
GD_PNTSMOOTH_CMODE
1 if antialiased points are supported in the normal framebuffer in color map mode, 0 otherwise. (See pntsmooth.)

GD_PNTSMOOTH_RGB
1 if antialiased points are supported in RGB mode in all framebuffers that support RGB mode, 0 otherwise. (See pntsmooth.)

GD_POLYMODE
1 if polymode is functional, 0 if it is not.

GD_POLYSMOOTH
1 if antialiased polygons are supported in RGB mode in all framebuffers that support RGB mode, 0 otherwise. (See polysmooth.)

GD_PUP_TO_OVERUNDER
1 if the popup bitplanes can be allocated as color map bitplanes in the overlay or underlay framebuffers, 0 if they cannot.

GD_READSOURCE
1 if readsoure sources SRC_AUTO, SRC_FRONT, and SRC_BACK are functional, 0 if they are not.

GD_READSOURCE_ZBUFFER
1 if readsource(SRC_ZBUFFER) is functional, 0 if it is not.

GD_SCRBOX
1 if scrbox is functional, 0 if it is not.

GD_SCRNTYPE
Type of the currently selected screen. Returns GD_SCRNTYPE_WM if there is window management on the screen or GD_SCRNTYPE_NOWM if there isn’t. There can be at most one window open on screens of the latter type.

GD_STEREO
1 if setmonitor(STR_RECT) is functional, 0 if it is not.

GD_SUBPIXEL_LINE
GD_SUBPIXEL_PNT
GD_SUBPIXEL_POLY
1 if subpixel positioned lines, points, and polygons (respectively) are supported in all framebuffers, 0 otherwise. (See subpixel.)
GD_TEXTPORT
1 if the textport routines are functional, 0 if they are not. Unlike
most of the others, this inquiry is independent of the currently
selected screen. (See textport.)

GD_TEXTURE
1 if texture mapping routines are functional, 0 if they are not.
(See texdef2d.)

GD_TIMERHZ
Frequency of graphics timer events.

GD_TRIMCURVE_ORDER
Maximum order of a trimming curve.

GD_WSYS
Type of window system running on the machine. Returns
GD_WSYS_4S if the 4Sight Window System is currently run-
ing or GD_WSYS_NONE if there is no window system
currently running. Unlike most of the others, this inquiry is
independent of the currently selected screen.

GD_ZDRAW_GEOM
GD_ZDRAW_PIXELS
1 if routines that render geometry and routines that render pixels
(respectively) will do it into the z-buffer when zdraw is TRUE,
0 if they do not.

Return Values
The following table result of each inquiry for each graphics type:

SEE ALSO

gversion

NOTES
This routine is available only in immediate mode.

To inquire about the screen on which the current window is displayed,
use the following sequence:
long savescrn;
...
savescrn = scrnselect(getwscrn());
val1 = getgdesc(inquiry1);
val2 = getgdesc(inquiry2);
...
scrnselect(savescrn);
NAME

getgpos – gets the current graphics position

C SPECIFICATION

void getgpos(fx, fy, fz, fw)
Coord *fx, *fy, *fz, *fw;

PARAMETERS

fx expects a pointer to the location into which you want the system to
write the x coordinate of the current graphics position.

fy expects a pointer to the location into which you want the system to
write the y coordinate of the current graphics position.

fz expects a pointer to the location into which you want the system to
write the z coordinate of the current graphics position.

fw expects a pointer to the location into which you want the system to
write the w coordinate of the current graphics position. The w value
is used when defining a three dimensional point in homogeneous
coordinates.

DESCRIPTION

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

SEE ALSO

getcpos

NOTE

This routine is available only in immediate mode.
NAME

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

C SPECIFICATION

long getheight()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the maximum height (in pixels) of a character in the current font.

DESCRIPTION

getheight returns the maximum height of the characters, in the current raster font. The height is defined as the number of pixels between the top of the tallest ascender (in characters such as f and h) and the bottom of the lowest descender (in characters such as y and p).

SEE ALSO

getdescender, getfont, strwidth

NOTE

This routine is available only in immediate mode.
NAME

gethitcode — returns the current hitcode

C SPECIFICATION

long gethitcode()

PARAMETERS

none

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, with one bit for each clipping plane:

<table>
<thead>
<tr>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>far</td>
<td>near</td>
<td>top</td>
<td>bottom</td>
<td>right</td>
<td>left</td>
</tr>
</tbody>
</table>

SEE ALSO

clearhitcode, gselect, pick

NOTES

This routine is available only in immediate mode.

The symbolic values for the hitcode bits shown above are defined in <gl/get.h>.

This routine only functions on IRIS-4D B and G models, and therefore we advise against its use in new development.
NAME

getlsbackup — has no function in the current system

C SPECIFICATION

Boolean getlsbackup()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the current state of linestyle backup mode.

DESCRIPTION

getlsbackup returns the current state of linestyle backup mode. TRUE, indicates that the final two pixels of a line segment are always colored. FALSE, the default, indicates that the linestyle determines whether the last two pixels are colored.

Use lsbackup to change the state of this mode.

SEE ALSO

lsbackup

NOTES

This routine is available only in immediate mode.

This routine only functions on IRIS-4D B and G models, and therefore we advise against its use in new development.
NAME

getsrepeat – returns the linestyle repeat count

C SPECIFICATION

long getsrepeat()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the repeat factor for the current linestyle.

DESCRIPTION

getsrepeat returns the current linestyle repeat factor. To set (or reset) the current linestyle repeat factor, call lsrepeat.

SEE ALSO

lsrepeat

NOTE

This routine is available only in immediate mode.
NAME

getlstyle – returns the current linestyle

C SPECIFICATION

long getlstyle()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the index into the linestyle table for the current linestyle.

DESCRIPTION

getlstyle returns the current linestyle.

SEE ALSO

deflinestyle, setlinestyle

NOTE

This routine is available only in immediate mode.
NAME
getlwidth – returns the current linewidth

C SPECIFICATION
long getlwidth()

PARAMETERS
none

FUNCTION RETURN VALUE
The returned value of this function is the current linewidth in pixels.

DESCRIPTION
getlwidth returns the current linewidth in pixels.

SEE ALSO
linewidth

NOTE
This routine is available only in immediate mode.
NAME

getmap – returns the number of the current color map

C SPECIFICATION

long getmap()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the number of the current color map.

DESCRIPTION

getmap returns the number of the current color map as set by setmap in multimap mode. In onemap mode, getmap returns zero.

SEE ALSO

multimap, onemap, setmap

NOTE

This routine is available only in immediate mode.
NAME

getmatrix – returns a copy of a transformation matrix

C SPECIFICATION

void getmatrix(m)
Matrix m;

PARAMETERS

m expects an array into which to copy a matrix.

DESCRIPTION

getmatrix copies a transformation matrix into a user-specified array. When mmode is MSINGLE, the matrix from the top of the single matrix stack is returned. When mmode is MVIEWING, the matrix from the top of the ModelView matrix stack is returned. When mmode is MPROJECTION, the projection matrix is returned. And when mmode is MTEXTURE, the texture matrix is returned.

getmatrix does not alter the state of the graphics system.

SEE ALSO

loadmatrix, mmode, multmatrix, popmatrix, pushmatrix

NOTE

This routine is available only in immediate mode.
NAME

getmcolor – gets a copy of the RGB values for a color map entry

C SPECIFICATION

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

PARAMETERS

i  expects an index into the color map

r  expects a pointer to the location into which you want to copy the red
value of the color at the color map index specified by i.

g  expects a pointer to the location into which you want to copy the
green value of the color at the color map index specified by i.

b  expects a pointer to the location into which you want to copy the blue
value of the color at the color map index specified by i.

DESCRIPTION

getmcolor gets the red, green, and blue components of a color map
entry and copies them to the specified locations.

SEE ALSO

drawmode, mapcolor, gRGBcolor

NOTE

This routine is available only in immediate mode.
NAME
getmmode – returns the current matrix mode

C SPECIFICATION
long getmmode()

PARAMETERS
none

FUNCTION RETURN VALUE
The returned value of this function is the current matrix mode. There are four possible values for this function.

MSINGLE indicates single matrix mode mode.
MPROJECTION indicates projection matrix mode.
MVVIEWING indicates viewing matrix mode.
MTEXTURE indicates texture matrix mode.

DESCRIPTION
getmmode returns the current matrix mode.

SEE ALSO
mmode

NOTE
This routine is available only in immediate mode.
NAME

getmonitor – returns the type of the current display monitor

C SPECIFICATION

long getmonitor()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the type of the current display monitor.

DESCRIPTION

getmonitor returns the type of the current display monitor. The possible return values are:

HZ30  30Hz interlaced monitor
HZ30_SG  30HZ noninterlaced with sync on green monitor
HZ60  60Hz noninterlaced monitor
NTSC  NTSC monitor
PAL  PAL or SECAM monitor
STR_RECT

monitor in stereo mode

SEE ALSO

getothermonitor, setmonitor, setvideo

NOTES

This routine is available only in immediate mode.

The symbolic return values mentioned above are defined in <gl/get.h>.
This function returns the value set previously by \textit{setmonitor}. It does not actually test the hardware.
NAME

getnurbsproperty – returns the current value of a trimmed NURBS surfaces display property

C SPECIFICATION

void getnurbsproperty(property, value)
long property;
float *value;

PARAMETERS

property expects the name of the property to be queried.
value expects pointer to the location into which the system should write the value of the named property.

DESCRIPTION

The display of NURBS surfaces can be controlled in different ways. The following is a list of the display properties that can be affected.

N_ERRORCHECKING: If value is 1.0, some error checking is enabled. If error checking is disabled, the system runs slightly faster. The default value is 0.0.

N_PIXEL_TOLERANCE: The value is the maximum length, in pixels, of egdes of polygons on the screen used to render trimmed NURBS surfaces. The default value is 50.0 pixels.

SEE ALSO

bgnsurface, nurbssurface, bgntrim, nurbscurve, pwlcurve,
setnurbsproperty

NOTE

This routine is available only in immediate mode.
NAME
getopenobj – returns the identifier of the currently open object

C SPECIFICATION
Object getopenobj()

PARAMETERS
none

FUNCTION RETURN VALUE
The returned value of this function is the object identifier of the currently open object. If no object is now open, the returned value is -1.

DESCRIPTION
getopenobj returns the number of the object that is currently open for editing.

NOTE
This routine is available only in immediate mode.
NAME

`getorigin` – returns the position of a graphics window

C SPECIFICATION

```c
void getorigin(x, y)
long *x, *y;
```

PARAMETERS

- `x` expects a pointer to the location into which the system should copy the `x` position (in pixels) of the lower left corner of the graphics window.
- `y` expects a pointer to the location into which the system should copy the `y` position (in pixels) of the lower left corner of the graphics window.

DESCRIPTION

`getorigin` returns the position (in pixels) of the lower-left corner of a graphics window. Call `getorigin` after graphics initialization.

SEE ALSO

- `winopen`

NOTE

This routine is available only in immediate mode.
NAME

gethermonitor — obsolete routine

C SPECIFICATION

long gethermonitor()

PARAMETERS

none

FUNCTION RETURN VALUE

The return value of this function indicates if the optional Composite Video and Genlock Board is installed in the system.

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use getvideo(CG_MODE) to determine if the optional Composite Video and Genlock Board is installed in the system.

SEE ALSO

getvideo

NOTE

This routine is available only in immediate mode.
NAME

getpattern – returns the index of the current pattern

C SPECIFICATION

long getpattern()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is an index into the table of available patterns.

DESCRIPTION

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

SEE ALSO

defpattern, setpattern

NOTE

This routine is available only in immediate mode.
NAME

getplanes – returns the number of available bitplanes

C SPECIFICATION

long getplanes()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the number of bitplanes available for drawing under the current drawmode.

DESCRIPTION

getplanes returns the number of bitplanes that are available for drawing under the current drawmode. When the drawmode is NORMALDRAW, the result also depends on the current buffer mode and whether or not multimap mode is active. When the drawmode is CURSORDRAW, getplanes always returns 0, since no direct drawing can be done into the cursor planes.

SEE ALSO

cmode, drawmode, doublebuffer, getgdesc, multimap, onemap, overlay, RGBmode, singlebuffer, underlay

NOTE

This routine is available only in immediate mode.
NAME

getport – obsolete routine

C SPECIFICATION

void getport(name)
String name;

PARAMETERS

name   expects the window title that is displayed on the left hand side of the title bar for the window.

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use its replacement, winopen.

SEE ALSO

winopen
NAME

gresetls – returns the state of linestyle reset mode

C SPECIFICATION

Boolean gresetls()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the current state of linestyle reset mode.

DESCRIPTION

gresetls returns the current state of linestyle reset mode. TRUE, indicates that the stippling of each segment of a line starts at the beginning of the linestyle pattern. FALSE, indicates that 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.

Use resetls to change the state of this mode.

SEE ALSO

resetls

NOTES

This routine is available only in immediate mode.

This routine only functions on IRIS-4D B and G models, and therefore we advise against its use in new development.
NAME

getscrbox — read back the current computed screen bounding box

C SPECIFICATION

void getscrbox(left, right, bottom, top)
long *left, *right, *bottom, *top;

PARAMETERS

left returns the window coordinate of the left-most pixel drawn
while scrbox has been tracking.

right returns the window coordinate of the right-most pixel drawn
while scrbox has been tracking.

bottom returns the window coordinate of the lowest pixel drawn while
scrbox has been tracking.

top returns the window coordinate of the highest pixel drawn while
scrbox has been tracking.

DESCRIPTION

getscrbox returns the current screen bounding box. scrbox is the com-
puted bounding box of all geometry (points, lines, polygons) in screen-
space. The hardware updates the four values each time geometry is
drawn (while scrbox is tracking).

If left is greater than right, or bottom is greater than top, nothing has
been drawn since scrbox was reset.

SEE ALSO

scrbox

NOTE

This routine is available only in immediate mode.
IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support \texttt{scrbox}, and therefore do not support \texttt{getscrbox}. Use \texttt{getgdesc} to determine whether \texttt{scrbox} is supported.
NAME

getscrmmask – returns the current screen mask

C SPECIFICATION

void getscrmmask(left, right, bottom, top)
Screencoord *left, *right, *bottom, *top;

PARAMETERS

left expects a pointer to a location into which the system should copy the x coordinate (in pixels) of the left side of the current screen mask.

right expects a pointer to a location into which the system should copy the x coordinate (in pixels) of the right side of the current screen mask.

bottom expects a pointer to a location into which the system should copy the y coordinate (in pixels) of the bottom side of the current screen mask.

top expects a pointer to a location into which the system should copy the y coordinate (in pixels) of the top side of the current screen mask.

DESCRIPTION

getscrmmask returns the screen coordinates of the current screen mask.

SEE ALSO

scrmmask, popviewport, pushviewport

NOTE

This routine is available only in immediate mode.
NAME

getshade – obsolete routine

C SPECIFICATION

long getshade()

PARAMETERS

none

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use its identical replacement, getcolor.

SEE ALSO

getcolor

NOTE

This routine is available only in immediate mode.
NAME

getsize — returns the size of a graphics window

C SPECIFICATION

void getsize(x, y)
long *x, *y;

PARAMETERS

x expects a pointer to the location into which the system should copy
the width (in pixels) of a graphics window.
y expects a pointer to the location into which the system should copy
the height (in pixels) of a graphics window.

DESCRIPTION

getsize gets the dimensions (in pixels) of the graphics window used by a
graphics program. Call getsize after winopen.

SEE ALSO

winopen

NOTE

This routine is available only in immediate mode.
NAME

getsm – returns the current shading model

C SPECIFICATION

long getsm()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function indicates which shading model is now active. There are two possible return values:

FLAT the system renders lines and filled polygons in a constant color.

GOURAUD the system renders lines and filled polygons with Gouraud shading.

DESCRIPTION

getsml returns the shading model that the system uses to render lines and filled polygons.

SEE ALSO

shademodel

NOTE

This routine is available only in immediate mode.
NAME

getvaluator – returns the current state of a valuator

C SPECIFICATION

long getvaluator(dev)
Device dev;

PARAMETERS

dev expects the identifier of the device (e.g., MOUSEX, BPADX, etc.)
from which you want to read.

FUNCTION RETURN VALUE

The returned value of this function is the value stored at the device
named by the dev parameter.

DESCRIPTION

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

SEE ALSO

getbutton, qdevice, tie

NOTE

This routine is available only in immediate mode.
NAME

setvideo, getvideo – set and get video hardware registers

C SPECIFICATION

void setvideo(reg, value)
long reg, value;

long getvideo(reg)
long reg;

PARAMETERS

reg    expects the name of the register to access.

value  expects the value which is to be placed into reg.

FUNCTION RETURN VALUE

The returned value of getvideo is the value read from register reg, or −1.
−1 indicates that reg is not a valid register or that you queried a video
register on a system without that particular board installed.

DESCRIPTION

setvideo sets the specified video hardware register to the specified value.
getvideo returns the value of the specified video hardware register.
Several different video boards are supported; the board names and regis-
ter identifiers are listed below.

Display Engine Board
DE_R1

CG2 Composite Video and Genlock Board
CG_CONTROL
CG_CPHASE
CG_HPHASE
CG_MODE
VP1 Live Video Digitizer Board
 VP_ALPHA
 VP_BRITE
 VP_CMD
 VP_CONT
 VP_DIGVAL
 VP_FBXORG
 VP_FBYORG
 VP_FGMODE
 VP_GBXORG
 VP_GBYORG
 VP_HBLANK
 VP_HEIGHT
 VP_HUE
 VP_MAPADD
 VP_MAPBLUE
 VP_MAPGREEN
 VP_MAPRED
 VP_MAPSRC
 VP_MAPSTROBE
 VP_PIXCNT
 VP_SAT
 VP_STATUS0
 VP_STATUS1
 VP_VBLANK
 VP_WIDTH

SEE ALSO
getmonitor, getothermonitor, setmonitor, videocmd

NOTES
These routines are available only in immediate mode.
The DE_R1 register is actually present only on the video board used in the IRIS-4D B, G, GT, and GTX models. It is emulated on all other models.
The Live Video Digitizer is available as an option for IRIS-4D GTX models only.

The symbolic constants named above are defined in the files <gl/cg2vme.h> and <gl/vp1.h>.
NAME

g getpidport — gets a copy of the dimensions of the current viewport

C SPECIFICATION

void getpidport(left, right, bottom, top)
SCREENCOORD *left, *right, *bottom, *top;

PARAMETERS

left expects a pointer to a location into which the system should copy the x coordinate (in pixels) of the left side of the current view port.

right expects a pointer to a location into which the system should copy the x coordinate (in pixels) of the right side of the current view port.

bottom expects a pointer to a location into which the system should copy the y coordinate (in pixels) of the bottom side of the current view port.

top expects a pointer to a location into which the system should copy the y coordinate (in pixels) of the top side of the current view port.

DESCRIPTION

g getpidport gets the dimensions of the current viewport and copies them to the locations specified as parameters. The current viewport is defined as the viewport at the top of the viewport stack.

SEE ALSO

popviewport, pushviewport, viewport

NOTE

This routine is available only in immediate mode.
NAME

getwritemask – returns the current writemask

C SPECIFICATION

long getwritemask()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is the current writemask for the current drawing mode.

DESCRIPTION

getwritemask returns the current writemask of the current drawing mode. Each bit in the writemask corresponds to an available bitplane. Thus, bit 2 describes bitplane 2 and so on. When a bit is set to zero in the writemask, the corresponding bitplane is read only. This routine is undefined in RGB mode.

SEE ALSO

RGBwritemask, writemask, drawmode

NOTE

This routine is available only in immediate mode.
NAME

getwscreen — returns the screen upon which the current window appears

C SPECIFICATION

long getwscreen()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned function value is the screen number upon which the current window appears.

DESCRIPTION

getwscreen gets the screen number of the current window.

NOTE

This routine is available only in immediate mode.
NAME

getzbuffer – returns whether z-buffering is on or off

C SPECIFICATION

Boolean getzbuffer()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is either TRUE or FALSE.
TRUE indicates that z-buffering is on.
FALSE indicates that z-buffering is off. (FALSE is the default.) For systems without the zbuffer option, this function always returns FALSE.

DESCRIPTION

getzbuffer returns the current value of the z-buffer flag.

SEE ALSO

lsetdepth zbuffer, zclear,

NOTE

This routine is available only in immediate mode.
NAME
gexit – exits graphics

C SPECIFICATION

void gexit()

PARAMETERS

none

DESCRIPTION
gexit closes all the windows of a process and then frees all Graphics Library data structures. Thereafter, the process can no longer call any routines that require the graphics to be initialized.

gexit does not alter the image on screens for which the getgdesc inquiry GD_SCRNTYPE returns GD_SCRNTYPE_NOWM.

SEE ALSO

getgdesc, winclose

NOTE

This routine is available only in immediate mode.
NAME
gflush — flushes the DGL client buffer

C SPECIFICATION
void gflush()

PARAMETERS
none

DESCRIPTION
gflush has no function in the Graphics Library, but is included to provide compatibility with Distributed Graphics Library (DGL).

SEE ALSO
finish
4Sight User’s Guide, “Using the GL/DGL Interfaces”.

NOTE
The DGL on the client buffers the output from most graphics routines for efficient block transfer to the server. The DGL version of gflush sends all buffered but untransmitted graphics data to the server. Certain graphics routines, notably those that return values, also flush the client buffer when they execute.
NAME

ginit, gbegin – create a window that occupies the entire screen

C SPECIFICATION

void ginit()
void gbegin()

PARAMETERS

none

DESCRIPTION

ginit creates a window that covers the entire screen, and initializes its graphics state to the the same values as would a winopen followed by a greset. It also sets the MOUSEX valuator to getgdesc(GD_XPMAX)/2 with range 0 to getgdesc(GD_XPMAX), and sets the MOUSEY valuator to getgdesc(GD_YPMAX)/2 with range 0 to getgdesc(GD_YPMAX)/2. gbegin does the same, except it does not alter the color map.

These routines are a carry-over from the days before there was a window manager. Although they continue function, we recommend that all new development be designed to work with the window manager and to use winopen.

SEE ALSO

greset, winopen

NOTE

These routines are available only in immediate mode.
NAME

glcompat – controls compatibility modes

C SPECIFICATION

void glcompat(mode, value)
long mode, value;

PARAMETERS

mode the name of the compatibility mode you want to change. The available modes are:

GLC_OLDPOLYGON controls the state of old-style polygon mode.

GLCZRANGEMAP controls the state of z-range mapping mode.

value the value you want to set for the specified compatibility mode.

DESCRIPTION

glcompat gives control over details of the graphics compatibility between IRIS-4D models.

Old-Style Polygon Mode (GLC_OLDPOLYGON)
By default, old-style polygon mode is 1. Setting it to 0 speeds up old-style drawing commands but the output is subtly different. See the “High-Performance Drawing” and “Old-Style Drawing” sections of the Graphics Library Programming Guide for further explanation of the two modes and their effects on various machines.

WARNING: some features added recently to the Graphics Library are not supported by old-style polygons. These features include texture mapping, fog, and polygon antialiasing. Use new-style polygon commands, or set GLC_OLDPOLYGON to 1, to insure correct operation of new rendering features.
This is a per-window mode.

Z-Range Mapping Mode (GLCZRANGEMAP)
When z-range mapping mode is 0, the domain of the z-range arguments to lsetdepth, lRGBRanget, and lshaderange depends on the graphics hardware. The minimum is the value returned by getgdesc(GD_ZMIN) and the maximum is the value returned by getgdesc(GD_ZMAX). When this mode is 1, these routines accept the range 0x0 to 0x7FFFFFFF; it is mapped to whatever range the graphics hardware supports.

In order to maintain backwards compatibility, the default GLCZRANGEMAP is 1 on IRIS-4D B and G models, and 0 on all others.

This is a per-process mode.

SEE ALSO
getgdesc, lRGBRangef, lsetdepth, lshaderange

NOTES
This routine is available only in immediate mode.

The state of old-style polygon mode is ignored on IRIS-4D B and G models.

BUG
GLCZRANGEMAP should be a per-window mode.
NAME
greset – resets graphics state

C SPECIFICATION

void greset()

PARAMETERS

none

DESCRIPTION
greset resets a portion of the graphics state of a window to its default.

See the following table for a listing of the state affected.
<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>backface mode</td>
<td>off</td>
</tr>
<tr>
<td>blinking</td>
<td>off</td>
</tr>
<tr>
<td>buffer mode</td>
<td>single</td>
</tr>
<tr>
<td>color</td>
<td>undefined</td>
</tr>
<tr>
<td>color map mode</td>
<td>one map</td>
</tr>
<tr>
<td>concave</td>
<td>off</td>
</tr>
<tr>
<td>cursor</td>
<td>0 (arrow)</td>
</tr>
<tr>
<td>depth range</td>
<td>$Z_{min}, Z_{max}$</td>
</tr>
<tr>
<td>depthcue mode</td>
<td>off</td>
</tr>
<tr>
<td>display mode</td>
<td>color map</td>
</tr>
<tr>
<td>drawmode</td>
<td>NORMALDRAW</td>
</tr>
<tr>
<td>font</td>
<td>0</td>
</tr>
<tr>
<td>linestyle</td>
<td>0 (solid)</td>
</tr>
<tr>
<td>linewidth</td>
<td>1 pixel</td>
</tr>
<tr>
<td>lsrepeat</td>
<td>1</td>
</tr>
<tr>
<td>pattern</td>
<td>0 (solid)</td>
</tr>
<tr>
<td>picking size</td>
<td>10×10 pixels</td>
</tr>
<tr>
<td>RGB color</td>
<td>undefined</td>
</tr>
<tr>
<td>RGB shaderange</td>
<td>undefined</td>
</tr>
<tr>
<td>RGB writemask</td>
<td>undefined</td>
</tr>
<tr>
<td>shademodel</td>
<td>GOURAUD</td>
</tr>
<tr>
<td>shaderange</td>
<td>0,7, $Z_{min}, Z_{max}$</td>
</tr>
<tr>
<td>viewport</td>
<td>entire screen</td>
</tr>
<tr>
<td>writemask</td>
<td>all planes enabled</td>
</tr>
<tr>
<td>zbuffer mode</td>
<td>off</td>
</tr>
</tbody>
</table>

Notes

- Font 0 is a Helvetica-like font.

- $Z_{min}$ and $Z_{max}$ are the minimum and maximum values that you can store in the z-buffer. These depend on the graphics hardware and are returned by `getgdesc(GD_ZMIN)` and `getgdesc(GD_ZMAX)`.

- On IRIS-4D B and G models, `greset` also sets `lsbackup(FALSE)` and `resetls(TRUE)`.

Version 4.0 - 2 - April 1990
**greset** loads a 2-D orthographic projection transformation on the matrix stack with left, right, bottom, and top set to the boundaries of the screen (not the current window). It also turns on the cursor.

**greset** loads certain entries in the color map, as follows:

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>RGB Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Red</td>
</tr>
<tr>
<td>0</td>
<td>BLACK</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>RED</td>
<td>255</td>
</tr>
<tr>
<td>2</td>
<td>GREEN</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>YELLOW</td>
<td>255</td>
</tr>
<tr>
<td>4</td>
<td>BLUE</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>MAGENTA</td>
<td>255</td>
</tr>
<tr>
<td>6</td>
<td>CYAN</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>WHITE</td>
<td>255</td>
</tr>
<tr>
<td>all others</td>
<td>unnamed</td>
<td></td>
</tr>
</tbody>
</table>

It loads the PUPDRAW color map with the following entries:

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>RGB Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Red</td>
</tr>
<tr>
<td>1</td>
<td>PUP_COLOR</td>
<td>255</td>
</tr>
<tr>
<td>2</td>
<td>PUP_BLACK</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>PUP_WHITE</td>
<td>255</td>
</tr>
</tbody>
</table>

It loads the CURSORDRAW color map with the following entries:

<table>
<thead>
<tr>
<th>Index</th>
<th>RGB Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Red</td>
</tr>
<tr>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>2</td>
<td>255</td>
</tr>
<tr>
<td>3</td>
<td>255</td>
</tr>
</tbody>
</table>

On systems that do not have a 2-plane cursor, only index 1 is loaded.

**SEE ALSO**

getgdesc
NOTES

This routine is available only in immediate mode.

greset sets the viewport and the projection transformation to values which assume that the current window occupies the entire screen, i.e. it was created via ginit or gbegin. If this is not the case, you will probably want to call reshapeviewport and load a different projection transformation after calling greset.

This routine remains a part of the Graphics Library for reasons of backwards compatibility only. We do not recommend the use of this routine in new development.
gRGBcolor – gets the current RGB color values

C SPECIFICATION

void gRGBcolor(red, green, blue)
short *red, *green, *blue;

PARAMETERS

red expects a pointer to the location into which you want the system to copy the current red value.

green expects a pointer to the location into which you want the system to copy the current green value.

blue expects a pointer to the location into which you want the system to copy the current blue value.

DESCRIPTION

gRGBcolor gets the current RGB color values and copies them into the parameters. The system must be in RGB mode when you call gRGBcolor.

SEE ALSO

RGBcolor, RGBmode, getmcolor

NOTE

This routine is available only in immediate mode.
NAME

gRGBcursor – obsolete routine

C SPECIFICATION

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

DESCRIPTION

This routine is obsolete. It continues to function only on IRIS-4D B and
G models to provide backwards compatibility. All new development
should use its replacement, getcursor.

SEE ALSO

getcursor

NOTE

This routine is available only in immediate mode.
NAME

gRGBmask – returns the current RGB writemask

C SPECIFICATION

void gRGBmask(redm, greenm, bluem)
short *redm, *greenm, *bluem;

PARAMETERS

redm expects a pointer to the location into which you want the system to copy the current red writemask value.
greenm expects a pointer to the location into which you want the system to copy the current green writemask value.
bluem expects a pointer to the location into which you want the system to copy the current blue writemask value.

DESCRIPTION

gRGBmask gets the current RGB writemask as three 8-bit masks and copies them into the parameters. gRGBmask places masks in the low order 8-bits of the locations. The system must be in RGB mode when this routine executes.

SEE ALSO

getwritemask, RGBwritemask

NOTE

This routine is available only in immediate mode.
NAME

gselect – puts the system in selecting mode

C SPECIFICATION

void gselect(buffer, numnam)
short buffer[];
long numnam;

PARAMETERS

buffer expects the buffer into which you want the system to save the contents of the names stack. A name is a 16-bit number, that you load on the name stack just before you called a drawing routine.

numnam expects the maximum number of names that you want the system to save.

DESCRIPTION

gselect turns on the selecting mode. When in selecting mode, the system notes when a drawing routine intersects the selecting region and writes the contents of the names stack to the specified buffer. If you push a name onto the names stack just before you call each drawing routine, you can record which drawing routines intersected the selecting region.

Use the current viewing matrix to define the selecting region.

gselect and pick are identical except gselect allows you to create a viewing matrix in selecting mode. To end select mode, call endselect.

SEE ALSO

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

NOTE

This routine is available only in immediate mode.
NAME
gsync — waits for a vertical retrace period

C SPECIFICATION
void gsync()

PARAMETERS
none

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
singlebuffer

NOTE
This routine is available only in immediate mode.
NAME

`gversion` — returns graphics hardware and library version information

C SPECIFICATION

```c
long gversion(v)
String v;
```

PARAMETERS

`v` expects a pointer to the location into which to copy a string. Reserve at least a 12 character buffer.

FUNCTION RETURN VALUE

There is no longer any use for the returned value of this function; it will always be zero.

DESCRIPTION

`gversion` fills the buffer, `v`, with a null-terminated string that specifies the graphics hardware type of the currently selected screen and the version number of Graphics Library.

<table>
<thead>
<tr>
<th>Graphics Type</th>
<th>String Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>B or G</td>
<td>GL4D—m.n</td>
</tr>
<tr>
<td>GT</td>
<td>GL4DGT—m.n</td>
</tr>
<tr>
<td>GTX</td>
<td>GL4DGTX—m.n</td>
</tr>
<tr>
<td>VGX</td>
<td>GL4DVGX—m.n</td>
</tr>
<tr>
<td>Personal Iris</td>
<td>GL4DPI2—m.n</td>
</tr>
<tr>
<td>Personal Iris with Turbo Graphics</td>
<td>GL4DPI2—m.n</td>
</tr>
<tr>
<td>Personal Iris (early serial numbers)</td>
<td>GL4DPI—m.n</td>
</tr>
</tbody>
</table>

`m` and `n` are the major and minor release numbers of the release to which the Graphics Library belongs.

`gversion` can be called prior to the first `winopen`.

SEE ALSO

scrmselect, winopen
uname(2) in the *Programmer's Reference Manual*.

**NOTES**

This subroutine is available only in immediate mode.

Early serial numbers of the Personal Iris do not support the complete Personal Iris graphics functionality.
NAME

iconsize – specifies the icon size of a window

C SPECIFICATION

void iconsize(x,y)
  long x, y;

PARAMETERS

x  expects the width (in pixels) for the icon.

y  expects the height (in pixels) for the icon.

DESCRIPTION

iconsize specifies the size (in pixels) of the window used to replace a stowed window. If a window has an icon size, the window manager will re-shape the window to be that size and send a REDRAWICONIC token to the graphics queue when the user stows that window. Your code can use an event loop to test for this token and can call graphics library subroutines to draw the icon for the stowed window. Windows without an icon size are handled by the window manager with the locally appropriate default behavior.

To assign a new window an icon size, call iconsize before you open the window. To give an existing window an icon size, use iconsize with winconstraints.

SEE ALSO

qdevice, winconstraints, winopen

NOTES

This routine is available only in immediate mode.

Any application using iconsize should also call qdevice to queue the tokens WINFREEZE and WINTHAW after opening the window.
NAME

icontitle — assigns the icon title for the current graphics window.

C SPECIFICATION

void icontitle(name)
String name;

PARAMETERS

name expects a pointer to the string containing the icon title.

DESCRIPTION

icontitle specifies the string displayed on an icon if the window manager
draws that window’s icon.

SEE ALSO

iconsiz
NAME

imakebackground — registers the screen background process

C SPECIFICATION

void imakebackground()

PARAMETERS

none

DESCRIPTION

imakebackground registers a process that maintains the screen background. Call it before winopen. The process should redraw the screen background each time it receives a REDRAW event.

SEE ALSO

winopen

NOTE

This routine is available only in immediate mode.
NAME

`initnames` – initializes the name stack

C SPECIFICATION

`void initnames();`

PARAMETERS

`none`

DESCRIPTION

`initnames` clears the name stack for picking and selecting.

SEE ALSO

`gselect, pick`
NAME

ismex – obsolete routine

C SPECIFICATION

Boolean ismex()

PARAMETERS

none

FUNCTION RETURN VALUE

This routine returns TRUE

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should test the return value of getgdesc(GD_WSYS) to determine what window system is running.

SEE ALSO

getgdesc

NOTE

This routine is available only in immediate mode.
NAME
  isobj – returns whether an object exists

C SPECIFICATION
  Boolean isobj(obj)
  Object obj;

PARAMETERS

  obj  expects the object identifier that you want to test.

FUNCTION RETURN VALUE

  There are two possible return values for this function:
  TRUE    indicates that object obj exists.
  FALSE   indicates that object obj does not exist.

DESCRIPTION

  isobj returns whether or not an object exists. If makeobj has been
called to create an object, and delobj has not been called to delete it,
isobj returns TRUE for it.

SEE ALSO

  delobj, genobj, istag, makeobj

NOTE

  This routine is available only in immediate mode.
NAME

isqueued —returns whether the specified device is enabled for queueing

C SPECIFICATION

Boolean isqueued(dev)
Device dev;

PARAMETERS

dev expects the identifier for the device you want to test (e.g., MOUSEX or BPADX).

FUNCTION RETURN VALUE

The returned value for this function is a boolean value:

TRUE indicates that dev is enabled for queueing.

FALSE indicates that dev is not enabled for queueing.

DESCRIPTION

isqueued returns whether or not the specified device is enabled for queueing.

SEE ALSO

qdevice, unqdevice, qread

NOTE

This routine is available only in immediate mode.
NAME

istag – returns whether a tag exists in the current open object

C SPECIFICATION

Boolean istag(t)
Tag t;

PARAMETERS

t  expects the tag identifier that you want to test.

FUNCTION RETURN VALUE

There are two possible return values for this function:

TRUE    indicates that tag \( t \) exists in the current open object.
FALSE   indicates that tag \( t \) does not exist in the current open object.
The return value is undefined if no object is currently open for editing.

DESCRIPTION

istag returns whether or not a tag is exists in the object currently open for editing. If maketag has been called to create a tag, and deltag has not been called to delete it, istag returns TRUE for it.

SEE ALSO

deltag, gentag, isobj, maketag

NOTE

This routine is available only in immediate mode.
NAME

keepaspect – specifies the aspect ratio of a graphics window

C SPECIFICATION

void keepaspect(x, y)
long x, y;

PARAMETERS

x expects the horizontal proportion of the aspect ratio.
y expects the vertical proportion of the aspect ratio.

DESCRIPTION

keepaspect specifies the aspect ratio of a graphics window. Call it at
the beginning of a graphics program. It takes effect when you call winopen. 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 win-
dow. You can also call keepaspect in conjunction with winconstraints
to modify the enforced aspect ratio after the window has been created.

SEE ALSO

fudge, winconstraints, winopen

NOTE

This routine is available only in immediate mode.
NAME

lampon, lampoff – control the keyboard display lights

C SPECIFICATION

void lampon(lamps)
Byte lamps;
void lampoff(lamps)
Byte lamps;

PARAMETERS

lamps expects a mask that specifies which lamps to manipulate. The
four low-order bits control the lamps labeled L1 through L4. If a
bit is set, then the corresponding keyboard lamp is turned on or
off.

DESCRIPTION

lampon turns on any combination of the four user-controlled lamps on
the keyboard and lampoff turns them off.

SEE ALSO

clkon, ringbell, setbell

NOTES

This routine is available only in immediate mode.
Future systems may not have these keyboard lights; therefore, we advise
against the use of these routines for new development.
NAME

linesmooth – specify antialiasing of lines

C SPECIFICATION

void linesmooth(mode)
unsigned long mode;

PARAMETERS

mode expects one of two values:

SML_OFF, defeats antialiasing of lines (default).

SML_ON enables antialiasing of lines. SML_ON can be
modified by either or both of two additional symbolic constants:

SML_SMOOTHER indicates that a higher quality filter should
be used during line drawing. This filter typically requires that
more pixels be modified, and therefore potentially reduces the
rate at which antialiased lines are rendered.

SML_ENDCORRECT indicates that the endpoints of
antialiased lines should be trimmed to the exact length specified
by the subpixel position of each line.

The constants SML_SMOOTHER and SML_ENDCORRECT are
specified with SML_ON by bitwise ORing them, or by adding them.
For example,

linesmooth(SML_ON + SML_SMOOTHER + SML_ENDCORRECT);

enables antialiased line drawing with the highest quality, and potentially
lowest performance, algorithm. These modifiers are hints, not direc-
tives, and are therefore ignored by systems that do not support the
requested feature.
DESCRIPTION

Antialiased lines can be drawn in both color map and RGB modes. `linesmooth` controls this capability. In both modes, for antialiased lines to draw properly:

- linewidth must be 1,
- linestyle must be 0xFFFF,
- Isrepeat must be 1.

For color map antialiased lines to draw correctly, a 16-entry colormap block (whose lowest entry location is a multiple of 16) must be initialized to a ramp between the background color (lowest index) and the line color (highest index). Before drawing lines, clear the area to the background color.

The linesmooth hardware replaces the least significant 4 bits of the current color index with bits that represent pixel coverage. Therefore, by changing the current color index (only the upper 8 bits are significant) you can select among many 16-entry color ramps, representing different colors and intensities. You can draw depthcued, antialiased lines in this manner.

The z-buffer hardware can be used to improve the quality of color map antialiased line images. Enabled in the standard depth-comparison mode, it ensures that lines nearer the viewer obscure more distant lines. Alternately, the z-buffer hardware can be used to compare color values by issuing:

```c
zbuffer(TRUE);
zsource(ZSRC_COLOR);
zfunction(ZF_GREATER);
```

Pixels are then replaced only by 'brighter' values, resulting in better intersections between lines drawn using the same ramp.

RGB antialiased lines can be drawn only on machines that support blending. For these lines to draw correctly, the blendfunction must be set to merge new pixel color components into the framebuffer using the incoming (source) alpha values. Incoming color components should always be multiplied by the source alpha (BF_SA). Current (destination) color components can be multiplied either by one minus the source alpha (BF_MSA), resulting in a weighted average blend, or by one (BF.ONE), resulting in color accumulation to saturation; issue:
blendfunction(BF_SA, BF_MSA); /* weighted average */

or

blendfunction(BF_SA, BF_ONE); /* saturation */

The linesmooth hardware scales incoming alpha components by an 8-bit computed coverage value. Therefore reducing the incoming source alpha results in transparent, antialiased lines.

RGB antialiased lines draw correctly over any background image. It is not necessary to clear the area in which they are to be drawn.

Both color map and RGB mode antialiased lines can be drawn with subpixel-positioned vertexes (see subpixel). In general, subpixel positioning of line vertexes results in higher quality but lower performance.

The modifier SML_SMOOTHER can be ORed or ADDED to the symbolic constant SML_ON when antialiased lines are enabled. When this is done, a higher quality and potentially lower performance filter is used to scan convert antialiased lines. SML_SMOOTHER is a hint, not a directive. Thus a higher quality filter is used only if it is available.

The modifier SML_ENDCORRECT can be ORed or ADDED to the symbolic constant SML_ON when antialiased lines are enabled. When this is done, the endpoints of antialiased lines are scaled to the exact length specified by their subpixel-positioned endpoints, rather than drawn to the nearest integer length. SML_ENDCORRECT is a hint, not a directive. Thus antialiased lines are drawn with corrected endpoints only if support is available in the hardware.

SEE ALSO
bgnline, blendfunction, deflinestyle, linewidth, lsrepeat, pntsmooth, setlinestyle, subpixel, v, zbuffer, zfunction, zsource

NOTES
This subroutine does not function on IRIS-4D B or G models.

IRIS-4D GT and GTX models, and the Personal Iris, do not support SML_SMOOTHER and SML_ENDCORRECT. Both hints are ignored on these systems.
IRIS-4D VGX models adjust the antialiasing filter for each line based on its slope when SML_SMOOTHER is requested. They support SML_ENDCORRECT only in RGB mode.

BUGS

On the IRIS-4D GT and GTX models ZSRC_COLOR z-buffering is supported only for non-subpixel positioned color map mode lines.

Before ZSRC_COLOR z-buffering is used on IRIS-4D GT and GTX models, bitplanes 12 through 23 must be explicitly cleared to zero. This must be done in RGB mode, with a code sequence such as:

```c
RGBmode();
doublebuffer();
gconfig();
frontbuffer(TRUE);
cpack(0);
clear();
cmode();
frontbuffer(FALSE);
gconfig();
```

The clear operation must be repeated only after bitplanes 12 through 23 are modified, which can result only from interaction with another window running in RGB mode.
NAME

linewidth – specifies width of lines

C SPECIFICATION

void linewidth(n)
short n;

PARAMETERS

n expects the width of the line. The width is measured in pixels.

DESCRIPTION

linewidth specifies the displayed width of a line. Mathematical lines have no width, but to display a line, you need to assign the line a width. As far as possible, the displayed line centers on the mathematical line. Because the pixels are arranged in a rectangular grid, only vertical and horizontal lines can have exactly the pixel width required.

SEE ALSO

setlinestyle

NOTE

On IRIS-4D models that support resetls, it must be set to TRUE to obtain reasonable results with line widths greater than one.
NAME

Imbind – selects a new material, light source, or lighting model

C SPECIFICATION

void Imbind(target, index)
short target, index;

PARAMETERS

target expects one of these symbolic constants: MATERIAL, BACKMATERIAL, LIGHT0, LIGHT1, LIGHT2, LIGHT3, LIGHT4, LIGHT5, LIGHT6, LIGHT7, or LMODEL.

index expects the name of a material (if target is MATERIAL or BACKMATERIAL), a light source (if target is one of LIGHT0 through LIGHT7), or a lighting model (if target is LMODEL). Name is the index passed to Imdef when the material, light source, or lighting model was defined.

DESCRIPTION

Lighting operation is controlled by eleven lighting resources, each of which has a symbolic constant as a name. Imbind binds a material, light source, or lighting model definition to one of these eleven lighting resources. Its first argument, target, takes the symbolic name of a lighting resource. Its second argument, index, takes the name of a lighting definition to be bound to that resource. index specifies a material definition if target is MATERIAL or BACKMATERIAL, a light source definition if target is LIGHT0 through LIGHT7, or a lighting model definition if target is LMODEL.

Two of these resources, MATERIAL and LMODEL, are special, in that they together determine whether lighting calculations are made or not. Lighting calculations are enabled when a material definition other than material 0 is bound to MATERIAL, and a lighting model definition other than model 0 is bound to LMODEL. When either MATERIAL is bound to material definition 0, or LMODEL is bound to lighting model definition 0, all lighting calculations are disabled.
Thus, for example, lighting is defined and enabled in the most primitive way by the following code sequence:

```c
lmdef (DEFMATERIAL, 1, 0, nullarray);
lmdef (DEFLMODEL, 1, 0, nullarray);
lmbind (MATERIAL, 1);
lmbind (LMODEL, 1);
```

This primitive lighting model is disabled efficiently by simple binding material 0 to `MATERIAL`.

```c
lmbind (MATERIAL, 0);
```

A lighting definition is unbound from a lighting resource only when another definition is bound to that resource. Changes made to a lighting definition while it is bound are effective immediately. By default all eleven lighting resources are bound to definition 0. If `lmbind` is passed a name that is not defined, definition 0 is bound to the specified lighting resource.

The eight light sources, named `LIGHT0` through `LIGHT7`, are enabled when bound to a light source definition other than 0. Light source positions are transformed by the current ModelView matrix when the source is bound. The object-coordinate position of the light source is maintained in the definition so that subsequent bindings are transformed from it, rather than from the previously transformed position. A light source definition cannot be bound to more than one lighting resource in a single window.

The default lighting model uses only a single material, namely the material definition that is bound to `MATERIAL`. Likewise, when a lighting model with `TWOSIDE` specified is bound, `MATERIAL` is used for both front and back facing polygons if `BACKMATERIAL` is bound to material definition 0. However, if a material definition other than 0 is bound to `BACKMATERIAL`, two-sided lighting uses `MATERIAL` for front-facing polygons and `BACKMATERIAL` for back-facing polygons. In all cases points, lines, and characters are lighted using `MATERIAL`.

Lighting models use only material and light properties that are appropriate to them. Other properties, such as color map mode properties while the current framebuffer is in RGB mode, are ignored.
SEE ALSO

Imcolor, Imdef, mmode, n, nmode

NOTES

Lighting requires that the matrix mode be multi-matrix. It does not operate correctly while mmode is MSINGLE.

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support two-sided lighting, and therefore do not support light resource BACK-MATERIAL.

It is a common error to bind a light source when an inappropriate ModelView matrix is on the stack. Be careful!
NAME

Imcolor – change the effect of color commands while lighting is active

C SPECIFICATION

void Imcolor(mode)
long mode;

PARAMETERS

mode the name of the mode to be used. Possible modes are:

LMC_COLOR, RGB color commands will set the current color. If a color is the last thing sent before a vertex the vertex will be colored. If a normal is the last thing sent before a vertex the vertex will be lighted. LMC_COLOR is the default mode.

LMC_EMISSION, RGB color commands will set the EMISSION color property of the current material.

LMC_AMBIENT, RGB color commands will set the AMBIENT color property of the current material.

LMC_DIFFUSE, RGB color commands will set the DIFFUSE color property of the current material. Alpha, the fourth color component specified by RGB color commands will set the ALPHA property of the current material.

LMC_SPECULAR, RGB color commands will set the SPECULAR color property of the current material.

LMC_AD, RGB color commands will set the DIFFUSE and AMBIENT color property of the current material. Alpha, the fourth color component specified by RGB color commands will set the ALPHA property of the current material.

LMC_NULL, RGB color commands will be ignored.

DESCRIPTION

Properties of the currently bound material can be changed by calls to Imdef. Because the data structure of the material must be modified by this operation, however, it is relatively slow to execute. Imcolor is provided to support fast and efficient changes to the current material as
maintained in the graphics hardware, without changing the definition of the currently bound material. Thus Imcolor changes are lost whenever a new material is bound.

The standard RGB color commands (RGBcolor, c, and cpack) are used to change material properties efficiently. Imcolor specifies which material property is to be affected by these commands. While lighting is not active color commands change the current color. Imcolor mode is significant only while lighting is on.

SEE ALSO

Imdef, Imbind, RGBcolor, c, cpack

NOTE

This routine is available only in immediate mode.

Imcolor allows changes only to the properties of MATERIAL, not to the properties of BACKMATERIAL.

While Imcolor is other than LMC_NULL or LMC_COLOR, and lighting is active, the results of lighting are undefined between the time that a material is bound and an RGB color command is issued.

While Imcolor is other than LMC_NULL or LMC_COLOR, and lighting is active, the results of lighting are undefined if an RGB color command is specified between an n command and the subsequent v command.
NAME

Imdef – defines or modifies a material, light source, or lighting model

C SPECIFICATION

void Imdef(deftype, index, np, props)
short deftype, index, np;
float props[];

PARAMETERS

deftype expects the category in which to create a new definition, or the
category of the definition to be modified. There are three
categories, each with its own symbolic constants:

DEFMATERIAL indicates that a material is being defined or
modified.

DEFLIGHT indicates that a light source is being defined or
modified.

DEFLMODEL indicates that a lighting model is being
defined or modified.

index expects the index into the table of stored definitions. There is a
unique definitions table for each category of definition created
by this routine (materials, light sources, or lighting models).
Indexes within each of these categories are independent. In
each category, index 0 is reserved as a null definition, and can-
not be redefined.

np expects the number of symbols and floating point values in
props, including the termination symbol LMNULL. If np is
zero, it is ignored. Operation over network connections is
more efficient when np is correctly specified, however.

props expects the array of floating point symbols and values that
define, or modify the definition of, the material, light source, or
lighting model named index. props must contain a sequence of
lighting symbols, each followed by the appropriate number of
floating point values. The last symbol must be LMNULL,
which is itself not followed by any values.
Different symbols are used to define materials, light sources, and lighting models. The symbols used when deftype is DEFMATERIAL are:

ALPHA specifies the transparency of the material. It is followed by a single floating point value in the range 0.0 through 1.0. This alpha value is assigned to all RGB triples generated by the lighting model. Alpha is ignored by systems that do not support blending, and is always valid in systems that do, regardless of whether alpha bit-planes are installed in the system. The default alpha value is 1.0.

AMBIENT specifies the ambient reflectance of the material. It is followed by three floating point values, typically in the range 0.0 through 1.0, specifying red, green, and blue reflectances. The default ambient reflectances are 0.2, 0.2, and 0.2.

COLORINDEXES specifies the material properties used when lighting in color map mode. This property is ignored while the current framebuffer is in RGB mode, as are most other material properties when the current framebuffer is in color map mode. (Material property SHININESS is used in color map mode.) It is followed by three floating point values, assigning the ambient, diffuse, and specular material color indices. The default color indices are 0.0, 127.5, and 255.0.

DIFFUSE specifies the diffuse reflectance of the material. It is followed by three floating point values, typically in the range 0.0 through 1.0, specifying red, green, and blue diffuse reflectances. The default diffuse reflectances are 0.8, 0.8, and 0.8.

EMISSION specifies the color of light emitted by the material. It is followed by three floating point values, typically in the range 0.0 through 1.0, specifying red, green, and blue emitted light levels. The default emission levels are 0.0, 0.0, and 0.0.

SHININESS specifies the specular scattering exponent, or the shininess, of the material. It is followed by a single floating point value in the range 0.0 through 128.0. Higher values result in smaller, hence more shiny, specular highlights. The default shininess is 0.0, which effectively disables specular reflection.
SPECULAR specifies the specular reflectance of the material. It is followed by three floating point values, typically in the range 0.0 through 1.0, specifying red, green, and blue specular reflectances. The default specular reflectances are 0.0, 0.0, and 0.0.

The symbols used when dectype is DEFLIGHT are:

AMBIENT specifies the ambient light associated with the light source. It is followed by three floating point values, typically in the range 0.0 through 1.0, specifying red, green, and blue ambient light levels. The default ambient levels are 0.0, 0.0, and 0.0.

LCOLOR specifies the color and intensity of the light that is emitted from the light source. It is followed by three floating point values, typically in the range 0.0 through 1.0, specifying the levels of red, green, and blue light emitted from the light source. The default light colors are 1.0, 1.0, and 1.0.

POSITION specifies the position of the light source in the scene. It is followed by four floating point values, specifying x, y, z, and w of the light source position in object-coordinates. If w is specified as 0.0, the light source is taken to be infinitely distant from the origin of the coordinate system. In this case x, y, and z specify the direction from the origin to the infinitely distant light source. There is typically a performance penalty associated with light sources that are not infinitely distant. The default light source position is at infinity directly behind the viewpoint, location (0.0, 0.0, 1.0, 0.0). The location of the light source is transformed by the current ModelView matrix when the source is bound (see Imbind).

SPOTTARGET specifies the direction that a spot light source emits its light. It is followed by three floating point values, specifying x, y, and z direction vectors in object-coordinates. These vectors are normalized automatically. The direction is ignored if the light source is not a spot light. By default the spot light points down the negative z axis, direction (0.0, 0.0, -1.0). The direction is transformed by the current ModelView matrix when the light source is bound (see Imbind).

SPOTLIGHT indicates that the light source is to be treated as a spot light. (The light source must not be positioned at infinity.) It is followed by two floating point values, specifying the exponent and the spread of the light cone. exponent controls intensity as a
function of angle from the spot light direction. Its range is 0.0 through 128.0, where 0.0 results in constant intensity throughout the cone, and 128.0 results in the sharpest dropoff of intensity as angle from spot direction increases. \textit{spread} is the angle in degrees, measured from the spot light direction, beyond which the cone is attenuated to zero intensity. Currently only \textit{spread} values in the range 0.0 through 90.0, and the special value 180.0, are supported. By default \textit{exponent} is 0.0 and \textit{spread} is 180.0, effectively disabling spot lighting.

The symbols used when \textit{deftype} is \textit{DEFLMODEL} are:

\textbf{AMBIENT} specifies an additional ambient light level that is associated with the entire scene, rather than with a light source. This light is added to the ambient light associated with each light source to yield the total ambient light in the scene. \textbf{AMBIENT} is followed by three floating point values, typically in the range 0.0 through 1.0, specifying red, green, and blue ambient light levels. The default lighting model ambient light levels are 0.2, 0.2, and 0.2.

\textbf{ATTENUATION} specifies the constant and linear attenuation factors associated with all non-infinite light sources. It is followed by two floating point values in the range 0.0 through positive infinity. The first attenuation factor is used to directly reduce the effect of a light source on objects in the scene. The second factor specifies attenuation that is proportional to the distance of the light source from the object being lighted. The default constant and linear attenuations are 1.0 and 0.0, effectively disabling constant and linear attenuation.

\textbf{ATTENUATION2} specifies the second-order attenuation factor associated with all non-infinite light sources. It is followed by a single floating point value in the range 0.0 through positive infinity. This factor specifies attenuation that is proportional to the square of the distance of the light source from the object being lighted. The default second-order attenuation is 0.0, effectively disabling second-order attenuation.

\textbf{LOCALVIEWER} specifies whether reflection calculations are done based on a local or infinitely distant viewpoint. It is followed by a single floating point value, which must be either 0.0 or 1.0. The value 0.0 indicates that the viewpoint is to be \textbf{(0.0, 0.0, infinity)}
in eye-coordinates, hence infinitely distant from all objects in the scene. The value 1.0 indicates that the viewpoint is to be (0.0, 0.0, 0.0) in eye-coordinates, hence local. There is typically a performance penalty associated with a local viewpoint. By default the viewpoint is infinitely distant.

**TWOSIDE** specifies whether lighting calculations are done assuming that only frontfacing polygons are visible, or are corrected for each polygon based on whether it is frontfacing or backfacing. It is followed by a single floating point value, which must be either 0.0 or 1.0. The value 0.0 specifies a lighting model that is correct only for polygons whose visible face is the facet for which normals have been provided. The value 1.0 specifies a lighting model that is correct for both frontfacing and backfacing polygons. When **TWOSIDE** is 1.0, vertex normals are reversed (all components multiplied by -1.0) for all vertices of backfacing polygons. Thus, for two-sided lighting to operate correctly, normals must be specified for the facet whose screen rotation is counter-clockwise (i.e. for frontfacing facets). If a material is bound to **BACKMATERIAL**, this material is used to light backfacing polygons (see **Imbind**). Otherwise, both frontfacing and backfacing polygons are lighted using **MATERIAL**.

Lighting calculations for all primitives other than polygons, such as points, lines, and characters, are unaffected by the **TWOSIDE** flag. By default two-sided lighting is disabled.

**DESCRIPTION**

**Imdef** either defines a new material, light source, or lighting model, or modifies the definition of a currently defined material, light source, or lighting model. **deftype** specifies whether a material, light source, or lighting model is being defined or modified. **index** is the name of the material, light source, or lighting model. **props** is a list of attribute tokens, each followed by one or more floating point values, that initializes or modifies the definition. The last attribute token in the array must be **LMNULL**.
When `Imdef` is first called with a particular `deftype/index` combination, the material, light source, or lighting model of name `index` is created with the attributes specified in `props`, and with default values for all attributes that are not included in `props`. Subsequent `Imdef` calls with a `deftype/index` combination modify only the attributes included in `props`. Prior to the first `Imdef` call, `deftype/index` combinations are undefined, and cannot be bound (see `Imbind`). A definition can be reset to all default attributes by passing its `deftype` and `index` to `Imdef` with a null attribute list (`props` contains only `LMNULL`). Changes made to a currently bound definition are effective immediately.

Lighting calculations are done in a numeric space where 0.0 is black (no light) and 1.0 is white (the brightest displayable light). Color attributes are specified in this numeric space, although values outside the range 0.0 through 1.0 are allowed. Lighting specified using `Imdef` occurs only at the vertices of points, lines, polygons, and at the origin of text strings. At each vertex the contributions of each light source are summed with the material emitted light and the lighting model ambient (scaled by the material ambient reflectance) to yield the vertex color. After each lighting calculation is completed, the computed color components are clamped to a maximum value of 1.0, then scaled by 255.0 prior to interpolation in the framebuffer.

The contribution of each light source is the sum of:

- light source ambient color, scaled by material ambient reflectance,
- light source color, scaled by material diffuse reflectance and the dot product of the vertex normal and the vertex-to-light source vector, and
- light source color, scaled by material specular reflectance and a function of the angle between the vertex-to-viewpoint vector and the vertex-to-light source vector.

Material 0, light source 0, and lighting model 0 are null definitions that cannot be changed.

The default material is defined as:
ALPHA 1.0
AMBIENT 0.2, 0.2, 0.2
COLORINDEXES 0.0, 127.5, 255.0
DIFFUSE 0.8, 0.8, 0.8
EMISSION 0.0, 0.0, 0.0
SHININESS 0.0
SPECULAR 0.0, 0.0, 0.0

The default light source is defined as:
AMBIENT 0.0, 0.0, 0.0
LCOLOR 1.0, 1.0, 1.0
POSITION 0.0, 0.0, 1.0, 0.0
SPOTDIRECTION 0.0, 0.0, -1.0
SPOTLIGHT 0.0, 180.0

The default lighting model is defined as:
AMBIENT 0.2, 0.2, 0.2
ATTENUATION 1.0, 0.0
ATTENUATION2 0.0
LOCALVIEWER 0.0
TWOSIDE 0.0

SEE ALSO
Imbind, Imcolor, mmode, n, nmode

NOTES
This routine is available only in immediate mode.

Lighting requires that the matrix mode be multi-matrix. It does not operate correctly while mmode is MSINGLE.

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support light source attributes SPOTLIGHT or SPOTDIRECTION, or lighting model attributes ATTENUATION2 or TWOSIDE. Use getgdesc to determine which lighting model attributes are supported.
BUGS

The results of lighting calculations are clamped to a maximum value of 1.0, but not to a minimum value of 0.0. If only positive color attributes are specified in the active material, light sources, and lighting model, the computed color cannot have negative components. However, if negative attributes are specified, care must be taken to not produce negative results.

Many attributes are not used by the current color map lighting model. Some may be used in future releases.
NAME

loadmatrix – loads a transformation matrix

C SPECIFICATION

void loadmatrix(m)
Matrix m;

PARAMETERS

m  expects the matrix which is to be loaded onto the matrix stack.

DESCRIPTION

loadmatrix loads a 4x4 floating point matrix onto the transformation stack, replacing the current top matrix.

SEE ALSO

getmatrix, multmatrix, popmatrix, pushmatrix
NAME

loadname — loads a name onto the name stack

C SPECIFICATION

void loadname(name)
short name;

PARAMETERS

name expects the name which is to be loaded onto the name stack.

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. This enables the user to quickly identify the part of an image that appears near the cursor.

SEE ALSO

gselect, pick
NAME

logicop – specifies a logical operation for pixel writes

C SPECIFICATION

void logicop(opcode)
long opcode;

PARAMETERS

opcode expects one of the 16 possible logical operations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO_ZERO</td>
<td>0</td>
</tr>
<tr>
<td>LO_AND</td>
<td>src AND dst</td>
</tr>
<tr>
<td>LO_ANDR</td>
<td>src AND (NOT dst)</td>
</tr>
<tr>
<td>LO_SRC</td>
<td>src</td>
</tr>
<tr>
<td>LO_ANDI</td>
<td>(NOT src) AND dst</td>
</tr>
<tr>
<td>LO_DST</td>
<td>dst</td>
</tr>
<tr>
<td>LO_XOR</td>
<td>src XOR dst</td>
</tr>
<tr>
<td>LO_OR</td>
<td>src OR dst</td>
</tr>
<tr>
<td>LO_NOR</td>
<td>NOT (src OR dst)</td>
</tr>
<tr>
<td>LO_XNOR</td>
<td>NOT (src XOR dst)</td>
</tr>
<tr>
<td>LO_NDST</td>
<td>NOT dst</td>
</tr>
<tr>
<td>LO_ORR</td>
<td>src OR (NOT dst)</td>
</tr>
<tr>
<td>LO_NSRC</td>
<td>NOT src</td>
</tr>
<tr>
<td>LO_ORI</td>
<td>(NOT src) OR dst</td>
</tr>
<tr>
<td>LO_NAND</td>
<td>NOT (src AND dst)</td>
</tr>
<tr>
<td>LO_ONE</td>
<td>1</td>
</tr>
</tbody>
</table>

Only the lower 4 bits of opcode are used.

The values of LO_SRC and LO_DST have been chosen so that expressing an operation as the equivalent combination of them and the C bitwise operators generates an acceptable opcode value; e.g., LO_NAND can be written as ~(LO_SRC & LO_DST).
DESCRIPTION

**logicop** specifies the bit-wise logical operation for pixel writes. The logical operation is applied between the source pixel value (incoming value) and existing destination value (previous value) to generate the final pixel value. In colorindex mode all of the (up to 12) writemask enabled index bits are changed. In RGB mode all of the (up to 32) enabled component bits are changed.

**logicop** defaults to LO_SRC, meaning that the incoming source value simply replaces the current (destination) value.

It is not possible to do logical operations and blend simultaneously. When opcode is set to any value other than LO_SRC, the blendfunction *sfactr* and *dfactr* values are forced to BF_ONE and BF_ZERO respectively (their default values). Likewise, calling **blendfunction** with arguments other than BF_ONE and BF_ZERO forces the logical opcode to LO_SRC,

Unlike the blendfunction, **logicop** is valid in all drawing modes (NORMALDRAW, UNDERDRAW, OVERDRAW, PUPDRAW, CURSORDRAW) and in both colorindex and RGB modes. Like the blendfunction, it affects all drawing operations, including points, lines, polygons, and pixel area transfers.

SEE ALSO

blendfunction, gversion

NOTES

The numeric assignments of the 16 operation names were chosen to be identical to those defined by the X Window System. They will not be changed in future software releases.

This routine does not function on IRIS-4D B, G, GT, and GTX models, nor does it function on early serial numbers of the Personal Iris. Use **gversion** to determine which type you have.
NAME

lookat – defines a viewing transformation

C SPECIFICATION

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

PARAMETERS

vx expects the x coordinate of the viewing point.
vy expects the y coordinate of the viewing point.
vz expects the z coordinate of the viewing point.
px expects the x coordinate of the reference point.
py expects the y coordinate of the reference point.
pz expects the z coordinate of the reference point.
twist expects the angle of rotation.

DESCRIPTION

lookat defines the viewpoint and a reference point on the line of sight in world coordinates. The viewpoint is at \((vx, vy, vz)\), and is the position from which you are looking. The reference point is at \((px, py, pz)\), and is the location on which the viewpoint is centered. The viewpoint and reference point define the line of sight. twist measures right-hand rotation about the line of sight.

The matrix computed by lookat premultiplies the current matrix, which is chosen based on the current matrix mode.

SEE ALSO

mmode, polarview
NAME

rectread, lrectread – reads a rectangular array of pixels into CPU memory

C SPECIFICATION

long rectread(x1, y1, x2, y2, parray)
Screencoord x1, y1, x2, y2;
Colorindex parray[];

long lrectread(x1, y1, x2, y2, parray)
Screencoord x1, y1, x2, y2;
unsigned long parray[];

PARAMETERS

x1 expects the x coordinate of the lower-left corner of the rectangle that you want to read.
y1 expects the y coordinate of the lower-left corner of the rectangle that you want to read.
x2 expects the x coordinate of the upper-right corner of the rectangle that you want to read.
y2 expects the y coordinate of the upper-right corner of the rectangle that you want to read.
parray expects the array to receive the pixels that you want to read.

FUNCTION RETURN VALUE

The returned value of this function is the number of pixels specified in the rectangular region, regardless of whether the pixels were actually readable (i.e. on-screen) or not.

DESCRIPTION

rectread and lrectread read the pixel values of a rectangular region of the screen and write them to the array, parray. The system fills the elements of parray from left-to-right, then bottom-to-top. All coordinates are relative to the lower-left corner of the window, not the screen or viewport.
rectread fills an array of 16-bit words, and therefore should be used only to read color index values. lrectread fills an array of 32-bit words. Based on the current pixmode, lrectread can return pixels of 1, 2, 4, 8, 12, 16, 24, or 32 bits each. Use it to read packed RGB or RGBA values, color index values, or z values. Use readssource to specify the pixel source from which both rectread and lrectread read pixels.

pixmode greatly affects the operation of lrectread, and has no effect on the operation of rectread. By default, lrectread returns 32-bit pixels in the format used by cpack. Different pixel sizes, framebuffer shifts, scan patterns through the framebuffer, and strides through memory, can all be specified using pixmode.

rectread and lrectread leave the current character position unpredictable.

SEE ALSO
lrectwrite, pixmode, readssource

NOTES

These routines are available only in immediate mode.

On IRIS-4D GT and GTX models, returned bits that do not correspond to valid bitplanes are undefined. Other models return zero in these bits.

On IRIS-4D GT, GTX, and VGX models, rectread performance will suffer if $x2 - x1 + 1$ is odd, or if parray is not 32-bit word aligned.
NAME

rectwrite, lrectwrite – draws a rectangular array of pixels into the frame buffer

C SPECIFICATION

void rectwrite(x1, y1, x2, y2, pararray)
Screncoord x1, y1, x2, y2;
Colorindex pararray[];

void lrectwrite(x1, y1, x2, y2, pararray)
Screncoord x1, y1, x2, y2;
unsigned long pararray[];

PARAMETERS

x1 expects the lower-left x coordinate of the rectangular region.
y1 expects the lower-left y coordinate of the rectangular region.
x2 expects the upper-right x coordinate of the rectangular region.
y2 expects the upper-right y coordinate of the rectangular region.
pararray expects the array which contains the values of the pixels to be drawn. For RGBA values, pack the bits thusly: 0xAAABBGGRR, where:

    AA contains the alpha value,
    BB contains the blue value,
    GG contains the green value, and
    RR contains the red value.

RGBA component values range from 0 to 0xFF (255). The alpha value will be ignored if blending is not active and the machine has no alpha bitplanes.

DESCRIPTION

rectwrite and lrectwrite draw pixels taken from the array pararray into the specified rectangular frame buffer region. The system draws pixels left-to-right, then bottom-to-top. All coordinates are relative to the lower-left corner of the window, not the screen or viewport. All normal drawing modes apply.
The size of \textit{parray} is always \((x2-x1+1) \times (y2-y1+1)\). If the zoom factors set by \texttt{rectzoom} are both 1.0, the screen region \(xI\) through \(x2\), \(yI\) through \(y2\), are filled. Other zoom factors result in filling past \(x2\) and/or past \(y2\) \((xI, yI\) is always the lower-left corner of the filled region).

\texttt{rectwrite} draws an array of 16-bit words, and therefore should be used only to write color index values. \texttt{lrectwrite} draws an array of 32-bit words. Based on the current \texttt{pixmode}, in can draw pixels of 1, 2, 4, 8, 12, 16, 24, or 32 bits each. Use it to write packed RGB or RGBA values, color index values, or \(z\) values.

\texttt{pixmode} greatly affects the operation of \texttt{lrectwrite}, and has no effect on the operation of \texttt{rectwrite}. By default, \texttt{lrectwrite} draws 32-bit pixels in the format used by \texttt{cpack}. Different pixel sizes, framebuffer shifts, scan patterns through the framebuffer, and strides through memory, can all be specified using \texttt{pixmode}.

\texttt{rectwrite} and \texttt{lrectwrite} leave the current character position unpredictable.

\textbf{SEE ALSO}

\texttt{blendfunction}, \texttt{lrectread}, \texttt{pixmode}, \texttt{rectcopy}, \texttt{rectzoom}

\textbf{NOTES}

These routines are available only in immediate mode.
NAME

IGRBRange – sets the range of RGB colors used for depth-cueing

C SPECIFICATION

void IGRBRange(rmin, gmin, bmin, rmax, gmax, bmax, znear, zfar);
short rmin, gmin, bmin, rmax, gmax, bmax;
long znear, zfar;

PARAMETERS

rmin  expects the minimum value to be stored in the red bitplanes.
gmin  expects the minimum value to be stored in the green bitplanes.
bmin  expects the minimum value to be stored in the blue bitplanes.
rmax  expects the maximum value to be stored in the red bitplanes.
gmax  expects the maximum value to be stored in the green bitplanes.
bmax  expects the maximum value to be stored in the blue bitplanes.
znear expects the nearer screen z, to which the maximum colors are mapped.
zfar  expects the farther screen z, to which the minimum colors are mapped.

DESCRIPTION

IGRBRange sets the range of RGB colors used for depth-cueing in RGB mode. The screen z range [znear, zfar] is mapped linearly into the RGB color range [(rmax,gmax,bmax), (rmin,gmin,bmin)]. Screen z values nearer than znear are mapped to (rmax,gmax,bmax); screen z values farther than zfar are mapped to (rmin,gmin,bmin).

The valid range for znear and zfar depends on the state of the GLC_ZRANGEMAP compatibility mode (see glcompat). If it is 0, the valid range depends on the graphics hardware, where the minimum is the value returned by getgdesc(GD_ZMIN) and the maximum is the value returned by getgdesc(GD_ZMAX). If it is 1, the minimum is 0x0 and the maximum is 0x7FFFFFF. Znear and zfar should be chosen to be consistent with the near and far parameters passed to lsetdepth. If
near < far, then znear should be less than zfar. If near > far, then znear should be greater than zfar. In either case, the range \([near, far]\) should bound the range \([znear, zfar]\).
NAME

lsbackup – controls whether the ends of a line segment are colored

C SPECIFICATION

void lsbackup(b)
Boolean b;

PARAMETERS

b expects either TRUE or FALSE.
TRUE forces the last pixel of a line segment to be colored.
FALSE allows the linestyle to depend whether the last pixel of a line segment to be colored.

DESCRIPTION

lsbackup enables or disables linestyle backup mode. This mode controls how the final pixel of a line segment are rendered. If it is enabled, it causes the current linestyle to be overridden and forces the final pixel of a line segment to be colored. If it is disabled (the default), this does not happen, and line segments can have invisible endpoints.

SEE ALSO

deflinestyle, getlsbackup, resetls

NOTE

This routine only functions on IRIS-4D B and G models, and therefore we advise against its use in new development.
NAME

Isetdepth – sets the depth range

C SPECIFICATION

void Isetdepth(near, far)
long near, far;

PARAMETERS

near expects the screen coordinate of the near clipping plane.

far expects the screen coordinate of the far clipping plane.

DESCRIPTION

viewport specifies the mapping of the left, right, bottom, and top clipping planes into screen coordinates. Isetdepth completes this mapping for homogeneous world coordinates; it specifies the mapping of the near and far clipping planes into values stored in the z-buffer.

Isetdepth is used in z-buffering, depth-cueing, and certain feedback applications.

The valid range of the parameters depends on the state of the GLC_ZRANGEMAP compatibility mode (see glcompat). If it is 0, the valid range depends on the graphics hardware, where the minimum is the value returned by getgdesc(GD_ZMIN) and the maximum is the value returned by getgdesc(GD_ZMAX). If it is 1, the minimum is 0x0 and the maximum is 0x7FFFFF. The depth range defaults to the full range supported by the graphics hardware.

Acceptable mappings include all those where both near and far are within the supported range, including mappings where near > far. In particular, it is sometimes desirable to call Isetdepth(0x7FFFFF, 0x0) on IRIS-4D GT and GTX models.

SEE ALSO

depthcue, feedback, getgdesc, glcompat, zbuffer
NOTE

Error accumulation in the iteration of z can cause wrapping when the full depth range supported by the graphics hardware is used. (An iteration wraps when it accidentally converts an large positive value into a negative value, or vice versa.) While the effects of wrapping are typically not observed, if they are, they can be eliminated by reducing the depth range by a small percentage.
NAME

Ishaderange – sets range of color indices used for depth-cueing

C SPECIFICATION

void Ishaderange(lowin, highin, znear, zfar)
Colorindex lowin, highin;
long znear, zfar;

PARAMETERS

lowin   expects the low-intensity color map index.
highin  expects the high-intensity color map index.
znear   expects the nearer screen z, to which highin is mapped.
zfar    expects the farther screen z, to which lowin is mapped.

DESCRIPTION

Ishaderange sets the range of color indices used for depth-cueing. The screen z range [znear, zfar] is mapped linearly into the color index range [highin, lowin]. Screen z values nearer than znear map to highin; screen z values farther than zfar map to lowin.

The valid range for znear and zfar depends on the state of the GLC_ZRANGEMAP compatibility mode (see glcompat). If it is 0, the valid range depends on the graphics hardware, where the minimum is the value returned by getgdesc(GD_ZMIN) and the maximum is the value returned by getgdesc(GD_ZMAX). If it is 1, the minimum is 0x0 and the maximum is 0x7FFFFFF. The default is Ishaderange(0, 7, Zmin, Zmax), where Zmin and Zmax are the values such that the full range supported by the graphics hardware is used.

Znear and zfar should be chosen to be consistent with the near and far parameters passed to lsetdepth. If near < far, then znear should be less than zfar. If near > far, then znear should be greater than zfar. In either case, the range [near, far] should bound the range [znear, zfar].
SEE ALSO

depthcue, getgdesc, glcompat, lsetdepth
NAME

lsrepeat – sets a repeat factor for the current linestyle

C SPECIFICATION

void lsrepeat(factor)
long factor;

PARAMETERS

factor expects the repeat factor of the linestyle pattern. The valid range of factor is 1 through 255.

DESCRIPTION

lsrepeat is used to create linestyles that are longer than 16 bits by multiplying each bit in the pattern by factor. When a line is drawn, pixels are written if there is a 1 in the corresponding position of the linestyle mask and not written if there is a 0 in the corresponding position. When lsrepeat is used each bit in the pattern is multiplied successively by factor. If the line pattern is 000000111111111 and factor = 3, the resulting linestyle would be 27 bits on followed by 21 bits off. Line patterns start from the least significant bit.

SEE ALSO

deflinestyle, getlsrepeat
NAME
makeobj – creates an object

C SPECIFICATION
void makeobj(obj)
Object obj;

PARAMETERS
obj expects the numeric identifier for the object being defined.

DESCRIPTION
makeobj creates and names a new object by entering the identifier, specified by obj, into a symbol table and allocating memory for its list of drawing routines. If obj is the number of an existing object, the contents of that object are deleted. Drawing routines are then added into the display list instead of executing, until closeobj is called.

SEE ALSO
callobj, closeobj, genobj, isobj, chunksize

NOTE
This routine is available only in immediate mode.
NAME

maketag – numbers a routine in the display list

C SPECIFICATION

void maketag(t)
    Tag t;

PARAMETERS

t expects a numeric identifier, or tag, which the system places between two list items. A tag locates display list items for editing.

DESCRIPTION

maketag places markers that identify specific locations of drawing routines within an object definition. 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
NAME

mapcolor — changes a color map entry

C SPECIFICATION

void mapcolor(i, red, green, blue)
Colorindex i;
short red, green, blue;

PARAMETERS

i        expects the index into the color map.
red      expects an intensity value in the range 0 to 255 for red to be associated with the index.
green    expects an intensity value in the range 0 to 255 for green to be associated with the index.
blue     expects an intensity value in the range 0 to 255 for blue to be associated with the index.

DESCRIPTION

mapcolor loads entry i of the color map for the current drawing mode with (red, green, blue). Pixels written with color index i are displayed with the specified RGB intensities. The valid range for i depends on the number of bitplanes available in the current drawing and buffer modes, i.e. the value returned by getplanes. Using Ni to represent 2 raised to the return value of getplanes in drawing mode i, the valid ranges are:

NORMALDRAW 0 to Nn–1.
OVERDRAW 1 to No–1.
UNDERDRAW 0 to Nu–1.
PUPDRAW 1 to Np–1.
CURSORDRAW 1 to getgdesc(GD_BITS_CURSOR).

If Ni is 1, then no indices are valid. Invalid indices are ignored by mapcolor.
In multimap mode, `mapcolor` updates only the small color map currently selected by `setmap`.

The color map entry that controls the color of the cross-hair cursor (cursor type CCROSS) is returned by the `getgdesc` inquiry GD_CROSSHAIR_CINDEX.

SEE ALSO

color, curstype, drawmode, gammaramp, getgdesc, getmcolor, getplanes, setmap

NOTES

This subroutine is available only in immediate mode.

On the IRIS-4D G, you should not alter the top 256 colors (color indices 3840 to 4095). The system uses these colors for the cursor, overlay bitplanes, and RGB mode. If you alter the colors to which these features are mapped, some screen features will appear in strange colors.
NAME

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

C SPECIFICATION

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

PARAMETERS

vobj expects a viewing object containing the transformations that map the current displayed objects to the screen.
sx expects the x coordinate of the screen point to be mapped.
sy expects the y coordinate of the screen point to be mapped.
wx1 returns the x world coordinate of one endpoint of a line.
wz1 returns the z world coordinate of one endpoint of a line.
wy1 returns the y world coordinate of one endpoint of a line.
wz2 returns the z world coordinate of the remaining endpoint of a line.
wx2 returns the x world coordinate of the remaining endpoint of a line.
wz2 returns the z world coordinate of the remaining endpoint of a line.

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 the viewing object, vobj.

A viewing object is a graphical object that contains only viewport, projection, viewing transformation, and modeling routines. A correct mapping from screen coordinate to world coordinates requires that 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

**NOTE**

This routine is available only in immediate mode.
NAME

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

C SPECIFICATION

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

PARAMETERS

vobj expects the transformations that map the displayed objects to world coordinates.
sx expects the x coordinate of the screen point to be mapped.
sy expects the y coordinate of the screen point to be mapped.
wx returns the corresponding x world coordinate.
wy returns the corresponding y world coordinate.

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 2-D, the result is undefined.

SEE ALSO

mapw

NOTE

This routine is available only in immediate mode.
NAME

maxsize – specifies the maximum size of a graphics window

C SPECIFICATION

void maxsize(x, y)
long x, y;

PARAMETERS

x expects the maximum width of a graphics window. The width is measured in pixels.
y expects the maximum height of a graphics window. The height is measured in pixels.

DESCRIPTION

maxsize specifies the maximum size (in pixels) of a graphics window. 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 getgdesc(GD_XPMAX) pixels wide and getgdesc(GD_YPMAX) pixels high. The user can reshape the graphics window, but the window manager does not allow it to become larger than the specified maximum size.

SEE ALSO

getgdesc, minsize, winopen

NOTE

This routine is available only in immediate mode.
NAME

minsize – specifies the minimum size of a graphics window

C SPECIFICATION

void minsize(x, y)
long x, y;

PARAMETERS

x expects the minimum width of a graphics window. The width is measured in pixels. The lowest legal value for this parameter is 1.

y expects the minimum height of a graphics window. The height is measured in pixels. The lowest legal value for this parameter is 1.

DESCRIPTION

minsize specifies the minimum size (in pixels) of a graphics window. Call it at the beginning of a graphics program. It takes effect when winopen is called. You can also call minsize 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.

SEE ALSO

maxsize, winopen

NOTE

This routine is available only in immediate mode.
NAME

mmode — sets the current matrix mode

C SPECIFICATION

void mmode(m)
short m;

PARAMETERS

m expects a symbolic constant, one of:

MSINGLE puts the system into single-matrix mode. In single-
matrix mode, all modeling, viewing, and projection transformations
are done using a single matrix that combines all these transforma-
tions. This is the default matrix mode.

MVIEWING puts the system into multi-matrix mode. In this mode,
separate ModelView, Projection, and Texture matrices are main-
tained. The ModelView matrix is modified by all matrix opera-
tions.

MPROJECTION puts the system into multi-matrix mode. In this mode,
separate ModelView, Projection, and Texture matrices are main-
tained. The Projection matrix is modified by all matrix opera-
tions.

MTEXTURE puts the system into multi-matrix mode. In this mode,
separate ModelView, Projection, and Texture matrices are main-
tained. The Texture matrix is modified by all matrix operations.

DESCRIPTION

mmode specifies which matrix is the current matrix, and also deter-
mines whether the system is in single-matrix mode, or in multi-matrix
mode. The matrix mode and current matrix are determined as follows:
<table>
<thead>
<tr>
<th>mmode</th>
<th>matrix mode</th>
<th>current matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSINGLE</td>
<td>single</td>
<td>only matrix</td>
</tr>
<tr>
<td>MVIEWING</td>
<td>multi</td>
<td>ModelView</td>
</tr>
<tr>
<td>MPROJECTION</td>
<td>multi</td>
<td>Projection</td>
</tr>
<tr>
<td>MTEXTURE</td>
<td>multi</td>
<td>Texture</td>
</tr>
</tbody>
</table>

In single-matrix mode, vertices are transformed directly from object-coordinates to clip-coordinates by a single matrix. All matrix commands operate on this, the only matrix. Single-matrix mode is the default mode, but its use is discouraged, because many of the newer GL rendering features cannot be used while the system is in single-matrix mode.

In multi-matrix mode, vertices are transformed from object-coordinates to eye-coordinates by the ModelView matrix, then from eye-coordinates to clip-coordinates by the Projection matrix. A third matrix, the Texture matrix, is maintained to transform texture coordinates. While in multi-matrix mode, mmodes MVIEWING, MPROJECTION, and MTEXTURE specify which of the three matrices is operated on by matrix modification commands. Many GL rendering operations, including lighting, texture mapping, and user-defined clipping planes, require that the matrix mode be multi-matrix.

Both the single matrix that is maintained while mmode is MSINGLE mode, and the ModelView matrix that is maintained while not in MSINGLE mode, have a stack depth of 32. The Projection and Texture matrices are not stacked. Thus matrix commands `pushmatrix` and `popmatrix` should not be called while the matrix mode is MPROJECTION or MTEXTURE.

Changes between matrix modes MVIEWING, MPROJECTION and MTEXTURE have no effect on the matrix values themselves. However, when matrix mode MSINGLE is entered or left, all matrix stacks are forced to be empty, and all matrices are initialized to the identity matrix.

SEE ALSO
clipplane, getmmode, Imbind, lookat, ortho, perspective, polarview, rot, rotate, scale, texbind, translate, window
BUGS

On IRIS-4D G, GT, GTX systems, and on the Personal IRIS, multi-matrix operation is incorrect while \texttt{mmode} is \texttt{MPROJECTION}. Specifically, vertices are transformed only by the Projection matrix, not by the ModelView matrix.
NAME

move, movei, moves, move2, move2i, move2s – moves the current graphics position to a specified point

C SPECIFICATION

void move(x, y, z)
Coord x, y, z;
void movei(x, y, z)
Icoord x, y, z;
void moves(x, y, z)
Scoord x, y, z;
void move2(x, y)
Coord x, y;
void move2i(x, y)
Icoord x, y;
void move2s(x, y)
Scoord x, y;

All of the above routines are functionally the same. They differ only in the type declarations of their parameters and in whether they assume a three- or two-dimensional space.

PARAMETERS

$x$ expects the new $x$ coordinate for the current graphics position.

$y$ expects the new $y$ coordinate for the current graphics position.

$z$ expects the new $z$ coordinate for the current graphics position (when applicable).

DESCRIPTION

move changes (without drawing) the current graphics position to the point specified by $x$, $y$, and $z$. The graphics position is the point from which the next drawing routine will start drawing.
move2(x, y) is equivalent to move(x, y, 0.0).

SEE ALSO
bgnline, draw, endline, v

NOTE
move should not be used in new development. Rather, lines should be drawn using the high-performance v commands, surrounded by calls to bgnline and endline.
NAME

`mswapbuffers` — swap multiple framebuffers simultaneously

C SPECIFICATION

```c
void mswapbuffers(fbuf)
long fbuf;
```

PARAMETERS

`fbuf`  
Expects a bitfield comprised of the logical OR of one or more of the following symbols:

- `NORMALDRAW` indicates that the normal framebuffer is to be swapped.
- `OVERDRAW` indicates that the overlay framebuffer is to be swapped.
- `UNDERDRAW` indicates that the underlay framebuffer is to be swapped.

DESCRIPTION

`mswapbuffers` exchanges the front and back buffers of multiple framebuffers simultaneously. Which framebuffers are to have their buffers exchanged is specified by the bitfield `fbuf`, the only argument. The normal, overlay, and underlay framebuffers are specified with bitmasks `NORMALDRAW`, `OVERDRAW`, and `UNDERDRAW`. These masks must be ORed together to generate the `fbuf` argument. For example, both the normal and overlay framebuffers are swapped by the command: `mswapbuffers(NORMALDRAW | OVERDRAW)`.

`mswapbuffers` is executed during a vertical retrace period that closely follows the time of the request (usually the next vertical retrace).

`mswapbuffers` is ignored by framebuffers that are not in doublebuffer mode.
SEE ALSO

doublebuffer, drawmode, swapbuffers, swapinterval

NOTES

IRIS-4D models G, GT, and GTX, and the Personal Iris, do not implement *mswapbuffers*.
NAME

multimap – organizes the color map as a number of smaller maps

C SPECIFICATION

void multimap()

PARAMETERS

none

DESCRIPTION

multimap organizes the color map of the currently active framebuffer as a number of smaller maps. Because only the normal framebuffer supports multiple color maps, multimap should be called only while drawmode is NORMALDRAW.

There are getgdesc(GD_NMMAPS) maps, each of which will have up to 256 entries, depending on the number of bitplanes available. Call getplanes after setting the drawing mode to the desired framebuffer to determine the color map size. getgdesc can also be called at any time to determine the size of the color map of any framebuffer.

multimap does not take effect until gconfig is called. When called, gconfig executes multimap requests pending for all drawing modes, regardless of the current drawing mode.

A framebuffer’s color map is used to display pixels only if the framebuffer is in color map mode.

SEE ALSO

cmode, drawmode, gconfig, getcmmode, getgdesc, getmap, onemap, setmap

NOTE

This routine is available only in immediate mode.
NAME

multmatrix – premultiplies the current transformation matrix

C SPECIFICATION

void multmatrix(m)
Matrix m;

PARAMETERS

m expects the matrix that is to multiply the current top matrix of the transformation stack.

DESCRIPTION

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

SEE ALSO

getmatrix, loadmatrix, popmatrix, pushmatrix
NAME

n3f – specifies a normal

C SPECIFICATION

void n3f(vector)
float vector[3]

PARAMETERS

vector expects the address of an array containing three floating point numbers. These numbers are used to set the value for the current vertex normal.

DESCRIPTION

n3f specifies a floating point normal for lighting calculations. The normal becomes the current normal for subsequent vertices; it is not necessary to respecify a normal if it is unchanged (e.g., a single call to n3f specifies normals for all vertices of a flat-shaded polygon).

Vector components are Nx, Ny, and Nz for indices 0, 1, and 2.

Lighting calculations assume that the specified normal is of unit length. If non-unit length normals are to be specified, use nmode to inform the system that normals must be normalized. Lighting performance may be reduced in this event.

When called with unequal arguments, scale causes the ModelView matrix to become nonorthonormal. In this case, or in any other case that results in a nonorthonormal ModelView matrix, normals are also renormalized automatically. Performance reduction, if any, matches that of nmode user-specified normalization.

SEE ALSO

Imbind, Imdef, nmode
NAME

newpup — allocates and initializes a structure for a new menu

C SPECIFICATION

long newpup()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned value of this function is a menu identifier.

DESCRIPTION

newpup allocates and initializes a structure for a new menu; it returns a positive menu identifier. Use newpup with addtopup to create pop-up menus.

SEE ALSO

addtopup, defpup, dopup, freepup

NOTE

This routine is available only in immediate mode.
NAME

newtag – creates a new tag within an object relative to an existing tag

C SPECIFICATION

void newtag(newtg, oldtg, offst)
Tag newtg, oldtg;
Offset offst;

PARAMETERS

newtg expects an identifier for the tag that will be created.
oldtg expects an existing tag. It will be used as a reference point for
inserting newtg.
offst expects the number of positions beyond oldtg where newtg. will
be placed.

DESCRIPTION

newtag creates a new tag and places it at the specified number of posi-
tions beyond oldtg. The number of positions is indicated by offst.
newtag is used within an object after at least one tag has been created
by calling maketag.

SEE ALSO

maketag

NOTE

This routine is available only in immediate mode.
NAME

nmode — specify renormalization of normals

C SPECIFICATION

void nmode(mode)
long mode;

PARAMETERS

mode expects a symbolic constant. There are two defined constants for this parameter:

NAUTO causes normals to be renormalized only if the current ModelView matrix is not orthonormal. (default)

NNORMALIZE causes normals to always be renormalized, regardless of the current ModelView matrix.

DESCRIPTION

IRIS systems transform vertex normals from object-coordinates to eye-coordinates before doing lighting calculations. While the matrix mode is MVIEWING, a separate Normal matrix is maintained to support this transformation. The Normal matrix is the inverse transpose of the upper-left 3 × 3 portion of the ModelView matrix.

Transformed normals must be unit length if the lighting calculations are to be meaningful. Transformed normals will be unit length if 1) they were unit length in object-coordinates, and 2) the current Normal matrix is orthonormal (see notes). If one or both of these conditions are not met, the normal must be normalized (corrected to have unit length) after it is transformed. nmode helps the system determine when normalization is required.

Each time the ModelView matrix is changed, the IRIS determines whether the resulting (inverse-transpose) Normal matrix is orthonormal or not, and saves the result of the test as a flag. After each normal is transformed, both this flag and the nmode flag are tested. If nmode is NAUTO, the normal is normalized if and only if the flag is set (i.e. the ModelView matrix is not orthonormal). NAUTO mode is appropriate when the model normals are known to be unit length. If nmode is
NNORMALIZE, the normal is normalized unconditionally. NNORMALIZE mode is appropriate when the model normals may not be unit length.

NAUTO is the default nmode.

Because normalization involves division by a computed square root, it can adversely affect system performance.

SEE ALSO

nmode, loadmatrix, multmatrix, rot, scale, translate, lmbind

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support nmode.

nmode cannot be used while draw mode is MSINGLE.

For our purposes a matrix is orthogonal if it transforms normals to the same length regardless of their direction, and it is orthonormal if this length is the same as the untransformed length. Rotation matrixes are always orthonormal. Scale matrixes are orthogonal but not orthonormal if the three scale values are identical, neither orthogonal nor orthonormal otherwise. Uniform scale ModelView matrixes can be normalized to the identity matrix, and are therefore ignored by the Normal matrix. Translations do not affect the upper-left 3x3 ModelView matrix, and are therefore also ignored by the Normal matrix.

The length of a normal is the square root of its dot product with itself.
NAME

noborder – specifies a window without any borders

C SPECIFICATION

void noborder()

PARAMETERS

none

DESCRIPTION

noborder specifies a window that has no borders around its drawable area. Call noborder before you open the window.

SEE ALSO

winconstraints
NAME

noise – filters valuator motion

C SPECIFICATION

void noise(v, delta)
Device v;
short delta;

PARAMETERS

\( v \)  expects a valuator. A valuator is a single-value input device.

\( delta \)  expects the number of units of change required before the valuator \( v \) can make a new queue entry.

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 is used to 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.

The default noise value for all valuators is 1, except for the timer devices (TIMERn), for which it is 10000. The frequency of timer events is returned by the getgdesc inquiry GD_TIMERHZ.

SEE ALSO

getgdesc, qdevice, setvaluator

NOTE

This routine is available only in immediate mode.
NAME

noport — specifies that a program does not need screen space

C SPECIFICATION

noport()

PARAMETERS

none

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 winopen to do a graphics initialization.

The system ignores noport if winopen is not called.

SEE ALSO

winopen

NOTE

This routine is available only in immediate mode.
NAME

normal – obsolete routine

C SPECIFICATION

void normal(narray)
Coord narray[3];

PARAMETERS

narray expects the address of an array containing three floating point numbers. These numbers are used to set the value for the current vertex normal.

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use its identical replacement, n3f.

SEE ALSO

n
NAME
	nurbscurve - controls the shape of a NURBS trimming curve

C SPECIFICATION

    void nurbscurve(knot_count, knot_list, offset, cttarray, order, type)   
    long knot_count, offset;                                               
    double knot_list[], *cttarray;                                       
    long order, type;

PARAMETERS

knot_count   expects the number of knots.
knot_list    expects an array of knot_count non-decreasing knot values.
offset       expects the offset (in bytes) between successive curve control points
cttarray    expects an array containing control points for the NURBS curve. The coordinates must appear as either (x, y) pairs or as (wx, wy, w) triples. The offset between successive control points is given by offset.
order        expects the order of the NURBS curve. The order is one more than the degree, hence, a cubic curve has an order of 4.
type         expects a value indicating the control point type. Current options are N_P2D and N_P2DR, denoting double-precision parametric coordinates in the two-dimensional parameter space of a trimmed surface. N_P2D denotes non-rational (2) coordinates, while N_P2DR denotes rational (3) coordinates.

DESCRIPTION

Use **nurbscurve** to describe a NURBS curve. Use NURBS curves within trimming loop definitions. A trimming loop definition is a set of oriented curve commands that describe a closed loop. To mark the beginning of a trimming loop definition, use the **bgntrim** command. To mark the end of a trimming loop definition, use an **endtrim** command.
nurbscurve  Graphics Reference, C Edition  nurbscurve

You use trimming loop definitions within NURBS surface definitions (see bgnsurface). The trimming loops are closed curves that the system uses to set the boundaries of a NURBS surface. You can describe a trimming loop by using a series of NURBS curves, piecewise linear curves (see pwlcurve), or both.

When the system needs to decide which part of a NURBS surface you want it to display, it displays the region of the NURBS surface that is to the left of the trimming curves as the parameter increases. Thus, for a counter-clockwise oriented trimming curve, the displayed region of the NURBS surface is the region inside the curve. For a clockwise oriented trimming curve, the displayed region of the NURBS surface is the region outside the curve.

The offset parameter is used in case the control points are part of an array of larger structure elements. The nurbscurve routine searches for the n-th control point pair or triple beginning at byte address ctlarray + n \times offset.

See the Graphics Library Programming Guide for a mathematical description of a NURBS curve.

SEE ALSO

bgnurface, nurbssurface, bgntrim, pwlcurve, getnurbsproperty, setnurbsproperty
NAME

nurbs_surface – controls the shape of a NURBS surface

C SPECIFICATION

void nurbs_surface(s_knot_count, s_knot, t_knot_count, t_knot,
    s_offset, t_offset, ctablarray,
    s_order, t_order, type)
long s_knot_count, t_knot_count;
double s_knot[], t_knot[];
long s_offset, t_offset;
double *cttablarray;
long s_order, t_order, type;

PARAMETERS

s_knot_count expects the number of knots in the parametric s direction.
s_knot expects an array of s_knot_count non-decreasing knot values in the parametric s direction.
t_knot_count expects the number of knots in the parametric t direction.
t_knot expects an array of t_knot_count non-decreasing knot values in the parametric t direction.
s_offset expects the offset (in bytes) between successive control points in the parametric s direction in ctablarray.
t_offset expects the offset (in bytes) between successive control points in the parametric t direction in ctablarray.
ctablarray expects an array containing control points for the NURBS surface. The coordinates must appear as either (x, y, z) triples or as (wx, wy, wz, w) quadruples. The offsets between successive control points in the parametric s and t directions are given by s_offset and t_offset.
expects the order of the NURBS surface in the parametric s direction. The order is one more than the degree, hence, a cubic surface has an order of 4.

t_order

expects the order of the NURBS surface in the parametric t direction. The order is one more than the degree, hence, a cubic surface has an order of 4.

type

expects a value indicating the control point type. Current options are N_V3D, N_V3DR, N_C4D, N_C4DR, and N_T2D, N_T2DR. Types N_V3D and N_V3DR denote double-precision positional coordinates in a three-dimensional model space. N_V3D denotes non-rational (3) coordinates and N_V3DR denotes rational (4) coordinates. Types N_C4D and N_C4DR denote double-precision color coordinates in a four-dimensional RGBA color space. N_C4D denotes non-rational coordinates and N_C4DR denotes rational coordinates. Types N_T2D and N_T2DR denote double-precision texture coordinates in a two-dimensional texture space. N_T2D denotes non-rational coordinates and N_T2DR denotes rational coordinates.

DESCRIPTION

Use the nurbssurface command within a NURBS (Non-Uniform Rational B-Spline) surface definition to describe the shape of a NURBS surface before any trimming takes place. To mark the beginning of a NURBS surface definition, use the bgnsurface command. To mark the end of a NURBS surface definition, use the endsurface command. Call nurbssurface within a NURBS surface definition only.

Positional, texture, and color coordinates are associated by presenting each as a separate nurbssurface between a bgnsurface/endsurface pair. No more than one call to nurbssurface for each of color and texture data may be made within a single bgnsurface/endsurface pair. Exactly one call must be made to describe position data and it must be the last call to nurbssurface between the bracketing bgnsurface/endsurface.
EXAMPLE

bgnsurface();
  nurbssurface( ..., N_C4D );  /* color data */
  nurbssurface( ..., N_T2D );  /* texture data */
  nurbssurface( ..., N_V3D );  /* position data */
endsurface();

You can trim a NURBS surface by using the commands nurbscurve
and pwlcurve between calls to bgntrim and endtrim.

Observe that a nurbssurface with s_knot_count knots in the s
direction and t_knot_count knots in the t direction with orders s_order
and t_order must have (s_knot_count − s_order) × (t_knot_count − t_order)
control points.

The system renders a NURBS surface as a polygonal mesh and analyti-
cally calculates normal vectors at the corners of the polygons within the
mesh. Therefore, your code should specify a lighting model when it
uses NURBS surfaces. Otherwise, you loose all the interesting surface
information. Use Imdef and Imbind to define or modify materials and
their properties.

See the Graphics Library Programming Guide for a mathematical
description of a NURBS surface.

SEE ALSO

bgnsurface, nurbscurve, bgntrim, pwlcurve, getnurbsproperty,
setnurbsproperty, texbind

NOTE

nurbssurface commands specifying color or texture coordinates
currently have no effect on IRIS-4D G, GT, GTX, and Personal IRIS.
NAME

`objdelete` — deletes routines from an object

C SPECIFICATION

```c
void objdelete(tag1, tag2)
    Tag tag1, tag2;
```

PARAMETERS

- `tag1` expects the tag indicating where the deletion is to be started from.
- `tag2` expects the tag indicating where the deletion should stop.

DESCRIPTION

`objdelete` is an object editing routine. It deletes the routines as well as any tags starting immediately after `tag1` and ending just prior to `tag2`. `tag1` and `tag2` remain in the text.

If no object is open for editing (see `editobj`) 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`

NOTE

This routine is available only in immediate mode.
NAME

`objinsert` – inserts routines in an object at a specified location

C SPECIFICATION

```c
void objinsert(t)
    Tag t;
```

PARAMETERS

`t` expects a tag within the object definition that is to be edited.

DESCRIPTION

`objinsert` positions an editing pointer on the routine specified by `t`. The additional graphics routines should now be inserted after the tag.

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

SEE ALSO

`closeobj`, `editobj`, `maketag`, `objdelete`, `objreplace`

NOTE

This routine is available only in immediate mode.
NAME

**objreplace** – overwrites existing display list routines with new ones

C SPECIFICATION

```c
void objreplace(t)
Tag t;
```

PARAMETERS

`t` expects a tag within the object definition that is to be edited.

DESCRIPTION

**objreplace** combines the functions of **objinsert** and **objdelete**. Graphics routines that follow **objreplace** overwrite existing ones until **closeobj**, **objinsert**, **objdelete**, or **objreplace** terminates the replacement. This replacement begins with the line immediately following the tag specified by `t`.

**objreplace** requires that 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

NOTE

This routine is available only in immediate mode.
NAME

onemap — organizes the color map as one large map

C SPECIFICATION

void onemap()

PARAMETERS

none

DESCRIPTION

onemap organizes the color map of the currently active framebuffer as a single map. Because single map mode is the default value for all GL framebuffers, it can be called in any of the framebuffer drawmodes (NORMALDRAW, PUPDRAW, OVERDRAW, and UNDERDRAW). To return the normal framebuffer to single map mode, however, you must call onemap while in drawmode NORMALDRAW.

The single color map allocated to a framebuffer in onemap mode has as many entries as are supported by the bitplanes in that framebuffer. The normal framebuffer has up to 12 bitplanes, and therefore up to 4096 color map entries. Call getplanes after setting draw mode to the desired framebuffer to determine the color map size. getgdesc can also be called at any time to determine the size of the color map of any framebuffer.

onemap does not take effect until gconfig is called. When called, gconfig executes onemap requests pending for all draw modes, regardless of the current draw mode.

A framebuffer’s color map is used to display pixels only if the framebuffer is in color map mode (see cmode).

SEE ALSO

cmode, drawmode, gconfig, getcmmode, getmap, multimap, setmap
NOTE

This routine is available only in immediate mode.
NAME

ortho, ortho2 – define an orthographic projection transformation

C SPECIFICATION

void ortho(left, right, bottom, top, near, far)
Coord left, right, bottom, top, near, far;

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

The above routines are functionally the same. They differ only in that ortho is used for 3-D applications and ortho2 is used for 2-D applications.

PARAMETERS

left expects the coordinate for the left vertical clipping plane.
right expects the coordinate for the right vertical clipping plane.
bottom expects the coordinate for the bottom horizontal clipping plane.
top expects the coordinate for the top horizontal clipping plane.
near expects the distance to the nearer depth clipping plane.
far expects the distance to the farther depth clipping plane.

DESCRIPTION

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

ortho2 is the 2-D version of ortho, and specifies a rectangle that is the mapped to the viewport. When you use ortho2 with 3-D world coordinates, the z coordinates are not transformed and will be clipped if they lie outside the range –1 ≤ z ≤ 1.
When the system is in single matrix mode, both `ortho` and `ortho2` load a matrix onto the matrix stack, thus replacing the current top matrix. When the system is in viewing, projection, or texture matrix mode, the system replace the current Projection matrix without changing the ModelView matrix stack or the Texture matrix.

GL window coordinates have integer values at the centers of pixels. Thus to correctly specify a one-to-one orthographic mapping from eye-coordinates to window-coordinates, the edges of the viewable volume should be set to 1/2-pixel values. For example, the $1280 \times 1024$ full screen is correctly mapped one-to-one from eye-coordinates to window-coordinates by the commands:

```
ortho2(-0.5,1279.5,-0.5,1023.5);
viewport(0,1279,0,1023);
```

Note that `ortho`, unlike `perspective` and `window`, allows the viewpoint to be moved from the origin of the coordinate system. Thus `ortho` combines a trivial viewing transformation (translation from the origin) with its projection operation. Be sure not to duplicate the orthographic translation in your viewing transformation.

**SEE ALSO**

`mmode`, `perspective`, `viewport`, `window`
NAME

overlay – allocates bitplanes for display of overlay colors

C SPECIFICATION

    void overlay(planes)
    long planes;

PARAMETERS

    planes expects the number of bitplanes to be allocated for overlay colors. Valid values are 0, 2 (the default), 4, and 8.

DESCRIPTION

The IRIS physical framebuffer is divided into four separate GL framebuffers: normal, popup, overlay, underlay. Because a single physical framebuffer is used to implement the four GL framebuffers, bitplanes must be allocated among the GL framebuffers. overlay specifies the number of bitplanes to be allocated to the overlay framebuffer. overlay does not take effect immediately. Rather, it is considered only when gconfig is called, at which time all requests for bitplane resources are resolved.

While only one of the four GL framebuffers can be drawn to at a time (see drawmode), all four are displayed simultaneously. The decision of which to display at each pixel is made based on the contents of the four framebuffer at that pixel location, using the following hierarchical rule:

if the popup pixel contents are non-zero
then display the popup bitplanes
else if overlay bitplanes are allocated AND the overlay pixel contents are non-zero
then display the overlay bitplanes
else if the normal pixel contents are non-zero OR no underlay bitplanes are allocated
then   display the normal bitplanes
else   display the underlay bitplanes

Thus images drawn into the overlay framebuffer appear over images in
the normal framebuffer, and images drawn into the underlay framebuffer
appear under images in the normal framebuffer. Popup images appear
over everything else.

The default configuration of the overlay framebuffer is 2 bitplanes, sin-
gle buffer, color map mode. To make a change to this configuration
other than to change the bitplane size, the drawing mode must be
OVERDRAW. For example, the overlay framebuffer can be configured
to be double buffered by calling doublebuffer while draw mode is
OVERDRAW.

On models that cannot support overlay and underlay bitplanes simulta-
neously, calling overlay with a non-zero argument forces underlay to
zero. When simultaneous overlay and underlay operation is supported,
calling overlay may have no effect on the number of underlay bitplanes.

SEE ALSO
doublebuffer, drawmode, gconfig, getgdesc, singlebuffer, underlay

NOTES
This routine is available only in immediate mode.
IRIS-4D G, GT, and GTX models, and the Personal Iris, support only
single buffered, color map mode overlay bitplanes.

The Personal Iris supports 0 or 2 overlay bitplanes. There are no over-
lay or underlay bitplanes in the minimum configuration of the Personal
Iris.

IRIS-4D GT and GTX models support 0, 2, or 4 overlay bitplanes. Be-
cause 4-bitplane allocation reduces the popup framebuffer to zero bit-
planes, however, its use is strongly discouraged. The window manager
cannot operate properly when no popup bitplanes are available.

IRIS-4D VGX models support 0, 2, 4, or 8 overlay bitplanes, either sin-
gle or double buffered, in color map mode only. The 4 and 8 bitplane
allocations utilize the alpha bitplanes, which must be present, and which
therefore are unavailable in draw mode NORMALDRAW.
BUGS

The Personal Iris does not support shade model GOURAUD in the overlay framebuffer.
NAME

`pagecolor` – sets the color of the textport background

C SPECIFICATION

```c
void pagecolor(pcolor)
    Colorindex pcolor;
```

PARAMETERS

`pcolor` expects an index into the current color map.

DESCRIPTION

`pagecolor` sets the background color of the textport of the calling process. If the calling process was invoked from a `wsh` window, this window is used for its textport; otherwise, the process does not have a textport and this routine does nothing.

SEE ALSO

`textcolor`

`wsh(1)` in the *User's Reference Manual.*

NOTES

This routine is available only in immediate mode.

A process launched from `4Sight` or *The IRIS WorkSpace™* will not have a textport. Therefore, we do not recommend the use of this routine in new development.
NAME

`passthrough` - passes a single token through the Geometry Pipeline

C SPECIFICATION

```c
void passthrough(token)
    short token;
```

PARAMETERS

`token` expects an integer which is used to mark specific sections in input data so that when it is returned from the feedback buffer the data is easier to decipher.

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.

NOTE

This routine is available only in feedback mode; otherwise it is ignored.
NAME

patch – draws a surface patch

C SPECIFICATION

void patch(geomx, geomy, geomz)
Matrix geomx, geomy, geomz;

PARAMETERS

geomx expects the 4x4 matrix which contains the x coordinates of the
16 control points of the patch.

geomy expects the 4x4 matrix which contains the y coordinates of the
16 control points of the patch.

geomz expects the 4x4 matrix which contains the z coordinates of the
16 control points of the patch.

DESCRIPTION

patch draws a surface patch using the current patchbasis, patchpreci-
sion, and patchcurves which are defined earlier. The control points
geomx, geomy, geomz determine the shape of the patch.

The patch is drawn as a web of curve segments. Each curve segment is
approximated by a sequence of straight lines. All lines use the current
linestyle, which is reset prior to the first line of each curve segment, and
continues through subsequent lines in each curve segment. Other line
modes, including depthcuing, line width, and line antialiasing, also
apply to the lines generated by patch.

SEE ALSO

defbasis, patchbasis, patchcurves, patchprecision, rpatch
NAME

`patchbasis` — sets current basis matrices

C SPECIFICATION

```c
void patchbasis(uid, vid)
long uid, vid;
```

PARAMETERS

- `uid` expects the basis that defines how the control points determine the shape of the patch in the "u" direction.
- `vid` expects the basis that defines how the control points determine the shape of the patch in the "v" direction.

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`, `patchprecision`, `patchcurves`, `rpatch`
NAME

patchcurves – sets the number of curves used to represent a patch

C SPECIFICATION

void patchcurves(ucurves, vcurves)
long ucurves, vcurves;

PARAMETERS

ucurves expects the number of curve segments that will be drawn in the "u" direction.

vcurves expects the number of curve segments that will be drawn in the "v" direction.

DESCRIPTION

patchcurves sets the number of u and v curves in the wire frame that represents a patch.

SEE ALSO

patch, patchbasis, patchprecision, rpatch
NAME

patchprecision — sets the precision at which curves are drawn in a patch

C SPECIFICATION

    void patchprecision(usegs, vsegs)
    long usegs, vsegs;

PARAMETERS

    usegs  expects the number of line segments used to draw a curve in the
           "u" direction.
    vsegs  expects the number of line segments used to draw a curve in the
           "v" direction.

DESCRIPTION

    patchprecision sets the precision with which the system draws the
    curves that make up a wireframe patch. Patch precisions are similar to
    curve precisions.

SEE ALSO

    curveprecision, patchbasis, patchcurves, patch, rpatch
NAME
pclos — closes a filled polygon

C SPECIFICATION

void pclos()

PARAMETERS

none

DESCRIPTION

pclos closes a filled polygon that has been created by using pmv and a sequence of pdr calls (or rpmv and rpdr calls). It is not needed when using poly or polf because these procedures close the polygon within their own routines. pclos closes the polygon by connecting the last point with the first. The polygon so defined is filled using the current pattern, color, and writemask. For example, the following sequence draws a filled square:

\[
\begin{align*}
\text{pmv}(0.0, 0.0, 0.0); \\
pdr(1.0, 0.0, 0.0); \\
pdr(1.0, 1.0, 0.0); \\
pdr(0.0, 1.0, 0.0); \\
\text{pclos}(); \\
\end{align*}
\]

SEE ALSO

bgnpolygon, endpolygon, pdr, pmv, v

NOTES

pclos should not be used in new development. Rather, polygons should be drawn using the high-performance v commands, surrounded by calls to bgnpolygon and endpolygon.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 pdr calls between pmv and pclos.
Be careful not to confuse `pclos` with the IRIX system call `pclose`, which closes an IRIX pipe.
NAME

\texttt{pdr}, \texttt{pdri}, \texttt{pdrs}, \texttt{pdr2}, \texttt{pdr2i}, \texttt{pdr2s} – specifies the next point of a polygon

C SPECIFICATION

\texttt{void pdr(x, y, z)}
\texttt{Coord x, y, z;}

\texttt{void pdri(x, y, z)}
\texttt{Icoord x, y, z;}

\texttt{void pdrs(x, y, z)}
\texttt{Scoord x, y, z;}

\texttt{void pdr2(x, y)}
\texttt{Coord x, y;}

\texttt{void pdr2i(x, y)}
\texttt{Icoord x, y;}

\texttt{void pdr2s(x, y)}
\texttt{Scoord x, y;}

All of the above routines are functionally the same. They differ only in the type declarations of their parameters and in whether they expect a two- or three-dimensional space.

PARAMETERS

\texttt{x} expects the \texttt{x} coordinate of the next defining point for the polygon.
\texttt{y} expects the \texttt{y} coordinate of the next defining point for the polygon.
\texttt{z} expects the \texttt{z} coordinate of the next defining point for the polygon.

DESCRIPTION

\texttt{pdr} specifies the next point of a polygon. When \texttt{pdr} is executed, it draws a line to the specified point \((x,y,z)\) which then becomes the current graphics position. The next \texttt{pdr} call will start drawing from that point. To draw a typical polygon start with \texttt{pmv}, follow it with a sequence of calls to \texttt{pdr} and end it with \texttt{pclos}. 
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();
```

SEE ALSO

bgnpolygon, endpolygon, pclos, pmv, v

NOTES

pdr should not be used in new development. Rather, polygons should be drawn using the high-performance v commands, surrounded by calls to bgnpolygon and endpolygon.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 pdr calls between pmv and pclos.
NAME

**perspective** – defines a perspective projection transformation

C SPECIFICATION

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

PARAMETERS

- **fovy** expects the field-of-view angle in the y direction. The field of view is the range of the area that is being viewed. **fovy** must be \( \geq 2 \) or an error results.

- **aspect** expects the aspect ratio which determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).

- **near** expects the distance from the viewer to the closest clipping plane (always positive).

- **far** expects the distance from the viewer to the farthest clipping plane (always positive).

DESCRIPTION

**perspective** specifies a viewing pyramid into the world coordinate system. 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.

When the system is in single matrix mode, **perspective** loads a matrix onto the transformation stack, replacing the current top matrix. When the system is in viewing, projection, or texture matrix mode, **perspective** replaces the current Projection matrix and leaves the ModelView matrix stack and the Texture matrix unchanged.
SEE ALSO

mmode, ortho, viewport, window
NAME

pick – puts the system in picking mode

C SPECIFICATION

void pick(buffer, numnames)
short buffer[];
long numnames;

PARAMETERS

buffer expects the array to use for storing names.
numnames expects the maximum number of names to store.

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 cursor origin.

The graphical items that intersect the picking region are hits and store
the contents of the name stack in buffer. Picking does not work if you
issue a new viewport in picking mode.

SEE ALSO

eendpick, endselect, gselect, picksize, pushname, popname, loadname

NOTE

This routine is available only in immediate mode.
NAME

picksize — sets the dimensions of the picking region

C SPECIFICATION

void picksize(deltax, deltay)
short deltax, deltay;

PARAMETERS

deltax  expects the new width of the picking region.
deltay  expects the new height of the picking region.

DESCRIPTION

picksize changes the dimensions of the picking region. The default setting is 10 pixels. The picking region is rectangular and 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 picking buffer.

SEE ALSO

pick

NOTE

This routine is available only in immediate mode.
NAME

pixmode – specify pixel transfer mode parameters

C SPECIFICATION

void pixmode(mode, value)
long mode, value;

PARAMETERS

mode One of the symbolic constants:
(parameters that affect read, write, and copy transfers)

PM_SHIFT, default value: 0. Number of bit positions that pixel
data are to be shifted. Positive shifts are left for write and copy,
right for read. Valid values: 0, +1, +4, +8, +12, +16, +24

PM_EXPAND, default value: 0. Enable (1) or disable (0)
expansion of single-bit pixel data to one of two 32-bit pixel
values. Valid values: 0, 1

PM_C0, default value: 0. Expansion value (32-bit packed color)
chosen when the single-bit pixel being expanded is zero. Valid
values: any 32-bit value

PM_C1, default value: 0. Expansion value (32-bit packed color)
chosen when the single-bit pixel being expanded is one. Valid
values: any 32-bit value

PM_ADD24, default value: 0. Amount to be added to the least-
significant 24 bits of the pixel (signed value). Valid values: a
32-bit signed value in the range -0x800000 through 0x7ffffff

Although this value is specified as a 32-bit integer, the sign bit
MUST be smeared across all 32 bits. Thus -0x800000 specifies
the minimum value; and 0x800000 is out of range at the positive
end.

PM_TTOB, default value: 0. Specifies that fill (for write and
copy transfers) and read (for read transfers) must be top-to-
bottom (1) or bottom-to-top (0). Valid values: 0, 1
PM_RTOL, default value: 0. Specifies that fill (for write and copy transfers) and read (for read transfers) is to be right-to-left (1) or left-to-right (0). Valid values: 0, 1

PM_SIZE, default value: 32. Number of bits per pixel. Used for packing during reads and writes. Used to optimize internal transfers during copies. Valid values: 1, 4, 8, 12, 16, 24, 32

Although size specification is for the entire pixel, there is no mechanism for specifying reduced RGBA component sizes (such as 12-bit RGB with 4 bits per component).

(parameters that affect read and write transfers only)

PM_OFFSET, default value: 0. Number of bits of the first CPU word of each scanline that are to be ignored. Valid values: 0 through 31

PM_STRIDE, default value: 0. Number of 32-bit CPU words per scanline in the original image (not just the portion that is being transferred by this command). Valid values: any non-negative integer

(parameters that affect write and copy transfers only)

PM_ZDATA, default value: 0. Indicates (1) that pixel data are to be treated as Z data rather than color data (0). Destination is the Z-buffer. Writes are conditional if zbuffering is on. Valid values: 0, 1

value Integer value assigned to mode.

DESCRIPTION

pixmode allows a variety of pixel transfer options to be selected. These options are available only for pixel transfer commands that operate on 32-bit data: lrectread, lrectwrite, and rectcopy. Pixel transfer commands that operate on 8-bit data (readRGB, writeRGB) and on 16-bit data (readpixels, writepixels, rectread, rectwrite) do not support pixmode capabilities. Note that lrectread, lrectwrite, and rectcopy are valid in both color map and RGB modes.
Padding in CPU Memory

Transfer commands `lrectread` and `lrectwrite` operate on pixel data structures in CPU memory. These data structures contain data organized in row-major format, each row corresponding to one scanline of pixel data. Adjacent pixels are packed next to each other with no padding, regardless of the pixel size. Thus in many cases pixels straddle the 32-bit word boundaries. It is always the case, however, that each scan line comprises an integer number of whole 32-bit words. If the pixel data do not exactly fill these words, the last word is padded with (undefined) data.

Addresses passed to `lrectread` and `lrectwrite` must be long word aligned. If not, an error message is generated and no action is taken.

Packing in CPU Memory

Transfer commands `lrectread` and `lrectwrite` operate on pixel data that are packed tightly into CPU memory. Adjacent pixels, regardless of their size, are stored with no bit padding between them. Pixel size, and thus packing, is specified by `PM_SIZE`. The default value of this parameter is 32, meaning that 32-bit pixels are packed into 32-bit CPU memory words.

Although the MIPS processor is a big-endian machine, its bit numbering is little-endian. Pixel data are packed consistent with the byte numbering scheme (big-endian), ignoring the bit numbering. Thus, 12-bit packed pixels are taken as follows (by `lrectwrite`) from the first 32-bit word of a CPU data structure:
first CPU word

<table>
<thead>
<tr>
<th>byte number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit number</td>
<td>33222222</td>
<td>22221111</td>
<td>111111</td>
<td>10987654 32109876 54321098 76543210</td>
</tr>
</tbody>
</table>

first unpacked pixel 11
10987654 3210
second unpacked pixel 11
1098 76543210
third unpacked pixel 11
10987654 ...

When being written, unpacked pixels are padded to the left with zeros to make their size equal to the size of the framebuffer target region (12 bits total in color map mode, 24 bits total when writing Z values, 32 bits total in RGB mode). The least significant bit of the unpacked pixel becomes the least significant bit of the framebuffer pixel it replaces.

Note that big-endian packing makes 8 and 16 bit packing equivalent to character and short arrays. Remember, however, that the address passed to `lrectwrite` or `lrectread` must be long-word aligned.

Packings of 1, 4, 8, 12, 16, 24, and 32 bits per pixel are supported. Setting `PM_SIZE` to a value other than one of these results in an error message, and leaves the current size unchanged.

**Order of Pixel Operations**

In addition to packing and unpacking, pixel streams are operated on in a variety of other ways. These operations occur in a consistent order, regardless of whether the stream is being written, read, or copied.

- write unpack->shift->expand->add24->zoom->fbpack
- copy format->shift->expand->add24->zoom->fbpack
- read format->shift->expand->add24 ->pack

Note that pixel data are unpacked only when being transferred from CPU memory to the framebuffer, and that they are unpacked prior to any other operation. Likewise, pixel data are packed only when being transferred to CPU memory. Packing occurs after all other operations have been completed. Because copy operations neither pack nor unpack.
pixel data, the rectcopy command ignores the value of PM_SIZE.

Framebuffer Format

Each IRIS framebuffer is always configured in one of two fundamental ways: color map or RGB. In the RGB configuration 3 or 4 color components (red, green, blue, and optionally alpha) are stored at each pixel location. Each component is stored with a maximum of 8 bits of precision, resulting in a packed 32-bit pixel with the following format:

33222222 22221111 111111
10987654 32109876 54321098 76543210

aaaaaaa bbbbbbbbb gggggggg rrrrrrr
76543210 76543210 76543210 76543210

Some IRIS framebuffers store fewer than 8 bits per color component while in RGB mode. These framebuffers, however, emulate all the behavior of full 32-bit framebuffers. Thus the first operation in both the copy and read streams (above) is format: converting the framebuffer-format data to the 8-bit per component RGBA format that all subsequent operations execute with. Likewise, the final operation in both the write and copy streams (above) is fbpack: converting the 8-bit per component RGBA data back to the hardware-specific storage format. Both the format and the fbpack operation are null operations if the hardware supports full 32-bit RGBA data.

In its color map configuration a single color index, from 1 to 12 bits, is stored at each pixel location:

33222222 22221111 111111
10987654 32109876 54321098 76543210

iiii iiiiiiii
11
1098 76543210

Pixel Shifting
Pixels taken from the framebuffer (lrectread, rectcopy) or unpacked from CPU memory (lrectwrite) are first rotated either left or right by an amount up to 24 bit positions. Unpacked pixels and pixel values to be packed are padded left and right by zeros during the shift operation. The resulting 32-bit pixel values therefore include ones only in the region that was filled with legitimate pre-shifted data. Copied pixels may not be padded with zeros; thus a writemask may be required to eliminate unwanted bits.

Pixel shifting is enabled by setting PM_SHIFT to a non-zero value. Positive values in the range 1-24 specify left shifts while writing or copying, right shifts while reading. Negative values in the range -1 through -24 specify right shifts while writing or copying, left shifts while reading.

The default shift value is zero (i.e. shifting disabled). Other accepted values are plus and minus 1, 4, 8, 12, 16, and 24.

Because pixels are always converted to the formats described above before they are shifted, shift operations are largely independent of the hardware framebuffer storage format.

Pixel Expansion

Single bit pixels can be expanded to one of two full 32-bit color values, based on their binary value. This expansion is enabled by setting PM_EXPAND to 1 (the default disabled value is 0). When expansion is enabled, zero value pixels are replaced by the packed color PM_C0, and one value pixels are replaced by PM_C1. Bits 11-0 of PM_C0 and PM_C1 specify color index values when in color map mode.

Pixel expansion is actually controlled by bit zero of the incoming pixel (regardless of the size of the incoming pixel). Because pixel shifting preceeds pixel expansion, any bit of the incoming pixel can be selected to control pixel expansion.

There are no constraints on the values of PM_C0 or PM_C1.

Pixel Addition

The pixel addition stage treats the lower 24 bits of each incoming pixel as a signed integer value. It adds a signed 24-bit constant to this field of the pixel, leaving the upper 8 bits unchanged. The result of the addition is clamped to the range $-2^{23}$ through $2^{23} - 1$. While this addition is
most useful when writing or copying depth data, it is enabled during all transfers. Thus PM_ADD24 is typically changed from its default zero value only while depth transfers are being done (See "Drawing Z Data" below). Pixel addition can also be used to offset the range of a color map image.

Rectangle within Rectangle in CPU Memory

Variables PM_OFFSET and PM_STRIDE support transfer operation on rectangular pixels regions that reside within larger regions in CPU memory. PM_OFFSET, set to a value in the range 0-31, specifies the number of significant bits of the first CPU word that are ignored at the start of each scanline transfer. For example, an lrectwrite transfer of 12-bit packed pixel data with PM_OFFSET set to 12 results in the following pixel extraction:

first CPU word of each scanline

<table>
<thead>
<tr>
<th>byte number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit number</td>
<td>33222222</td>
<td>22221111</td>
<td>111111</td>
<td>10987654</td>
</tr>
</tbody>
</table>

first unpacked pixel
11
1098 76543210

second unpacked pixel
11
10987654 ...

Pixel unpacking continues tightly throughout all the CPU words that define a single scanline. After the last CPU word that defines a scanline has been transferred, the CPU read pointer is advanced to the 32-bit word at location (first PM_STRIDE). PM_OFFSET pixels of this word are skipped, then this scanline is transferred to the graphics engine. The PM_STRIDE value of zero is exceptional, causing the CPU read pointer to be advanced to the 32-bit word that immediately follows the last word of each scanline.

PM_OFFSET and PM_STRIDE, like PM_SIZE, are ignored by framebuffer-to-framebuffer transfers (rectcopy). They both default to a value of zero.
Alternate Fill Directions

During read, copy, and write pixel operations pixels are always transferred in row-major order. By default scanlines are read or written left-to-right, starting with the bottom scanline and working up. Parameters PM_RTOL and PM_TTOB allow the horizontal and vertical read/fill directions to be reversed, but do not change the fundamental row-major scan order. PM_RTOL specifies right-to-left traversal/fill when set to one, left-to-right when set to its default value of zero. PM_TTOB specifies top-to-bottom traversal/fill when set to one, bottom-to-top when set to its default value of zero.

These parameters can be used to properly deal with CPU data formats that differ from the default IRIS pixel order. They also can be used to generate image reflections about either the X or Y screen axes. (see notes.)

Fill direction does not affect the location of the destination rectangle (i.e. the destination rectangle is always specified by its lower-left pixel, regardless of its traversal/fill direction).

Drawing Z Data

Normally pixel data are treated as colors. Zbuffer mode must be false during rectwrite and rectcopy of color values, because there are no source Z values to do the buffer compares with. Setting PM_ZDATA to 1.0, however, instructs the GL to treat incoming pixel values as Z values, and to treat source color as undefined. When drawing pixels with PM_ZDATA enabled, the system automatically insures that no changes are made to color bitplanes, regardless of the current color write mask. When PM_ZDATA and zbuffer are both enabled, pixel values will be conditionally written into the z-buffer in the usual manner, and the color buffer will be unaffected.

Z-buffered images are drawn by doing two transfers, first of the Z values (with PM_ZDATA enabled and stencil set based on the outcome of the Z comparison) and then of the color values (with PM_ZDATA disabled, drawn conditionally based on the stencil value).

It is not necessary (or correct) to enable zdraw mode while doing pixel transfers with PM_ZDATA enabled.
SEE ALSO
lrectread, lrectwrite, rectcopy, rectzoom, stencil

NOTES
IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support pixmode.
NAME

pmv, pmvi, pmvs, pmv2, pmv2i, pmv2s – specifies the first point of a polygon

C SPECIFICATION

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

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

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

void pmv2(x, y)
Coord x, y;

void pmv2i(x, y)
Icoord x, y;

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

All of the above routines are functionally the same. They differ only in the type declarations of their parameters and in whether or not they assume a two-dimensional or three-dimensional space.

PARAMETERS

x expects the x coordinate of the first point if a polygon.
y expects the y coordinate of the first point of a polygon.
z expects the z coordinate of the first point of a polygon.

DESCRIPTION

pmv specifies the starting point of a polygon. The next drawing command will start drawing from this point. You draw a typical polygon with a pmv, a sequence of pdr, and close it with a pclos.
Between `pmv` and `pclos`, you can only issue the following Graphics Library subroutines: `c`, `color`, `RGBcolor`, `cpack`, `lmdef`, `lmbind` `n`, `pdr`, and `v`. Only use `lmdef` and `lmbind` to respecify materials and their properties.

**EXAMPLE**

The following sequence draws a square:

```c
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();
```

**SEE ALSO**

`bgnpolygon`, `endpolygon`, `pclos`, `pdr`, `v`

**NOTES**

`pmv` should not be used in new development. Rather, polygons should be drawn using the high-performance `v` commands, surrounded by calls to `bgnpolygon` and `endpolygon`.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 `pdr` calls between `pmv` and `pclos`. 
NAME

pnt, pnti, pnts, pnt2, pnt2i, pnt2s – draws a point

C SPECIFICATION

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

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

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

void pnt2(x, y)
Coord x, y;

void pnt2i(x, y)
Icoord x, y;

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

All of the above routines are functionally the same. They differ only in
the type declarations of their parameters and in whether or not they
assume a two-dimensional or three-dimensional space.

PARAMETERS

x  expects the x coordinate of the point to be drawn.
y  expects the y coordinate of the point to be drawn.
z  expects the z coordinate of the point to be drawn.

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. A drawing routine immediately following pnt will start draw-
ing from the point specified.
SEE ALSO

bgnpoint, endpoint, v

NOTE

pnt should not be used in new development. Rather, points should be drawn using the high-performance v commands, surrounded by calls to bgnpoint and endpoint.
NAME

pntsmooth – specify antialiasing of points

C SPECIFICATION

void pntsmooth(mode)
unsigned long mode;

PARAMETERS

mode expects one of two values:

SMP_OFF defeats antialiasing of points (default).

SMP_ON enables antialiasing of points. It can be modified by
an optional symbolic constant:

SMP_SMOOTHER indicates that a higher quality filter should
be used during point drawing. This filter typically requires that
more pixels be modified, and therefore potentially reduces the
rate at which antialiased points are rendered.

The constant SMP_SMOOTHER is specified by bitwise ORing it, or by
adding it, to SMP_ON. For example:

pntsmooth(SMP_ON + SMP_SMOOTHER);

enables antialiased point drawing with the highest quality, and poten-
tially lowest performance, algorithm. The modifier is a hint, not a direc-
tive, and is therefore ignored by systems that do not support the
requested feature.

DESCRIPTION

pntsmooth controls the capability to draw antialiased points. You can
draw antialiased points in either color map mode or RGB mode.

For color map antialiased points to draw correctly, you must initialize a
16-entry colormap block (whose lowest entry location is a multiple of
16) to a ramp between the background color (lowest index) and the
point color (highest index). Before drawing points, clear the area to the
background color.
The pntsmooth hardware replaces the least significant 4 bits of the current color index with bits that represent pixel coverage. Therefore, by changing the current color index (only the upper 8 bits are significant) you can select among many 16-entry color ramps, representing different colors and intensities. You can draw depthcued antialiased points in this manner.

The z-buffer hardware can be used to improve the quality of color map antialiased point images. Enabled in the standard depth-comparison mode, it ensures that points nearer the viewer obscure more distant points. Alternately, the z-buffer hardware can be used to compare color values by issuing:

```c
zbuffer(TRUE);
zsource(ZSRC_COLOR);
zfunction(ZF_GREATER);
```

Pixels are then replaced only by 'brighter' values, resulting in better intersections between points drawn using the same ramp.

RGB antialiased points can be drawn only on machines that support blending. For these points to draw correctly, the blendfunction must be set to merge new pixel color components into the framebuffer using the incoming (source) alpha values. Incoming color components should always be multiplied by the source alpha (BF_SA). Current (destination) color components can be multiplied either by one minus the source alpha (BF_MSA), resulting in a weighted average blend, or by one (BF_ONE), resulting in color accumulation to saturation; issue:

```c
blendfunction(BF_SA, BF_MSA); /* weighted average */
```

or

```c
blendfunction(BF_SA, BF_ONE); /* saturation */
```

The pntsmooth hardware scales incoming alpha components by an 8-bit computed coverage value. Therefore reducing the incoming source alpha results in transparent, antialiased points.

RGB antialiased points draw correctly over any background image. It is not necessary to clear the area in which they are to be drawn.
Both color map and RGB mode points are antialiased effectively only when subpixel mode is enabled; issue:

    subpixel(TRUE);

The modifier SMP_SMOOTHER can be ORed or ADDED to the symbolic constant SMP_ON when antialiased points are enabled. When this is done, a higher quality and potentially lower performance filter is used to scan convert antialiased points. SMP_SMOOTHER is a hint, not a directive. Thus a higher quality filter is used only if it is available.

SEE ALSO

bgnpoint, blendfunction, endpoint, gversion, linesmooth, subpixel, v, zbuffer, zfunction, zsource

NOTES

This routine does not function on IRIS-4D B or G models, or on early serial numbers of the Personal Iris. Use gversion to determine which type you have.

IRIS-4D GT and GTX models, and the Personal Iris, do not support SMP_SMOOTHER. These systems ignore this hint.

BUGS

Color-comparison z-buffering is not supported on the IRIS-4D GT or GTX models.
NAME

polarview – defines the viewer's position in polar coordinates

C SPECIFICATION

void polarview(dist, azim, inc, twist)
Coord dist;
Angle azim, inc, twist;

PARAMETERS

dist expects the distance from the viewpoint to the world space origin.

azim expects the azimuthal angle in the x-y plane, measured from the y axis. The azimuth angle is the viewing angle of the observer.

inc expects the angle of incidence in the y-z plane, measured from the z axis. The incidence angle is the angle of the viewport relative to the z axis.

twist expects the amount that the viewpoint is to be rotated around the line of sight using the right-hand rule.

DESCRIPTION

polarview defines the viewer's position in polar coordinates. It sets up a right-hand world coordinate system with x to the right, y straight up, and z towards the viewer. The line of sight extends from the viewpoint through the world space origin. All angles are specified in tenths of degrees and are integers.

The matrix computed by polarview premultiplies the current matrix, which is chosen based on the current matrix mode.

SEE ALSO

mmode, lookat
NAME

colf, polfi, polfs, colf2, polf2i, polf2s — draws a filled polygon

C SPECIFICATION

void colf(n, parey)
long n;
Coord parey[][3];

void polfi(n, parey)
long n;
Icoord parey[][3];

void polfs(n, parey)
long n;
Scoord parey[][3];

void colf2(n, parey)
long n;
Coord parey[][2];

void colf2i(n, parey)
long n;
Icoord parey[][2];

void polf2s(n, parey)
long n;
Scoord parey[][2];

All of the above routines are functionally the same. They differ only in
the declared type of their parameters and whether or not they assume a
two-dimensional or three-dimensional world.

PARAMETERS

n expects the number of points in the polygon.

parey expects the array containing the vertices of the polygon.

DESCRIPTION

colf fills polygonal areas using the current pattern, color, and writemask.
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.

There can be no more than 256 points (corners) in a polygon. In addition, \texttt{polf} cannot correctly draw polygons that intersect themselves.

SEE ALSO

concave, poly, rect, rectf, pdr, pmv, rpdr, rpmv
NAME

poly, polyi, polys, poly2, poly2i, poly2s – outlines a polygon

C SPECIFICATION

```c
void poly(n, pararray)
  long n;
  Coord pararray[][3];

void polyi(n, pararray)
  long n;
  Icoord pararray[][3];

void polys(n, pararray)
  long n;
  Scoord pararray[][3];

void poly2(n, pararray)
  long n;
  Coord pararray[][2];

void poly2i(n, pararray)
  long n;
  Icoord pararray[][2];

void poly2s(n, pararray)
  long n;
  Scoord pararray[][2];
```

All of the above routines are functionally the same. They differ only in the type declarations of their parameters and in whether they assume a two- or three-dimensional space.

PARAMETERS

- \( n \) expects the number of points in the polygon.
- \( pararray \) expects the array containing the vertices of the polygon.

DESCRIPTION

poly outlines a polygon. 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 256.

SEE ALSO

polf, rect, rectf, pmv, pdr, pclos, rpmv, rpdr
NAME

polymode – control the rendering of polygons

C SPECIFICATION

void polymode(mode)
long mode;

PARAMETERS

mode  Expects one of the symbolic constants:
       PYM_POINT, draw only points at the vertices.
       PYM_LINE, draw lines from vertex to vertex.
       PYM_FILL, fill the polygon interior.
       PYM_HOLLOW, fill only interior pixels at the boundaries.

DESCRIPTION

polymode specifies whether polygons are filled, outlined, drawn with
points at their vertices, or outlined with a hollow fill algorithm.
Affected polygons include all polygons that are normally filled, including
those generated by bgnpolygon and endpolygon, by bgntmesh and
endtmesh, by bgnqstrip and endqstrip, by arcf, circf, and rectf, and
by NURBS surfaces. Also affected are polygons generated by the
obsolete pmv, pdr, and pclos commands, and by polf and spolf.

PYM_FILL is the default mode. In this mode polygons are filled with
a point-sample algorithm. (Refer to the Graphics Library Programmer’s
Guide for an detailed explanation of point-sampling.)

PYM_POINT causes a single point to be drawn at each polygon vertex,
including vertices generated by clipping. All point rendering modes,
including antialiasing specified by pntsmooth, apply to these points.

PYM_LINE causes lines to be drawn from vertex to vertex around the
perimeter of the polygon. This line forms a single outline, because it
passes through both projected vertices and vertices generated by clip-
ing. All line rendering modes, including line width, line stipple style,
and line antialiasing specified by linesmooth, apply to these lines.
**PYM_HOLLOW** supports a special kind of polygon fill with the following properties:

1. Only pixels on the polygon edge are filled. These pixels form a single-width line (regardless of the current linewidth) around the inner perimeter of the polygon.

2. Only pixels that would have been filled (**PYM_FILL**) are changed (i.e. the outline does not extend beyond the exact polygon boundary).

3. Pixels that are changed take the *exact* color and depth values that they would have had the polygon been filled.

Because their pixel depth values are exact, hollow polygons can be composed with filled polygons accurately. Both hidden-line and scribed-surface renderings can be done taking advantage of this fact.

**SEE ALSO**

bgnpolygon, endpolygon, polysmooth, stencil

**NOTES**

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support **polymode**. Use **getgdesc** to determine whether support for **polymode** is available.

**BUGS**

In order to support polygon fill mode **PYM_HOLLOW**, IRIS-4D VGX models require that the following conditions be met:

1. Stencil planes be allocated (at least one plane).

2. The stencil planes be initialized to zero prior to drawing hollow polygons.

3. The stencil planes be used for no other purpose while drawing hollow polygons. (The stencil function is controlled by the hardware and must not be user specified.)
IRIS-4D VGX models have an error in their microcode that results in matching errors between pixels generated by PYM_FILL and PSM_HOLLOW in some conditions. This error will be corrected in the next software release.
NAME

polysmooth – specify antialiasing of polygons

C SPECIFICATION

void polysmooth(mode)
long mode;

PARAMETERS

mode  Expects one of the symbolic constants:

PYSM_OFF: do not antialias polygons. (default)

PYSM_ON: compute coverage values for all perimeter polygon pixels in such a way as to not change the size of the polygon.

PYSM_SHRINK: Compute coverage values for all perimeter polygon pixels in such a way as to shrink the polygon slightly.

DESCRIPTION

polysmooth specifies one-pass antialiasing of polygons. Unlike pointsmooth and linesmooth, it is available only in RGB mode. Also, unlike pointsmooth and linesmooth, its use in complex scenes requires attention to primitive drawing order if acceptable results are to be achieved. Thus polysmooth use is somewhat more complex than that of pointsmooth and linesmooth.

Like points and lines, polygons are antialiased by computing a coverage value for each scan-converted pixel, and using this coverage value to scale pixel alpha. Thus, for RGB antialiased polygons to draw correctly, blendfunction must be set to merge new pixel color components with the previous components using the incoming alpha. In the simplistic case of adding a single, antialiased polygon to a previously rendered scene, the same blendfunction as is typically used for point and line antialiasing can be used:

blendfunction(BF_SA, BF_MSA).

Pixels in the interior of the polygon will have coverage assigned to 1.0, and will therefore replace their framebuffer counterparts. Pixels on the perimeter of the polygon are blended into the framebuffer in proportion to their computed coverage.
A more typical case, however, is that of antialiasing the polygons that comprise the surface of a solid object. Here the standard blendfunction will result in 'leakage' of color between adjacent polygons. For example, if the first polygon drawn covers a sample pixel 40%, and the second (adjacent) polygon covers the pixel 60%, the net coverage of %100 still leaves %24 background color in the pixel.

If the solid object is to be correctly antialiased, with no leakage through interior edges, and with proper silhouettes, the following rules must be followed:

1. Polygons must be drawn in view order from nearest to farthest. (Not farthest to nearest as is done with transparency.)
2. Polygons that face away from the viewer must not be drawn. (Use backface(TRUE).)
3. The special blendfunction(BF_MIN_SA_MDA, BF_ONE) must be used to blend polygons into the framebuffer.
4. Polysmooth mode PYSM_ON must be used.

The special polysmooth mode PYSM_SHRINK specifies a coverage algorithm that includes only pixels that would have been scan-converted had the mode been PYSM_OFF. (PYSM_ON includes pixels that are outside that range of those point-sampled by the PYSM_OFF algorithm.) PYSM_SHRINK necessarily leaks background color between adjacent polygons, but does this in a way that resembles antialiased lines. Thus, PYSM_SHRINK can be used in conjunction with blendfunction(BF_SA, BF_ZERO), and with no sorting of polygons (use the z-buffer), to generate solid images tesselated with black, antialiased lines.

SEE ALSO

linesmooth, pntsmoth, blendfunction, subpixel

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support polysmooth. getgdesc to determine whether polysmooth is supported.
subpixel mode should always be enabled while polysmooth is used.

BUGS

IRIS-4D VGX models reveal their decomposition of 4+ sided polygons into triangles when PYSM_SHRINK is selected. This behavior is not intended, and may not be duplicated by future VGX software releases, or by future models.
NAME

`popattributes` – pops the attribute stack

C SPECIFICATION

```c
void popattributes()
```

PARAMETERS

`none`

DESCRIPTION

`popattributes` pops the attribute stack, restoring the values of the global state attributes listed below that were most recently saved with `pushattributes`.

<table>
<thead>
<tr>
<th>Attributes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>backbuffer</td>
<td>linewidth</td>
</tr>
<tr>
<td>cmode or RGBmode</td>
<td>lsrepeat</td>
</tr>
<tr>
<td>color</td>
<td>pattern</td>
</tr>
<tr>
<td>drawmode</td>
<td>font</td>
</tr>
<tr>
<td>frontbuffer</td>
<td>RGB color</td>
</tr>
<tr>
<td>linestyle</td>
<td>RGB writemask</td>
</tr>
<tr>
<td>writemask</td>
<td>shademodel</td>
</tr>
</tbody>
</table>

SEE ALSO

backbuffer, cmode, color, drawmode, frontbuffer, linewidth, lsrepeat, pushattributes, RGBcolor, RGBwritemask, setlinestyle, setpattern, shademodel, writemask
NAME

`popmatrix` - pops the transformation matrix stack

C SPECIFICATION

`void popmatrix()`

PARAMETERS

`none`

DESCRIPTION

`popmatrix` pops the transformation matrix stack. It operates on the single transformation stack when `mmode` is `MSINGLE`, and on the ModelView matrix stack when `mmode` is `MVIEWING`. It should not be called when `mmode` is `MPROJECTION` or `MTEXTURE`.

`popmatrix` is ignored when there is only one matrix on the stack.

SEE ALSO

getmatrix, loadmatrix, mmode, multmatrix, pushmatrix
NAME

**popname** — pops a name off the name stack

C SPECIFICATION

```c
void popname()
```

PARAMETERS

*none*

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*
NAME

popviewport – pops the viewport stack

C SPECIFICATION

void popviewport()

PARAMETERS

none

DESCRIPTION

popviewport pops the viewport stack, restoring the values of the viewport, screenmask, and depth range most recently saved with pushviewport.

SEE ALSO

isetdepth, pushviewport, scrmask, viewport
NAME

defposition – specifies the preferred location and size of a graphics window

C SPECIFICATION

void defposition(x1, x2, y1, y2)
long x1, x2, y1, y2;

PARAMETERS

x1 expects the x coordinate position (in pixels) of the point at which one corner of the window is to be.
x2 expects the x coordinate position (in pixels) of the point at which the opposite corner of the window is to be.
y1 expects the y coordinate position (in pixels) of the point at which one corner of the window is to be.
y2 expects the y coordinate position (in pixels) of the point at which the opposite corner of the window is to be.

DESCRIPTION

defposition specifies the preferred location and size of a graphics window. You specify the location in pixels (x1, x2, y1, y2). Call defposition at the beginning of a graphics program. Use defposition with winconstraints to modify the enforced size and location after the window has been created. Calling winopen activates the constraints specified by defposition. If winopen is not called, defposition is ignored.

SEE ALSO

winconstraints, winopen

NOTE

This routine is available only in immediate mode.
NAME

prefsize – specifies the preferred size of a graphics window

C SPECIFICATION

void prefsize(x, y)
long x, y;

PARAMETERS

x  expects the width of the graphics window. The width is measured in pixels.
y  expects the height of the graphics window. The height is measured in pixels.

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.

Once a window is created, you must use prefsize with winconstraints in order to modify the enforced window size. Calling winopen activates the constraints specified by prefsize. If winopen is not called, prefsize is ignored.

SEE ALSO

winconstraints, winopen

NOTE

This routine is available only in immediate mode.
NAME

pupmode, endpupmode – obsolete routines

C SPECIFICATION

void pupmode()
void endpupmode()

PARAMETERS

none

DESCRIPTION

These routines are obsolete. Although pupmode/endpupmode continue to function (to provide backwards compatibility) all new development should use drawmode to access the pop-up menu bitplanes.

SEE ALSO

drawmode
NAME

pushattributes — pushes down the attribute stack

C SPECIFICATION

void pushattributes()

PARAMETERS

none

DESCRIPTION

pushattributes pushes down the attribute stack, duplicating the following global state attributes:

<table>
<thead>
<tr>
<th>Attributes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>backbuffer</td>
<td>linewidth</td>
</tr>
<tr>
<td>cmode or RGBmode</td>
<td>lsrepeat</td>
</tr>
<tr>
<td>color</td>
<td>pattern</td>
</tr>
<tr>
<td>drawmode</td>
<td>font</td>
</tr>
<tr>
<td>frontbuffer</td>
<td>RGB color</td>
</tr>
<tr>
<td>linestyle</td>
<td>RGB writemask</td>
</tr>
<tr>
<td>writemask</td>
<td>shademodel</td>
</tr>
</tbody>
</table>

The saved values can be restored using popattributes.

The attribute stack is ATTRIBSTACKDEPTH levels deep. pushattributes is ignored if the stack is full.

SEE ALSO

backbuffer, cmode, color, drawmode, frontbuffer, linewidth, lsrepeat, popattributes, RGBcolor, RGBmode, RGBwritemask, setlinestyle, setpattern, shademodel, writemask
NAME

**pushmatrix** -- pushes down the transformation matrix stack

C SPECIFICATION

```c
definition
```

PARAMETERS

**none**

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 stack contains two copies of \( M \). Only the top copy can be modified.

**pushmatrix** operates on the single transformation stack when **mmode** is **MSINGLE**, and on the ModelView matrix stack when **mmode** is **MVIEWING**. It should not be called when **mmode** is **MPROJECTION** or **MTEXTURE**.

SEE ALSO

getmatrix, loadmatrix, mmode, multmatrix, popmatrix
NAME

pushname – pushes a new name on the name stack

C SPECIFICATION

void pushname(name)
short name;

PARAMETERS

name expects the name which is to be added onto the name stack.

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, popname, loadname, pick
NAME

pushviewport – pushes down the viewport stack

C SPECIFICATION

void pushviewport()

PARAMETERS

none

DESCRIPTION

pushviewport pushes down the viewport stack, duplicating the current viewport, screenmask, and depth range. These saved values can be restored using popviewport.

The viewport stack is VPSTACKDEPTH levels deep. pushviewport is ignored if the stack is full.

SEE ALSO

lsetdepth, popviewport, scrmask, viewport
NAME

pwlcurve – describes a piecewise linear trimming curve for NURBS surfaces

C SPECIFICATION

void pwlcurve(n, data_array, byte_size, type)
long n, byte_size, type;
double *data_array;

PARAMETERS

$n$ expects the number of points on the curve
$data_array$ expects an array containing the curve points
$byte_size$ expects the offset (in bytes) between points on the curve
$type$ expects a value indicating the point type. Currently, the only data type supported is N_ST, corresponding to pairs of s-t coordinates. The offset parameter is used in case the curve points are part of an array of larger structure elements. pwlcurve searches for the $n$-th coordinate pair beginning at $data_array + n \times byte_size$.

DESCRIPTION

Use pwlcurve to describe a piecewise linear curve. A piecewise linear curve consists of a list of pairs of coordinates of points in the parameter space for the NURBS surface. These points are connected together with straight lines to form a path. If a piecewise linear curve is an approximation to a real curve, the points should be close enough together that the resulting path will appear curved at the resolution used in the application.

You use piecewise linear curves within trimming loop definitions. A trimming loop definition is a set of oriented curve commands that describe a closed loop. To mark the beginning of a trimming loop definition, use the bgntrim command. To mark the end of a trimming loop definition, use an endtrim command.
You use trimming loop definitions within NURBS surface definitions (see \texttt{bgnsurface}). The trimming loops are closed curves that the system uses to set the boundaries of a NURBS surface. You can describe a trimming loop by using a series of piecewise linear curves or NURBS curves (see \texttt{nurbscurve}), or both.

When the system needs to decide which part of a NURBS surface you want it to display, it displays the region of the NURBS surface that is to the left of the trimming curves as the parameter increases. Thus, for a counter-clockwise oriented trimming curve, the displayed region of the NURBS surface is the region inside the curve. For a clockwise oriented trimming curve, the displayed region of the NURBS surface is the region outside the curve.

See the \textit{Graphics Library Programming Guide} for a mathematical description of a NURBS curve.

\textbf{SEE ALSO}

\texttt{bgnsurface, nurbsurface, bgntrim, nurbscurve, getnurbsproperty, setnurbsproperty}
NAME

qcontrol – administers event queue

C SPECIFICATION

long qcontrol(cmd, icnt, idata, ocnt, odata)
long cmd;
long icnt;
short idata[];
long ocnt;
short odata[];

PARAMETERS

cmd specifies which operation to perform.
icnt expects the number of elements in idata. If the operation only
returns data, set icnt to zero.
idata expects an array containing the data to be used by cmd. If icnt is
zero, then idata can be NULL.
ocnt expects the number of elements in odata. If the operation does
not return data, set ocnt to zero.
odata expects the array into which you want the system to write the
values returned by cmd. If ocnt is zero, then odata can be
NULL.

FUNCTION RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, the
returned value is a negative integer whose absolute value is an error
value defined in <errno.h>.

DESCRIPTION

The qcontrol routine is used to control various administrative aspects of
the event queue and is intended for use by window management systems
and input device daemons.
The following values for \textit{cmd} are currently recognized:

**QC_ADDDEVICE**

Registers the device number given by \textit{idata[0]} as a new input device with flags specified in \textit{idata[1]}. All input devices must be added in this manner before \textbf{QC_CHANGEDEVICE} (described below) may be used on them. The 16-bit device name space is partitioned as follows:

\begin{align*}
0x0000 & \rightarrow 0x0FFF & \text{Devices defined by SGI} \\
0x0001 & \rightarrow 0x00FF & \text{Buttons} \\
0x0100 & \rightarrow 0x01FF & \text{Valuators} \\
0x0200 & \rightarrow 0x02FF & \text{Pseudo devices} \\
0x0300 & \rightarrow 0x0EFF & \text{Reserved} \\
0x0F00 & \rightarrow 0x0FFF & \text{Additional buttons} \\
0x1000 & \rightarrow 0x7FFF & \text{Devices defined by users} \\
0x1000 & \rightarrow 0x2FFF & \text{Buttons} \\
0x3000 & \rightarrow 0x3FFF & \text{Valuators} \\
0x4000 & \rightarrow 0x7FFF & \text{Pseudo devices} \\
0x8000 & \rightarrow 0xFFFF & \text{Can not be used}
\end{align*}

Possible values for flags are given in \texttt{<gl/qcontrol.h>}.

**QC_SETMOUSEWARP**

Sets the mouse acceleration threshold and multiplier to \textit{idata[0]} and \textit{idata[1]} respectively. Whenever the mouse is moved, a delta value from its last known position is computed. If the delta value exceeds the acceleration threshold, then both the \(x\) and \(y\) components of the motion are multiplied by the acceleration multiplier.

**QC_GETMOUSEWARP**

Returns in \textit{odata[0]} and \textit{odata[1]} the current mouse accelerator threshold and multiplier values.

**QC_SETKEYWARP**

Sets the graphics console keyboard auto-repeat threshold and rate to \textit{idata[0]} and \textit{idata[1]} respectively. The threshold represents the time between the initial key press and the beginning of auto-repeat. The rate value represents the inter-character repeat interval.
QC_GETKEYWARP
Returns in the keyboard repeat threshold in \textit{odata}[0] and the repeat rate in \textit{odata}[1].

QC_SETFOCUS
Sets the input driver focus to the sub-channel given by \textit{idata}[0]. This call is intended to be used only by the window management system and may give unpredictable results if called by another program.

QC_GETFOCUS
Returns the current focus input sub-channel number in \textit{odata}[0].

QC_CHANGEDEVICE
The array of device-value pairs given by \textit{(idata}[0], \textit{idata}[1]) \ldots are used to alter the current state of buttons or valuators. The input system may send events to the window management system or GL clients based on the device type as a result of this call. The list of device-value pairs is entered as an atomic group into the input system. The device NULLDEV signifies the end of the list and must be included in the value for \textit{icnt}. The use of QC_CHANGEDEVICE on the MOUSEX, MOUSEY, and DIAL0..7 devices is currently undefined.

QC_SENDEVEN
A group of device-value pairs specified by \textit{(idata}[2], \textit{idata}[3]) \ldots is sent to the input sub-channel whose number is given in \textit{idata}[0]. The number of pairs is specified by \textit{idata}[1]. This call is intended to be used only by the window management system.

SEE ALSO
qdevice, setvaluator

NOTES
This routine is available only in immediate mode.

The symbolic command values mentioned above are defined in \texttt{<gl/qcontrol.h>}. 

NAME

qdevice – queues a device

C SPECIFICATION

void qdevice(dev)
    Device dev;

PARAMETERS

dev    expects the device whose state is to be changed so that it will enter
       events into the event queue.

DESCRIPTION

qdevice changes the state of the specified device so that events occurring
within the device are entered in the event queue. The event queue
contains a time-ordered list of input events. The device can be the key-
board, a button, a valuator, or certain other pseudo-devices.

The maximum number of queue entries is 101.

SEE ALSO

noise, tie, unqdevice

NOTE

This routine is available only in immediate mode.
NAME
qenter — creates an event queue entry

C SPECIFICATION

void qenter(dev, val)
Device dev;
short val;

PARAMETERS

dev expects the device number to be entered into the event queue.
val expects the value to be entered into the event queue.

DESCRIPTION

qenter takes a device number and a value and enters them into the event queue of the calling process. There is no way to distinguish user-generated and system-generated events except by device number.

The 16-bit device number name space is partitioned as follows:

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 → 0x0FFF</td>
<td>Devices defined by SGI</td>
</tr>
<tr>
<td>0x0001 → 0x00FF</td>
<td>Buttons</td>
</tr>
<tr>
<td>0x0100 → 0x01FF</td>
<td>Valuators</td>
</tr>
<tr>
<td>0x0200 → 0x02FF</td>
<td>Pseudo devices</td>
</tr>
<tr>
<td>0x0300 → 0x0EFF</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x0F00 → 0x0FFF</td>
<td>Additional buttons</td>
</tr>
<tr>
<td>0x1000 → 0x7FFF</td>
<td>Devices defined by users</td>
</tr>
<tr>
<td>0x1000 → 0x2FFF</td>
<td>Buttons</td>
</tr>
<tr>
<td>0x3000 → 0x3FFF</td>
<td>Valuators</td>
</tr>
<tr>
<td>0x4000 → 0x7FFF</td>
<td>Pseudo devices</td>
</tr>
<tr>
<td>0x8000 → 0xFFFF</td>
<td>Can not be used</td>
</tr>
</tbody>
</table>

SEE ALSO
qcontrol, qread, qreset, qtest
NOTE
This routine is available only in immediate mode.
NAME

qgetfd – returns the file descriptor of the event queue

C SPECIFICATION

long qgetfd()

PARAMETERS

none

FUNCTION RETURN VALUE

The returned function value is the file descriptor associated with the event queue. If there was an error, the returned value is a negative integer whose absolute value is an error value defined in <errno.h>.

DESCRIPTION

qgetfd returns the file descriptor associated with the event queue. The file descriptor can then be used with the select(2) system call.

SEE ALSO

qread


NOTES

This routine is available only in immediate mode.

This routine is not available in the Distributed Graphics Library.
NAME
qread – reads the first entry in the event queue

C SPECIFICATION
long qread(data)
short *data;

PARAMETERS

data
    expects a pointer to the location that is to receive the data the event queue.

FUNCTION RETURN VALUE
The returned function value is the identifier for the device read.

DESCRIPTION
When there is an entry in the queue, qread returns the device number of the queue entry, writes the data of the entry into data, and removes the entry from the queue.

If there is not an entry in the queue, qread will block and return when an entry is made.

SEE ALSO
qreset, qtest

NOTE
This routine is available only in immediate mode.
NAME
qreset – empties the event queue

C SPECIFICATION
void qreset()

PARAMETERS
none

DESCRIPTION
qreset removes all entries from the event queue and discards them.

SEE ALSO
qenter, qread, qtest

NOTE
This routine is available only in immediate mode.
NAME

qtest – checks the contents of the event queue

C SPECIFICATION

long qtest()

PARAMETERS

none

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

qcenter, qread, qreset

NOTE

This routine is available only in immediate mode.
NAME

crv – draws a rational curve

C SPECIFICATION

void crv(geom)
Coord geom[4][4];

PARAMETERS

geom expects the array containing the four control points of the curve segment.

DESCRIPTION

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

The curve segment is approximated by a sequence of straight lines. All lines use the current linestyle, which is reset prior to the first line and continues through subsequent lines. Other line modes, including depthcuing, line width, and line antialiasing, also apply to the lines generated by crv.

After crv executes, the graphics position is undefined.

SEE ALSO

crv, crvn, curvebasis, curveprecision, defbasis, depthcue, linesmooth, linewidth, crvn, setlinestyle
NAME

rcrvn — draws a series of curve segments

C SPECIFICATION

void rcrvn(n, geom)
long n;
Coord geom[][4];

PARAMETERS

n expects the number of control points to be used in drawing the curve.

geom expects the matrix containing the control points of the curve segments.

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.

Each curve segment is approximated by a sequence of straight lines. All lines use the current linestyle, which is reset prior to the first line and continues through subsequent lines. Other line modes, including depthcuing, line width, and line antialiasing, also apply to the lines generated by rcrvn.

After rcrvn executes, the graphics position is undefined.

SEE ALSO

crv, crvn, curvebasis, curveprecision, defbasis, depthcue, linesmooth, linewidth, rcrv, setlinestyle
 NAME

rdr, rdri, rdrs, rdr2, rdr2i, rdr2s – relative draw

C SPECIFICATION

void rdr(dx, dy, dz)
Coord dx, dy, dz;

void rdri(dx, dy, dz)
Icoord dx, dy, dz;

void rdrs(dx, dy, dz)
Scoord dx, dy, dz;

void rdr2(dx, dy)
Coord dx, dy;

void rdr2i(dx, dy)
Icoord dx, dy;

void rdr2s(dx, dy)
Scoord dx, dy;

All of the above routines are functionally the same. They differ only in
the type declarations of their parameters and whether or not they assume
a two- or three-dimensional space.

PARAMETERS

dx  expects the distance from the x coordinate of the current graphics
    position to the x coordinate of the new point.

dy  expects the distance from the y coordinate of the current graphics
    position to the y coordinate of the new point.

dz  expects the distance from the z coordinate of the current graphics
    position to the z coordinate of the new point.

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

Version 4.0  - 1 -  April 1990
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

bgnline, endline, popmatrix, pushmatrix, rmv, translate, v

NOTE

rdr should not be used in new development. Rather, lines should be drawn using the high-performance v commands, surrounded by calls to bgnline and endline. Matrix commands pushmatrix, translate, and popmatrix should be used to accomplish relative positioning.
NAME

readpixels – returns values of specific pixels

C SPECIFICATION

long readpixels(n, colors)
short n;
Colorindex colors[];

PARAMETERS

n expects the number of pixels to be read by the function.

colors expects the array in which the pixel values are to be stored.

FUNCTION RETURN VALUE

The returned value of this function is the number of pixels actually read.
A returned function value of 0 indicates an error, that the starting point
is not a valid character position.

DESCRIPTION

readpixels attempts to read up to n pixel values from the bitplanes in
color map mode. 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 read,
which is the number requested if the starting point is a valid character
position (inside the current viewport). readpixels returns zero if the
starting point is not a valid character position. The values of pixels read
outside the viewport or 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 by default. On
machines that support it, you can use readsoure to control which
buffer is read.

The system must be in color map mode for readpixels to function
correctly.
SEE ALSO
lrectread, readsource

NOTES

readpixels should not be used in new development. Rather, pixels should be read using the high-performance lrectread command.

This routine is available only in immediate mode.

The upper bits of a color value (an element of the colors array) are undefined. You can write this information to the frame buffer without problems. However, if you intend to interpret this data, be sure you mask out the upper bits.
NAME

readRGB – gets values of specific pixels

C SPECIFICATION

long readRGB(n, red, green, blue)
short n;
RGBValue red[], green[], blue[];

PARAMETERS

n expects the number of pixels to be read by the function.
red expects the array in which the pixel red values will be stored.
green expects the array in which the pixel green values will be stored.
blue expects the array in which the pixel blue values will be stored.

FUNCTION RETURN VALUE

The returned value of this function is the number of pixels actually read.
A returned function value of 0 indicates an error, namely, that the starting point is not a valid character position.

DESCRIPTION

readRGB attempts to read up to n pixel values from the bitplanes in RGB mode. It reads them into the red, green, and 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 read, which is the number requested if the starting point is a valid character position (inside the current viewport). readRGB returns zero if the starting point is not a valid character position. The values of pixels read outside of the viewport or 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.
In RGB double buffer mode, only the back buffer is read by default. On machines that support it, you can use \texttt{readsource} to control which buffer is read.

The system must be in RGB mode for \texttt{readRGB} to function correctly.

\textbf{SEE ALSO}

\texttt{lrectread}, \texttt{readsource}

\textbf{NOTES}

\texttt{readRGB} should not be used in new development. Rather, pixels should be read using the high-performance \texttt{lrectread} command.

This routine is available only in immediate mode.
NAME

`readsource` — sets the source for pixels that various routines read

C SPECIFICATION

```c
void readsource(src)
long src;
```

PARAMETERS

`src` expects a symbolic constant that identifies the pixel source that is to be used:

- **SRC_AUTO** selects the front color buffer when the current framebuffer, as specified by `drawmode`, is in single buffer mode. It selects the back color buffer when the current framebuffer is in double buffer mode. This is the default.

- **SRC_FRONT** selects the front color buffer of the current framebuffer, as specified by `drawmode`. This source is valid for both single buffer and double buffer operation.

- **SRC_BACK** selects the back color buffer of the current framebuffer, as specified by `drawmode`. This source is valid only while the current framebuffer is in double buffer mode.

- **SRC_ZBUFFER** selects the z-buffer of the current framebuffer. Because only the normal framebuffer has a z-buffer, this source is currently valid only while draw mode is `NORMALDRAW`.

- **SRC_FRAMEGRABBER** selects the Live Video Digitizer as the pixel source, regardless of the current draw mode. This source is valid only on IRIS-4D GTX and VGX models with the Live Video Digitizer option board. IRIS-4D GTX models support this source only during `rectcopy`, not `rectread` or `lrectread`.

- **SRC_OVER** selects the overlay planes, and is valid only while draw mode is `NORMALDRAW`. This source is valid only on the Personal Iris.
SRC_UNDER selects the underlay planes, and is valid only while draw mode is NORMALDRAW. This source is valid only on the Personal Iris.

SRC_PUP selects the pop-up planes, and is valid only while draw mode is NORMALDRAW. This source is valid only on the Personal Iris.

DESCRIPTION

readsource specifies the pixel source buffer that rectcopy, readpixels, readRGB, rectread, and lrectread use. A separate read source is maintained for each of the GL framebuffers: normal, pop-up, overlay, and underlay. Calls to readsource change the read source of the currently active framebuffer, as specified by drawmode. By default the read source for each framebuffer is SRC_AUTO.

Because read sources, with the exception of some implemented only on the Personal Iris, always specify a source within the current framebuffer, it is not possible to copy pixels from one framebuffer to another. Such a copy must be implemented by first reading pixels out of the source framebuffer, then changing the draw mode to the destination framebuffer, and writing the pixels.

SEE ALSO

lrectread, readpixels, readRGB, rectcopy

NOTES

This subroutine is available only in immediate mode.

This subroutine does not function on IRIS-4D B or G models.

Read sources SRC_OVER, SRC_UNDER, and SRC_PUP operate only on the Personal Iris.

BUGS

On the IRIS-4D GT and GTX models, and on the Personal IRIS, a single readsource variable is shared between the four framebuffers. Separate variables will be implemented in the next software release.
On some IRIS-4D GT and GTX models, while copying rectangles with blending active, `readsource` also specifies the bank from which destination color and alpha are read (overriding the `blendfunction` setting).
NAME

rect, recti, rects – outlines a rectangular region

C SPECIFICATION

void rect(x1, y1, x2, y2)
  Coord x1, y1, x2, y2;

void recti(x1, y1, x2, y2)
  Icoord x1, y1, x2, y2;

void rects(x1,y1, x2, y2)
  Scoord x1, y1, x2, y2;

All of the above routines are functionally the same. They differ only in
the type declarations of their parameters.

PARAMETERS

  x1  expects the x coordinate of one of the corners of the rectangle.
  y1  expects the y coordinate of one of the corners of the rectangle.
  x2  expects the x coordinate of the opposite corner of the rectangle.
  y2  expects the y coordinate of the opposite corner of the rectangle.

DESCRIPTION

rect draws an unfilled rectangle in the x-y plane with z assumed to be
zero. The sides of the rectangle are parallel to the x and y axes. To
create a rectangle that does not lie in the x-y plane, draw the rectangle in
the x-y plane, then rotate and/or translate the rectangle.

A rectangle is drawn as a sequence of four line segments, and therefore
inherits all properties that affect the drawing of lines. These include the
current color, writemask, line width, stipple pattern, shade model, line
antialiasing mode, and subpixel mode. The stipple pattern is initialized
to bit zero of the current linestyle before the rectangle is drawn, then
shifted continuously through the segments of the rectangle.
After `rect` executes, the graphics position is undefined.

SEE ALSO

`bgnclosedline`, `linewidth`, `linesmooth`, `lsrepea`, `rectf`, `sbox`, `scrsdivide`, `setlinestyle`, `shademodel`, `subpixel`
NAME

rectcopy – copies a rectangle of pixels with an optional zoom

C SPECIFICATION

void rectcopy(x1, y1, x2, y2, newx, newy)
Screencoord x1,y1, x2, y2, newx, newy;

PARAMETERS

x1  expects the x coordinate of one corner of the rectangle.
y1  expects the y coordinate of one corner of the rectangle.
x2  expects the x coordinate of the opposite corner of the rectangle.
y2  expects the y coordinate of the opposite corner of the rectangle.
newx expects the x coordinate of the lower-left corner of the new position of the rectangle.
newy expects the y coordinate of the lower-left corner of the new position of the rectangle.

DESCRIPTION

rectcopy copies a rectangular array of pixels (x1, y1, x2, y2) to another position on the screen. The current viewport and screenmask mask the drawing of the copied region. Self-intersecting copies work correctly in all cases.

Use readsoure to specify the front buffer, the back buffer, the z-buffer, or the optional Live Video Digitizer frame buffer as the source. When using the Live Video Digitizer as the readsoure, the coordinate arguments should be specified relative to a video origin of (0,0).

On machines that support it, you can use rectzoom to independently zoom the destination in both the x and y directions. Self-intersecting copies with zoom also work correctly. Likewise, on machines that support it, pixmode can be used to greatly affect the copy operation. Pixel shifts, reflections, and numeric offsets are all possible.
Use `frontbuffer`, `backbuffer`, and `zdraw` to specify the destination. All coordinates are relative to the lower-left corner of the window.

The result of `rectcopy` is undefined if `zbuffer` is TRUE, except when pixel mode `PM_ZDATA` is enabled (see `pixmode`). This special pixel mode, in conjunction with stencil operation, can be used to implement rectangle copies with depth buffering.

`rectcopy` always operates within the currently active framebuffer, as specified by `drawmode`.

`rectcopy` leaves the current character position unpredictable.

SEE ALSO

`pixmode`, `readsource`, `rectzoom`, `stencil`

NOTES

This subroutine is available only in immediate mode.

The Live Video Digitizer option is available only on IRIS-4D GTX and VGX models.

BUGS

Pixel format is not considered during the copy. For example, if you copy pixels that contain color index data into an RGB window, the display subsystem cannot correctly interpret it.

On IRIS-4D GTX models the zoom factor is not applied when reading from the Live Video Digitizer’s frame buffer.
NAME
rectf, rectfi, rectfs – fills a rectangular area

C SPECIFICATION

void rectf(x1, y1, x2, y2)
Coord x1, y1, x2, y2;
void rectfi(x1, y1, x2, y2)
Icoord x1, y1, x2, y2;
void rectfs(x1, y1, x2, y2)
Scoord x1, y1, x2, y2;

All of the above routines are functionally the same. They differ only in
the type declarations of their parameters.

PARAMETERS

xl expects the x coordinate of one corner of the rectangle that is to be
drawn.
yl expects the y coordinate of one corner of the rectangle that is to be
drawn.
x2 expects the x coordinate of the opposite corner of the rectangle that
is to be drawn.
y2 expects the y coordinate of the opposite corner of the rectangle that
is to be drawn.

DESCRIPTION

rect draws a filled rectangle in the x-y plane with z assumed to be zero.
The sides of the rectangle are parallel to the x and y axes. To create a
rectangle that does not lie in the x-y plane, draw the rectangle in the x-y
plane, then rotate and/or translate the rectangle.

A rectangle is drawn as a single polygon, and therefore inherits all pro-
properties that affect the drawing of polygons. These include the current
color, writemask, fill pattern, shade model, polygon antialiasing mode,
polygon scan conversion mode, and subpixel mode. Front-face and
back-face elimination work correctly with filled rectangles. The front-
face of a rectangle faces the positive z half-space when (xl, yl) is the
lower-left corner of the rectangle in object coordinates.

After `rectf` executes, the graphics position is undefined.

SEE ALSO

backface, bgnpolygon, frontface, polymode, polysmooth, rect, scrsubdivide, setpattern, shademodel, subpixel

NOTE

Previous graphics library implementations set the current graphics position to \((xI, yI)\) after the rectangle was drawn. Current graphics position is now undefined after a rectangle is drawn.
NAME

rectread, lrectread – reads a rectangular array of pixels into CPU memory

C SPECIFICATION

    long rectread(x1, y1, x2, y2, parray)
    Screencoord x1, y1, x2, y2;
    Colorindex parray[];

    long lrectread(x1, y1, x2, y2, parray)
    Screencoord x1, y1, x2, y2;
    unsigned long parray[];

PARAMETERS

    x1  expects the x coordinate of the lower-left corner of the rectangle
         that you want to read.
    y1  expects the y coordinate of the lower-left corner of the rectangle
         that you want to read.
    x2  expects the x coordinate of the upper-right corner of the rectan-
         gle that you want to read.
    y2  expects the y coordinate of the upper-right corner of the rectan-
         gle that you want to read.
    parray expects the array to receive the pixels that you want to read.

FUNCTION RETURN VALUE

    The returned value of this function is the number of pixels specified in
    the rectangular region, regardless of whether the pixels were actually
    readable (i.e. on-screen) or not.

DESCRIPTION

rectread and lrectread read the pixel values of a rectangular region of
the screen and write them to the array, parray. The system fills the ele-
ments of parray from left-to-right, then bottom-to-top. All coordinates
are relative to the lower-left corner of the window, not the screen or
viewport.
**rectread** fills an array of 16-bit words, and therefore should be used only to read color index values. **lrectread** fills an array of 32-bit words. Based on the current **pixmode**, it can return pixels of 1, 2, 4, 8, 12, 16, 24, or 32 bits each. Use it to read packed RGB or RGBA values, color index values, or z values. Use **readsource** to specify the pixel source from which both **rectread** and **lrectread** read pixels.

**pixmode** greatly affects the operation of **lrectread**, and has no effect on the operation of **rectread**. By default, **lrectread** returns 32-bit pixels in the format used by **cpack**. Different pixel sizes, framebuffer shifts, scan patterns through the framebuffer, and strides through memory, can all be specified using **pixmode**.

**rectread** and **lrectread** leave the current character position unpredictable.

**SEE ALSO**

lrectwrite, pixmode, readsource

**NOTES**

These routines are available only in immediate mode.

On IRIS-4D GT and GTX models, returned bits that do not correspond to valid bitplanes are undefined. Other models return zero in these bits.

On IRIS-4D GT, GTX, and VGX models, **rectread** performance will suffer if \( x_2 - x_1 + 1 \) is odd, or if **parray** is not 32-bit word aligned.
NAME

rectwrite, lrectwrite – draws a rectangular array of pixels into the frame buffer

C SPECIFICATION

void rectwrite(x1, y1, x2, y2, parray)
Screencoord x1, y1, x2, y2;
Colorindex parray[];

void lrectwrite(x1, y1, x2, y2, parray)
Screencoord x1, y1, x2, y2;
unsigned long parray[];

PARAMETERS

x1 expects the lower-left x coordinate of the rectangular region.
y1 expects the lower-left y coordinate of the rectangular region.
x2 expects the upper-right x coordinate of the rectangular region.
y2 expects the upper-right y coordinate of the rectangular region.
parray expects the array which contains the values of the pixels to be drawn. For RGBA values, pack the bits thusly: 0xAAABBGRR, where:

    AA contains the alpha value,
    BB contains the blue value,
    GG contains the green value, and
    RR contains the red value.

RGBA component values range from 0 to 0xFF (255). The alpha value will be ignored if blending is not active and the machine has no alpha bitplanes.

DESCRIPTION

rectwrite and lrectwrite draw pixels taken from the array parray into the specified rectangular frame buffer region. The system draws pixels left-to-right, then bottom-to-top. All coordinates are relative to the lower-left corner of the window, not the screen or viewport. All normal drawing modes apply.
The size of \texttt{parray} is always \((x2-x1+1) \times (y2-y1+1)\). If the zoom factors set by \texttt{rectzoom} are both 1.0, the screen region \(x1\) through \(x2\), \(y1\) through \(y2\), are filled. Other zoom factors result in filling past \(x2\) and/or past \(y2\) (\(x1,y1\) is always the lower-left corner of the filled region).

\texttt{rectwrite} draws an array of 16-bit words, and therefore should be used only to write color index values. \texttt{lrectwrite} draws an array of 32-bit words. Based on the current \texttt{pixmode}, in can draw pixels of 1, 2, 4, 8, 12, 16, 24, or 32 bits each. Use it to write packed RGB or RGBA values, color index values, or \(z\) values.

\texttt{pixmode} greatly affects the operation of \texttt{lrectwrite}, and has no effect on the operation of \texttt{rectwrite}. By default, \texttt{lrectwrite} draws 32-bit pixels in the format used by \texttt{cpack}. Different pixel sizes, framebuffer shifts, scan patterns through the framebuffer, and strides through memory, can all be specified using \texttt{pixmode}.

\texttt{rectwrite} and \texttt{lrectwrite} leave the current character position unpredictable.

\textbf{SEE ALSO}

\texttt{blendfunction}, \texttt{lrectread}, \texttt{pixmode}, \texttt{rectcopy}, \texttt{rectzoom}

\textbf{NOTES}

These routines are available only in immediate mode.
NAME

`rectzoom` – specifies the zoom for rectangular pixel copies and writes

C SPECIFICATION

```c
void rectzoom(xfactor, yfactor)
float xfactor, yfactor;
```

PARAMETERS

`xfactor` expects the multiplier of the rectangle in the x direction.
`yfactor` expects the multiplier of the rectangle in the y direction.

DESCRIPTION

`rectzoom` specifies independent x and y zoom factors that `rectcopy`, `rectwrite`, and `lrectwrite` use. `rectzoom` scales the source image by the numbers specified by `xfactor` and `yfactor`. If `rectzoom(2.0, 3.0)` is called, and the following rectangle is copied:

```
  1 3
  5 7
```

the copy will be:

```
  1 1 3 3
  1 1 3 3
  1 1 3 3
  5 5 7 7
  5 5 7 7
  5 5 7 7
```

Although zoom factors are specified as floating point values, some graphics systems do not support fractional zooms. These systems round each floating point zoom factor to the nearest integer value. Systems that do support fractional zoom replicate source image pixels when the zoom factor is greater than 1.0, and decimate the source image when the zoom factor is less than 1.0.
By default, xfactor and yfactor are 1.0.

SEE ALSO
lirectwrite, pixmode, rectcopy, rectwrite

NOTE
This subroutine is available only in immediate mode.
IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support fractional zoom.
NAME
resetls – controls the continuity of linestyles

C SPECIFICATION
void resetls(b)
Boolean b;

PARAMETERS

b expects either TRUE or FALSE.

TRUE causes the linestyle to be reset at the beginning of each line segment.
FALSE causes the linestyle to be continued across the segments of a line.

DESCRIPTION
resetls enables or disable linestyle reset mode. This mode affects the reinitialization of the linestyle pattern between line segments. If it is enabled (the default), it causes the stippling of each segment of a line to start at the beginning of the linestyle pattern. If it is disabled, 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.

Changing resetls from FALSE to TRUE in the middle of a line causes the linestyle to be reset. If resetls is FALSE when setlinestyle is called, the linestyle does not change until resetls(TRUE) is issued. resetls(TRUE) also initializes the lsrepeat factor to 1.

SEE ALSO
deflinestyle, getresetls, lsrepeat, setlinestyle

NOTES

The setting of resetls is ignored for Graphics Library primitives such as arcs, circles, and curves, even though they are currently implemented using lines.
This routine only functions on IRIS-4D B and G models, and therefore we advise against its use in new development.
NAME

**reshapeviewport** – sets the viewport to the dimensions of the current graphics window

C SPECIFICATION

```c
void reshapeviewport()
```

PARAMETERS

*none*

DESCRIPTION

**reshapeviewport** sets the viewport to the dimensions of the current graphics window. Call it whenever REDRAW events are received for windows whose size is unconstrained, and therefore could have changed.

**reshapeviewport** is equivalent to:

```c
long xsize, ysize;
getsize(&xsize, &ysize);
viewport(0, xsize-1, 0, ysize-1);
```

SEE ALSO

getorigin, getsize, viewport

NOTE

This routine is available only in immediate mode.
NAME

RGBcolor – sets the current color in RGB mode

C SPECIFICATION

void RGBcolor(red, green, blue)
short red, green, blue;

PARAMETERS

red expects the value indicating the intensity of the red component.
green expects the value indicating the intensity of the green component.
blue expects the value indicating the intensity of the blue component.

DESCRIPTION

RGBcolor sets the red, green, and blue color components of the currently active GL framebuffer, one of normal, popup, overlay, or underlay as specified by drawmode, to the specified values. Alpha, when supported, is set to the maximum value. The current framebuffer must be in RGB mode for the RGBcolor command to be applicable. Most drawing commands copy the current RGBA color components into the color bitplanes of the current framebuffer. Color components are retained in each draw mode, so when a draw mode is re-entered, red, green, blue, and alpha are reset to the last values specified in that draw mode.

Color component values range from 0, specifying no intensity, through 255, specifying maximum intensity. Values that exceed 255 are clamped to it. Values less than 0 are not clamped, and therefore result in unpredictable operation.

It is an error to call RGBcolor while the current framebuffer is in color map mode.

The color components of all framebuffers in RGB mode are set to zero when gconfig is called.
SEE ALSO

c, cpack, drawmode, gRGBcolor, lmcolor, RGBmode

NOTE

RGBcolor can also be used to modify the current material while lighting is active (see lmcolor).
NAME

RGBcursor – obsolete routine

C SPECIFICATION

void RGBcursor(index, red, green, blue, redm, greenm, bluem)
short index, red, green, blue, redm, greenm, bluem;

DESCRIPTION

This routine is obsolete. It continues to function only on IRIS-4D B and
G models to provide backwards compatibility. All new development
should use its replacement, setcursor.

SEE ALSO

setcursor

NOTE

This routine is available only in immediate mode.
NAME

RGBmode – sets a rendering and display mode that bypasses the color map

C SPECIFICATION

void RGBmode()

PARAMETERS

none

DESCRIPTION

RGBmode instructs the system to treat color as a 4-component entity in the currently active drawmode. Currently RGB mode is supported only by the normal framebuffer, so RGBmode should be called only while in draw mode NORMALDRAW. You must call gconfig for RGBmode to take effect.

While in RGB mode, a framebuffer is configured to store separate color values for red, green, blue, and alpha components. Lighting, shading, and fog calculations are done in the true, RGB color space. Colors and writemasks must be specified using RGB-compatible commands such as c, cpack, and wmpack. The red, green, and blue components stored in the framebuffer are used (after correction for monitor non-linearity) to directly control the color guns of the monitor.

Many advanced rendering features, such as complex lighting models, texture mapping, polygon antialiasing, and fog, are available only in RGB mode.

SEE ALSO

c, cmode, cpack, gammaramp, gconfig, getdisplaymode, getgdesc, wmpack

NOTE

RGBmode is not supported on all hardware configurations. The command getgdesc can be used to determine how many bitplanes in the normal framebuffer are available for each color component in both single
and double buffered RGB mode.
This routine is available only in immediate mode.
NAME

RGBrange – obsolete routine

C SPECIFICATION

void RGBrange(rmin, gmin, bmin, rmax, gmax, bmax, z1, z2)
short rmin, gmin, bmin, rmax, gmax, bmax;
Screencoord z1, z2;

PARAMETERS

rmin expects the minimum value to be stored in the red bitplanes.
gmin expects the minimum value to be stored in the green bitplanes.
bmin expects the minimum value to be stored in the blue bitplanes.
rmax expects the maximum value to be stored in the red bitplanes.
gmax expects the maximum value to be stored in the green bitplanes.
bmax expects the maximum value to be stored in the blue bitplanes.
z1 expects the minimum z value that is to be used as criteria for linear mapping.
z2 expects the maximum z value that is to be used as criteria for linear mapping.

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use its replacement, IRGBrange.

SEE ALSO

IRGBrange
NAME

RGBwritemask – grants write access to a subset of available bitplanes

C SPECIFICATION

void RGBwritemask(red, green, blue)
short red, green, blue;

PARAMETERS

red   expects the mask for the corresponding red bitplanes.
green expects the mask for the corresponding green bitplanes.
blue  expects the mask for the corresponding blue bitplanes.

DESCRIPTION

RGBwritemask sets the red, green, and blue write mask components of
the currently active GL framebuffer, one of normal, popup, overlay, or
underlay as specified by drawmode, to the specified values. The alpha
mask component is set to enable writing to all alpha bitplanes. The
current framebuffer must be in RGB mode for the RGBwritemask com-
mand to be applicable. All drawing into the color bitplanes of the
current framebuffer is masked by the current write mask. Write mask
components are retained in each draw mode, so when a draw mode is
re-entered, the red, green, blue, and alpha masks are reset to the last
values specified in that draw mode.

Each write mask component is an 8-bit mask, which allows changes
only to bitplanes corresponding to ones in the mask. For example,
RGBwritemask(0xF0,0x00,0x00) allows changes only to the 4 most
significant bits of red, and to all the bits of alpha. Bits 8 through 15 of
each component specification are ignored, only bits 0 through 7 are
significant.

It is an error to call RGBwritemask while the current framebuffer is in
color map mode.

The write mask components of all framebuffers in RGB mode are set to
0xFF when gconfig is called.
SEE ALSO

drawmode, gRGBmask, RGBmode, wmpack
NAME
  ringbell – rings the keyboard bell

C SPECIFICATION
  void ringbell()

PARAMETERS
  none

DESCRIPTION
  ringbell rings the keyboard bell for the length of time set earlier by setbell.

SEE ALSO
  clkon, lampon, setbell

NOTE
  This routine is available only in immediate mode.
NAME

rmv, rmvi, rmvs, rmv2, rmv2i, rmv2s – relative move

C SPECIFICATION

void rmv(dx, dy, dz)
Coord dx, dy, dz;

void rmvi(dx, dy, dz)
Icoord dx, dy, dz;

void rmvs(dx, dy, dz)
Scoord dx, dy, dz;

void rmv2(dx, dy)
Coord dx, dy;

void rmv2i(dx, dy)
Icoord dx, dy;

void rmv2s(dx, dy)
Scoord dx, dy;

All of the above routines are functionally the same. They differ only in the type declarations of their parameters and in whether or not they assume a two- or three-dimensional space.

PARAMETERS

dx expects the distance from the x coordinate of the current graphics position to the x coordinate of the new graphics position.

dy expects the distance from the y coordinate of the current graphics position to the y coordinate of the new graphics position.

dz expects the distance from the z coordinate of the current graphics position to the z coordinate of the new graphics position.

DESCRIPTION

rmv is the relative version of move. It moves (without drawing) the graphics position the specified amount relative to its current value. rmv2(x, y) is equivalent to rmv(x, y, 0.0).
SEE ALSO
bgnline, endline, popmatrix, pushmatrix, rdr, translate, v

NOTE
rmv should not be used in new development. Rather, lines should be
drawn using the high-performance v commands, surrounded by calls to
bgnline and endline. Matrix commands pushmatrix, translate, and
popmatrix should be used to accomplish relative positioning.
NAME

rotate, rot — rotate graphical primitives

C SPECIFICATION

void rotate(a, axis)
Angle a;
char axis;

void rot(a, axis)
float a;
char axis;

PARAMETERS

a expects the angle of rotation of an object.
axis expects the relative axis of rotation. There are three character
literal values for this parameter:
'x' indicates the x-axis.
'y' indicates the y-axis.
'z' indicates the z-axis.

DESCRIPTION

rotate and rot specify an angle (a) and an axis of rotation (axis). The
angle given to rotate is an integer and is specified in tenths of degrees
according to the right-hand rule. The angle given to rot is a floating
point value and is specified in degrees according to the right-hand rule.

These are modeling routines; they changes the current transformation
matrix. All objects drawn after rotate or rot are rotated. Use
pushmatrix and popmatrix to preserve and restore an unrotated world
space.

SEE ALSO

popmatrix, pushmatrix, scale, translate
NAME

rpitch – draws a rational surface patch

C SPECIFICATION

void rpatch(geomx, geomy, geomz, geomw)
Matrix geomx, geomy, geomz, geomw;

PARAMETERS

geomx expects a 4x4 matrix containing the x coordinates of the 16 control points of the patch.

geomy expects a 4x4 matrix containing the y coordinates of the 16 control points of the patch.

geomz expects a 4x4 matrix containing the z coordinates of the 16 control points of the patch.

geomw expects a 4x4 matrix containing the w coordinates of the 16 control points of the patch.

DESCRIPTION

rpitch draws a rational surface patch using the current patchbasis, patchprecision, and patchcurves which are defined earlier. The control points geomx, geomy, geomz geomw determine the shape of the patch.

The patch is drawn as a web of curve segments. Each curve segment is approximated by a sequence of straight lines. All lines use the current linestyle, which is reset prior to the first line of each curve segment, and continues through subsequent lines in each curve segment. Other line modes, including depthcuing, line width, and line antialiasing, also apply to the lines generated by patch.

SEE ALSO

defbasis, patch, patchbasis, patchcurves, patchprecision
NAME

rpdr, rpdi, rpdrs, rpdr2, rpdr2i, rpdr2s – relative polygon draw

C SPECIFICATION

void rpdr(dx, dy, dz)
Coord dx, dy, dz;

void rpdi(dx, dy, dz)
Icoord dx, dy, dz;

void rpdrs(dx, dy, dz)
Scoord dx, dy, dz;

void rpdr2(dx, dy)
Coord dx, dy;

void rpdr2i(dx, dy)
Icoord dx, dy;

void rpdr2s(dx, dy)
Scoord dx, dy;

All of the above routines are functionally the same. They differ only in the type declarations for their parameters and in whether they assume a two- or three-dimensional space.

PARAMETERS

dx expects the distance from the x coordinate of the current graphics position to the x coordinate of the next corner of the polygon.

dy expects the distance from the y coordinate of the current graphics position to the y coordinate of the next corner of the polygon.

dz expects the distance from the z coordinate of the current graphics position to the z coordinate of the next corner of the polygon.

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. The next drawing routine will start drawing from that point.
SEE ALSO

bgnpolygon, endpolygon, pclos, rpmv, v

NOTES

rpdr should not be used in new development. Rather, polygons should be drawn using the high-performance v commands, surrounded by calls to bgnpolygon and endpolygon. Matrix commands pushmatrix, translate, and popmatrix should be used to accomplish relative positioning.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 rpdr calls between rpmv and pclos.
NAME

rpmv, rpmvi, rpmvs, rpmv2, rpmv2i, rpmv2s – relative polygon move

C SPECIFICATION

void rpmv(dx, dy, dz)
 Coord dx, dy, dz;

void rpmvi(dx, dy, dz)
 Icoord dx, dy, dz;

void rpmvs(dx, dy, dz)
 Scoord dx, dy, dz;

void rpmv2(dx, dy)
 Coord dx, dy;

void rpmv2i(dx, dy)
 Icoord dx, dy;

void rpmv2s(dx, dy)
 Scoord dx, dy;

All of the above routines are functionally the same. They differ only in the type declarations of their parameters and in whether they assume a two-dimensional or three-dimensional space.

PARAMETERS

dx expects the distance from the x coordinate of the current graphics position to the x coordinate of the first point in a polygon.

dy expects the distance from the y coordinate of the current graphics position to the y coordinate of the first point in a polygon.

dz expects the distance from the z coordinate of the current graphics position to the z coordinate of the first point in a polygon.

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.
Between \texttt{rpmv} and \texttt{pclos}, you can issue only the following Graphics Library subroutines: \texttt{color}, \texttt{RGBcolor}, \texttt{c}, \texttt{cpack}, \texttt{n}, \texttt{v}, \texttt{lmdef}, and \texttt{lmbind}. Use \texttt{lmdef} and \texttt{lmbind} to respecify only materials and their properties.

\textbf{SEE ALSO}

\texttt{bgnpolygon}, \texttt{endpolygon}, \texttt{pclos}, \texttt{rpdr}, \texttt{v}

\textbf{NOTES}

\texttt{rpmv} should not be used in new development. Rather, polygons should be drawn using the high-performance \texttt{v} commands, surrounded by calls to \texttt{bgnpolygon} and \texttt{endpolygon}. Matrix commands \texttt{pushmatrix}, \texttt{translate}, and \texttt{popmatrix} should be used to accomplish relative positioning.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 \texttt{rpdr} calls between \texttt{rpmv} and \texttt{pclos}. 
NAME

sbox, sboxi, sboxs — draw a screen-aligned rectangle

C SPECIFICATION

```c
void sbox(x1, y1, x2, y2)
Coord x1, y1, x2, y2;

void sboxi(x1, y1, x2, y2)
Icoord x1, y1, x2, y2;

void sboxs(x1, y1, x2, y2)
Scoord x1, y1, x2, y2;
```

All of the above functions are functionally the same except for the type declarations of the parameters.

PARAMETERS

- `x1` expects the x coordinate of a corner of the box.
- `y1` expects the y coordinate of a corner of the box.
- `x2` expects the x coordinate of the opposite corner of the box.
- `y2` expects the y coordinate of the opposite corner of the box.

DESCRIPTION

sbox draws an unfilled, two-dimensional, screen-aligned rectangle. The rectangle is drawn as a sequence of four line segments, and therefore inherits many properties that affect the drawing of lines. These include the current color, writemask, line width, stipple pattern, and subpixel mode. The stipple pattern is initialized to bit zero of the current linestyle before the rectangle is drawn, then shifted continuously through the segments of the rectangle.

The sides of the rectangle will be parallel to the screen x and y axes. This rectangle cannot be rotated. The z coordinate is set to zero.

When you use sbox, you must not use alpha blending, backfacing or frontfacing, depthcueing, fog, gouraud shading, lighting, line antialiasing, screen subdivision, stenciling, texture mapping, or z-buffering.
sbox may be faster than rect on some machines. Use sbox when you need to draw a large number of screen-aligned rectangles.

After sbox executes, the graphics position is undefined.

SEE ALSO

backface, bgnclosedline, blendfunction, deflinestyle, depthcue, linewidth, linesmooth, lmbind, lsmrepeat, rect, scrsnddivide, setlinestyle, shademodel, stencil, subpixel, texbind, zbuffer
NAME

sboxf, sboxfi, sboxfs – draw a filled screen-aligned rectangle

C SPECIFICATION

void sboxf(x1, y1, x2, y2)
Coord x1, y1, x2, y2;

void sboxfi(x1, y1, x2, y2)
Icoord x1, y1, x2, y2;

void sboxfs(x1, y1, x2, y2)
Scoord x1, y1, x2, y2;

All of the above functions are functionally the same except for the type declarations of the parameters.

PARAMETERS

x1 expects the x coordinate of a corner of the filled box.
y1 expects the y coordinate of a corner of the filled box.
x2 expects the x coordinate of the opposite corner of the filled box.
y2 expects the y coordinate of the opposite corner of the filled box.

DESCRIPTION

sboxf draws a filled, two-dimensional, screen-aligned rectangle. The rectangle is drawn as a single polygon, and therefore inherits many properties that affect the drawing of polygons. These include the current color, writemask, fill pattern, and subpixel mode.

The sides of the rectangle will be parallel to the screen x and y axes. This rectangle cannot be rotated. The z coordinate is set to zero.

When you use sboxf, you must not use alpha blending, backfacing or frontfacing, depthcueing, fog, gouraud shading, lighting, polygon antialiasing, screen subdivision, stenciling, texture mapping, or z-buffering. polymode must be set to PYM_FILLED.
sboxf may be faster than rectf on some machines. Use sboxf when you need to draw a large number of screen-aligned rectangles.

After sboxf executes, the graphics position is undefined.

SEE ALSO

backface, bgnpolygon, blendfunction, depthcue, frontface, Imbind, polymode, polynomials, rectf, scrsubdivide, setpattern, shademodel, stencil, subpixel, texbind, zbuffer
NAME

scale – scales and mirrors objects

C SPECIFICATION

void scale(x, y, z)
float x, y, z;

PARAMETERS

x  expects the scaling of the object in the x direction.
y  expects the scaling of the object in the y direction.
z  expects the scaling of the object in the z direction.

DESCRIPTION

scale shrinks, expands, and mirrors objects. Values with a magnitude greater than 1 expand the object; values with a magnitude less than 1 shrink it. Negative values mirror the object. Mirroring places the reflection of an object in the area defined. The original is no longer displayed.

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, rotate, translate
NAME

sclear – clear the stencil planes to a specified value

C SPECIFICATION

void sclear(sval)
unsigned long sval;

PARAMETERS

sval expects the integer value that is to be written to every stencil location

DESCRIPTION

sclear sets every pixel in the currently allocated stencil buffer (see stensize) to sval, the passed parameter. This clearing operation is limited by the current viewport and scrmask, and is masked by the current swritemask. Current color is unchanged. The polygon pattern, if any, is ignored.

SEE ALSO

stensize, swritemask

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support stencil, and therefore also do not support sclear. Use getgdesc to determine whether stencil is supported.

Because only the normal framebuffer includes stencil resources, sclear should be called only while draw mode is NORMALDRAW.
NAME

scrbox – control the screen box

C SPECIFICATION

void scrbox(argc)
    long argc;

PARAMETERS

arg  Expects one of the symbolic constants:

SB_RESET: initialize screen box limits. (default)

SB_TRACK: track scan-converted geometry and characters and update the scrbox limits accordingly.

SB_HOLD: disable update of screen box limits; hold current values.

DESCRIPTION

scrbox is a dual of the scrmask capability. Rather than limiting drawing effects to a screen-aligned subregion of the viewport, it tracks the screen-aligned subregion (screen box) that has been affected. Unlike scrmask, which defaults to the viewport boundry if not explicitly enabled, scrbox must be explicitly turned on to be effective.

While enabled (mode SB_TRACK) scrbox maintains leftmost, rightmost, lowest, and highest window coordinates of all pixels that are scan converted. Because scrbox operates on the pixels that result from the scan conversion of points, lines, polygons, and characters; it correctly handles wide lines, antialiased (smooth) points and lines, and characters. Because scrbox operation may precede the framebuffer, scan-converted pixels may update the screen box regardless of their Z compare, WID compare, or stencil compare results.

scrbox results are guaranteed to bound the modified framebuffer region, but they may exceed the bounds of this region.
When reset, the leftmost and lowest screen box values are set to be greater than the rightmost and highest values.

SEE ALSO

getscrbox, scrmask

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support scrbox. Use getgdesc to determine whether scrbox is supported.
NAME

screenspace – map world space to absolute screen coordinates

C SPECIFICATION

void screenspace()

PARAMETERS

none

DESCRIPTION

screenspace sets the projection matrix and viewport of the current window so as to map world space to absolute screen coordinates (instead of to the more usual window-relative screen coordinates). This provides a convenient coordinate system for operations that are not constrained to a window, e.g. reading pixels.

global

screenspace is equivalent to:

long xmin, ymin;
getorigin(&xmin, &ymin);
viewport(-xmin, getgdesc(GD_XPAX) - xmin,
- -ymin, getgdesc(GD_YPAX) - ymin);
ortho2(-0.5, getgdesc(GD_XPAX) + 0.5,
- -0.5, getgdesc(GD_YPAX) + 0.5);

SEE ALSO

fullscrn, getgdesc, getorigin, viewport, ortho2

NOTE

This routine is available only in immediate mode.
NAME

scrmask – defines a rectangular screen clipping mask

C SPECIFICATION

void scrmask(left, right, bottom, top)
Screencoord left, right, bottom, top;

PARAMETERS

left expects the window coordinate of the left-most pixel column within the mask region.

right expects the window coordinate of the right-most pixel column within the mask region.

bottom expects the window coordinate of the lowest pixel row within the mask region.

top expects the window coordinate of the highest pixel row within the mask region.

DESCRIPTION

scrmask defines a subregion of the current viewport that can be updated by drawing commands. Pixels outside this region cannot be modified by any drawing commands, including point, line, polygon, character, pixel write, and pixel copy commands. All pixel bitplane buffers, including color, depth, accumulation, and stencil buffers, are write protected. scrmask operates in all draw modes.

The enabled subregion is specified as a screen-aligned rectangle in window coordinates. Like viewport, the boundary specification is inclusive, so the call scrmask(0,0,0,0) specifies a 1-pixel rectangle in the lower-left corner of the window.

When viewport is called, the screen mask is set to match the newly specified viewport. Any previously scrmask specification is lost.

scrmask must be specified entirely within the current viewport.
SEE ALSO
drawmode, getscrmask, viewport

NOTE
If you set left to be greater than right or bottom to be greater than top, all pixels in the viewport are write protected.
NAME

scrnattach — attaches the input focus to a screen

C SPECIFICATION

long scrnattach(gsnr)
long gsnr;

PARAMETERS

gsnr expects a screen number or the symbolic constant, INFOCUSSCRN.

FUNCTION RETURN VALUE

The function returns the screen that previously had input focus, or -1 if there was an error (e.g., gsnr is not a valid screen number).

DESCRIPTION

scrnattach attaches the input focus to the specified screen. It waits for any window manager or menu interaction to be completed before doing the attach. Calling scrnattach with the argument INFOCUSSCRN simply returns the screen that currently has the input focus.

On error, scrnattach leaves the input focus unchanged.

SEE ALSO

getgdesc, getwscrn, scrnselect

NOTES

This routine is available only in immediate mode.

Use getgdesc(GD_NSCRNS) to determine the number of screens available to your program. Screens are numbered starting from zero.
NAME

`scrnselect` - selects the screen upon which new windows are placed

C SPECIFICATION

```c
long scrnselect(gsnr)
long gsnr;
```

PARAMETERS

`gsnr` expects a screen number or the symbolic constant, INFOCUSSCRN, to select the screen with the input focus at the time the routine is called.

FUNCTION RETURN VALUE

The function returns the previously selected screen, or −1 if there was an error (e.g., `gsnr` is not a valid screen number).

DESCRIPTION

`scrnselect` selects the screen upon which a subsequent `winopen`, `ginit`, `gbegin`, creates a window. It also selects the screen to which `getgdesc` and `gversion` inquiries refer. The default is the screen with the input focus at the time the first Graphics Library routine is called.

On error, `scrnselect` leaves the currently selected screen unchanged.

You can call `scrnselect` prior to graphics initialization.

SEE ALSO

`getgdesc`, `ginit`, `gversion`, `scrmattach`, `winopen`

NOTES

This routine is available only in immediate mode.

Use `getgdesc(GD_NSCRNS)` to determine the number of screens available to your program. Screens are numbered starting from zero.
NAME

scrssubdivide – subdivide lines and polygons to a screen-space limit

C SPECIFICATION

void scrssubdivide(mode, params)
long mode;
float params[];

PARAMETERS

mode Specify whether and how lines and polygons are to be subdivided. Options are:

SS_OFF: do not subdivide. (default)

SS_DEPTH: subdivide based on z values in screen-coordinates.

params Expects an array that contains parameter specifications for the subdivision mode that has been selected.

SS_OFF expects no values in the params array.

SS_DEPTH expects three values in the params array: maxz, minsize, and maxsize. maxz specifies the distance, in screen-coordinates, between \( z = \text{constant} \) subdivision planes. (Z-buffer screen coordinates are defined by \texttt{isetdepth}.) minsize and maxsize specify bounds, in units of pixels, of the screen size of the resulting subdivided polygons. Setting maxz to 0.0 eliminates screen-coordinate \( z \) from consideration during the subdivision. Likewise, setting minsize or maxsize to 0.0 eliminates lower or upper bounds on screen size from consideration.

DESCRIPTION

When scrssubdivide mode is not SS_OFF, lines and polygons are subdivided until the specified criteria are met. Parameters are assigned to created vertices as though they have been interpolated in eye-coordinates, rather than in screen-coordinates. Thus effects that result from (incorrect) linear interpolation in screen-coordinates can be compensated for with scrssubdivide.
Mode SS_DEPTH slices polygons into strips whose edges have constant screen z value. It divides lines into segments whose endpoint z values differ by maxz. This subdivision is done after lighting, so the newly created vertices are not lighted, but rather simply take color values as linear interpolants of the original vertices (in eye-coordinates). Both fog and texture mapping are done after the depth subdivision, so both benefit from its operation.

Polygon slices created by SS_DEPTH subdivision have edges whose z values differ by maxz. However, if the width of the resulting slices is less than minsize, the slices are increased to have width equal to minsize. For example, if maxsize is set to 0.0 (i.e. defeated), a polygon that directly faces the viewer is not subdivided, because all vertices have the same z value. As this polygon is rotated away from the viewer, it is sliced into strips whose edges are parallel to the axis of rotation. The number of strips increases as the rotation increases, until the strips reach a width (measured perpendicular to the axis of rotation) of minsize. At this angle the number of slices is at its maximum. As the rotation is continued, the slice width remains constant, and the number of slices decreases, reaching zero as the polygon becomes perpendicular to the viewer.

When maxsize is non-zero, the description above changes only in that large polygons that are nearly perpendicular to the viewer are subdivided into strips of width maxsize. Likewise, lines segments created by SS_DEPTH subdivision are limited to a minimum length of minsize, and a maximum length of maxsize.

SS_DEPTH subdivision improves the accuracy of texture mapping when non-orthographic projections are used, and improves the accuracy of fog calculations. It is not useful for lighting improvement.

SEE ALSO
fogvertex, texbind, tevbind

NOTE
scrsdivide cannot be used while mmode is MSINGLE.
IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support scrsdivide.

BUGS

When the screen size of subdivided polygons is limited, either by minsize or by maxsize, adjacent polygons can subdivide differently such that newly created vertices on their shared boundary do not coincide. In this case, some pixels at their shared boundary may not be scan converted by either polygon.

Incorrect specification of either maxz or minsize can result in near-infinite polygon subdivision. To avoid the resulting poor graphics system response, IRIS-4D VGX models do not subdivide polygons whose SS_DEPTH subdivision would result in more than 2000 slices.
NAME

setbell – sets the duration of the beep of the keyboard bell

C SPECIFICATION

void setbell(durat)
Byte durat;

PARAMETERS

durat expects a value indicating the length of time the keyboard bell
will sound.

  0 no beep.
  1 short beep.
  2 long beep.

DESCRIPTION

setbell sets the duration of the beep of the keyboard bell. The keyboard
bell is activated by ringbell.

SEE ALSO

eclkon, lampon, ringbell eclkon, lampon, ringbe

NOTE

This routine is available only in immediate mode.
NAME

setcursor – sets the cursor characteristics

C SPECIFICATION

void setcursor(index, color, wtm)
short index;
Colorindex color, wtm;

PARAMETERS

index  expects an index into the predefined definition table.

color  argument ignored.

wtm   argument ignored.

DESCRIPTION

setcursor selects a cursor glyph from among those defined with defcursor. color and wtm are ignored by this routine. To set the color for the cursor use mapcolor and drawmode.

SEE ALSO

attachcursor, curstype, defcursor, curorigin, drawmode, getcursor, mapcolor, RGBcursor

NOTE

This routine is available only in immediate mode.
NAME

setdblights – sets the lights on the dial and button box

C SPECIFICATION

void setdblights(mask)
unsigned long mask;

PARAMETERS

mask expects 32 packed bits indicating which lights you want turned on.

DESCRIPTION

setdblights turns on a combination of the lights on the dial and switch box. A dial and switch box is an I/O device which has thirty-two lighted switches on it. 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

NOTE

This routine is available only in immediate mode.
NAME

setdepth – obsolete routine

C SPECIFICATION

void setdepth(near, far)
Screencoord near, far;

PARAMETERS

near expects the screen coordinate of the near clipping plane.

far expects the screen coordinate of the far clipping plane.

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use its replacement, lsetdepth.

SEE ALSO

lsetdepth
NAME

setlinestyle — selects a linestyle pattern

C SPECIFICATION

void setlinestyle(index)
short index;

PARAMETERS

index expects an index into the linestyle table.

DESCRIPTION

setlinestyle selects a linestyle pattern from a linestyle table defined 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
NAME

setmap – selects one of the small color maps provided by multimap mode

C SPECIFICATION

void setmap(mapnum)
short mapnum;

PARAMETERS

mapnum expects the number of the small color map to be used.

DESCRIPTION

setmap selects one of the small color maps provided by multimap mode. There are getgdesc(GD_NMMAPS) maps, whose numbering starts from 0. setmap can only be used in multimap mode; it is ignored in onemap mode.

SEE ALSO

getgdesc, getmap, multimap, onemap

NOTE

This routine is available only in immediate mode.
NAME

setmonitor — sets the monitor type

C SPECIFICATION

void setmonitor(mtype)
short mtype;

PARAMETERS

mtype expects a symbolic constant that identifies the monitor mode to be used. There are five constants defined for this parameter:

HZ30 selects 30Hz interlaced monitor mode.

HZ30_SG selects 30HZ noninterlaced with sync on green monitor mode.

HZ60 selects 60Hz noninterlaced monitor mode.

NTSC selects NTSC monitor mode.

PAL selects PAL or SECAM monitor mode.

STR_RECT puts your monitor in stereo viewing mode if it has that option. (Otherwise this is ignored.)

DESCRIPTION

setmonitor sets the monitor to 30Hz interlaced, 60Hz noninterlaced, 30Hz interlaced with sync on green, NTSC, or PAL, depending on whether mtype is HZ30, HZ60, HZ30_SG, NTSC, or PAL, respectively.

SEE ALSO

getmonitor, getothermonitor, setvideo

NOTES

This routine is available only in immediate mode.

The symbolic values for mtype mentioned above are defined in <gl/get.h>.
BUGS

IRIS-4D VGX models may hang the graphics pipe, resulting in failure of all running programs including the window manager, when setmonitor is called while other graphics processes are running.
NAME

setnurbsproperty – sets a property for the display of trimmed NURBS surfaces

C SPECIFICATION

    void setnurbsproperty(property, value)
    long property;
    float value;

PARAMETERS

property  expects the name of the property to be set.

value    expects the value to which the named property will be set.

DESCRIPTION

The display of NURBS surfaces can be controlled in different ways. The following is a list of the display properties that can be affected.

N_ERRORCHECKING:  If value is 1.0, some error checking is enabled. If error checking is disabled, the system runs slightly faster. The default value is 0.0.

N_PIXEL_TOLERANCE:  The value is the maximum length, in pixels, of edges of polygons on the screen used to render trimmed NURBS surfaces. The default value is 50.0 pixels.

SEE ALSO

bgnsurface, nurbssurface, bgntrim, nurbscurve, pwlcurve,
getnurbsproperty
NAME

setpattern – selects a pattern for filling polygons and rectangles

C SPECIFICATION

void setpattern(index)
short index;

PARAMETERS

index expects the index into the table of defined patterns.

DESCRIPTION

setpattern selects a pattern from a table of patterns previously defined by defpattern. 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
NAME

setup – sets the display characteristics of a given pop up menu entry

C SPECIFICATION

void setup(pup, entry, mode)
long pup, entry;
unsigned long mode;

PARAMETERS

pup expects the menu identifier of the menu whose entries you want to change. The menu identifier is the returned function value of the menu creation call to either newpup or defpup.

entry expects the position of the entry in the menu, indexed from 1.

mode expects a symbolic constant that indicates the display characteristics you want to apply to the chosen entry. For this parameter there are two defined symbolic constants:

PUP_NONE, no special display characteristics, fully functional if selected. This is the default mode for newly created menu entries.

PUP_GREY, entry is greyed-out and disabled. Selecting a greyed-out entry has the same behavior as selecting the title bar. If the greyed-out entry has a submenu associated with it, that submenu does not display.

DESCRIPTION

Use setup to alter the display characteristics of a pop up menu entry. Currently, you use this routine to disable and grey-out a menu entry.

EXAMPLE

Here is an example that disables a single entry:

menu = newpup();
addtopup(menu,"menu %t |item 1 |item 2 |item 3 |item 4",0);
setup(menu, 1, PUP_GREY);
Subsequent calls of \texttt{dopup(menu)} would display the menu with the menu entry labeled "item 1" is greyed out, and never gets a return value of 1.

\textbf{SEE ALSO}

defpup, dopup, freepup, newpup

\textbf{NOTE}

This routine is available only in immediate mode.
NAME

setshade – obsolete routine

C SPECIFICATION

void setshade(shade)
Colorindex shade;

PARAMETERS

none

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use its identical replacement, color.

SEE ALSO

color
NAME

setvaluator — assigns an initial value and a range to a valuator

C SPECIFICATION

void setvaluator(v, init, vmin, vmax)
Device v;
short init, vmin, vmax;

PARAMETERS

v expects the device number for the valuator being set.
init expects the initial value to be assigned to the valuator.
vmin expects the minimum value that the device can assume.
vmax expects the maximum value that the device can assume.

DESCRIPTION

setvaluator sets the initial value and the minimum and maximum values the specified 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

NOTE

This routine is available only in immediate mode.
NAME

setvideo, getvideo — set and get video hardware registers

C SPECIFICATION

void setvideo(reg, value)
long reg, value;

long getvideo(reg)
long reg;

PARAMETERS

reg      expects the name of the register to access.
value    expects the value which is to be placed into reg.

FUNCTION RETURN VALUE

The returned value of getvideo is the value read from register reg, or -1. -1 indicates that reg is not a valid register or that you queried a video register on a system without that particular board installed.

DESCRIPTION

setvideo sets the specified video hardware register to the specified value. getvideo returns the value of the specified video hardware register. Several different video boards are supported; the board names and register identifiers are listed below.

Display Engine Board
DE_R1

CG2 Composite Video and Genlock Board
CG_CONTROL
CG_CPHASE
CG_HPHASE
CG_MODE
VP1 Live Video Digitizer Board

VP_ALPHA
VP_BRITE
VP_CMD
VP_CONT
VP_DIGVAL
VP_FBXORG
VP_FBYORG
VP_FGMODE
VP_GBXORG
VP_GBYORG
VP_HBLANK
VP_HEIGHT
VP_HUE
VP_MAPADD
VP_MAPBLUE
VP_MAPGREEN
VP_MAPRED
VP_MAPSRC
VP_MAPSTROBE
VP_PIXCNT
VP_SAT
VP_STATUS0
VP_STATUS1
VP_VBLANK
VP_WIDTH

SEE ALSO

getmonitor, getothermonitor, setmonitor, videocmd

NOTES

These routines are available only in immediate mode.

The DE_R1 register is actually present only on the video board used in the IRIS-4D B, G, GT, and GTX models. It is emulated on all other models.
The Live Video Digitizer is available as an option for IRIS-4D GTX models only.

The symbolic constants named above are defined in the files `<gl/cg2vme.h>` and `<gl/vpl.h>`.
NAME

shademodel – selects the shading model

C SPECIFICATION

void shademodel(model)
long model;

PARAMETERS

model expects one of two possible flags:

FLAT, tells the system to assign the same color to each pixel of lines and polygons during scan conversion.

GOURAUD, tells the system to interpolate color from vertex to vertex when scan converting lines, and to interpolate color throughout the area of filled polygons when they are scan converted. This is the default shading model.

DESCRIPTION

shademodel determines the shading model that the system uses to render lines and filled polygons. When the system uses Gouraud shading, the colors along a line segment are an interpolation of the colors at its vertices, and the colors in the interior of a filled polygon are an interpolation of the colors at its vertices. Currently the interpolation is linear. Future architectures may do nonlinear interpolation to compensate for errors due to extreme projection.

When flat shading is specified, color is not interpolated. Rather, the color of the second vertex of a line segment, or of the last vertex of a polygon, is used at each pixel of the line segment or polygon. Thus connected lines, triangles, and quadrilaterals can be successfully flat shaded. For example, the color of the nth segment in a connected line is determined by the color of vertex n+1, and the color of the nth triangle in a mesh is determined by the color of vertex n+2.

Color intepolation or flat shading occurs after lighting and depthcueing calculations are made, so both these color generation operations can be either flat or Gouraud shaded. Texture, fog, and blending calculations, however, occur after pixel shading is completed. These operations are
always themselves Gouraud shaded, regardless of the current shade model. This means, for example, that a triangle mesh can be lighted or depthcued with flat shaded facets, then fogged or texture mapped smoothly.

SEE ALSO

getsm

NOTE

Gouraud is the default shading model, but is in general slower than flat shading. To improve performance, specify flat shading where Gouraud shading is not required.

BUGS

On IRIS-4D B and G models, and on the Personal Iris without Turbo Graphics, lines are always drawn with constant color regardless of the current shading model. Also, flat shaded independent polygons scan convert to a single color, but this color is not always that of the last vertex specified. Flat shaded polygons yeild consistent results on these models only when the same color is specified at each vertex.

On IRIS-4D GT and GTX models, lines drawn with move and draw are always flat shaded, and lines drawn with v commands are always Gouraud shaded, regardless of the current shading model. Independent polygons are always Gouraud shaded. Triangle meshes are correctly flat or Gouraud shaded, depending on the shading model.
NAME

shaderange – obsolete routine

C SPECIFICATION

void shaderange(lowin, highin, z1, z2)
Colorindex lowin, highin;
Screencoord z1, z2;

PARAMETERS

lowin  expects the low-intensity color map index.
highin expects the high-intensity color map index.
z1     expects the low z value to be mapped to.
z2     expects the high z value to be mapped to.

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use its replacement, lshaderange.

SEE ALSO

lshaderange
singlebuffer

NAME

singlebuffer — writes and displays all bitplanes

C SPECIFICATION

void singlebuffer()

PARAMETERS

none

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

cmode, doublebuffer, gconfig, getdisplaymode, gsync, RGBmode

NOTE

This routine is available only in immediate mode.
NAME

smoothline – obsolete routine

C SPECIFICATION

void smoothline(mode)
long mode;

PARAMETERS

mode  expects 1 to turn on antialiasing or 0 to turn it off.

DESCRIPTION

This routine is obsolete. Although it continues to function to provide backwards compatibility, all new development should use its identical replacement, linesmooth.

SEE ALSO

linesmooth
NAME
  spclos – obsolete routine

C SPECIFICATION
  void spclos()

PARAMETERS
  none

DESCRIPTION
  This routine is obsolete. Since setting of shademodel determines if a
gon is shaded, spclos simply functions as pclos. All new develop-
ment should use pclos.

SEE ALSO
  pclose, shademodel
NAME

splatz, splfi, splfs, splf2, splf2i, splf2s – draws a shaded filled polygon

C SPECIFICATION

void splf(n, parray, iarray)
long n;
Coord parray[][][3];
Colorindex iarray[];

void splfi(n, parray, iarray)
long n;
Icoord parray[][][3];
Colorindex iarray[];

void splfs(n, parray, iarray)
long n;
Scoord parray[][][3];
Colorindex iarray[];

void splf2(n, parray, iarray)
long n;
Coord parray[][][2];
Colorindex iarray[];

void splf2i(n, parray, iarray)
long n;
Icoord parray[][][2];
Colorindex iarray[];

void splf2s(n, parray, iarray)
long n;
Scoord parray[][][2];
Colorindex iarray[];

All of the above routines are functionally the same. They differ only in the type declarations of their parameters and in whether they assume a two- or three-dimensional space.
PARAMETERS

\( n \) expects the number of vertices in the polygon. There can be no more than 256 vertices in a single polygon.

\( parray \) expects an array containing the vertices of a polygon.

\( iarray \) expects the array containing the color map indices which determine the intensities of the vertices of the polygon.

DESCRIPTION

\texttt{splf} draws Gouraud-shaded polygons using the current pattern and writemask. Polygons are represented as arrays of points. The first and last points automatically connect to close a polygon. After the polygon is drawn, the current graphics position is set to the first point in the array. \texttt{splf} must be used in color map mode.

SEE ALSO

cmode, concave, poly, rect, rectf, pdr, pmv, rpdr, rpmv
NAME

stencil – alter the operating parameters of the stencil

C SPECIFICATION

void stencil(enable, ref, func, mask, fail, pass, zpass)
long enable;
unsigned long ref;
long func;
unsigned long mask;
long fail, pass, zpass;

PARAMETERS

enable expects either TRUE or FALSE, enabling or disabling stencil operation. When stencil operation is disabled (the default), the values of the subsequent six parameters are ignored,

ref expects a reference value used by the stencil compare function.

func expects one of eight flags specifying the stencil comparison function. These flags are SF_NEVER, SF_LESS, SF_EQUAL, SF_LEQUAL, SF_GREATER, SF_NOTEQUAL, SF_GEQUAL, and SF_ALWAYS.

mask expects a mask specifying which stencil bitplanes are significant during the comparison operation.

fail expects one of six flags indicating which stencil operation should be performed should the stencil test fail. The values are ST_KEEP, ST_ZERO, ST_REPLACE, ST_INCR, ST_DECR, and ST_INVERT.

pass expects one of six flags indicating which stencil operation should be performed should the stencil test pass, and the z-buffer test (if z-buffering is enabled) fail. The values are ST_KEEP, ST_ZERO, ST_REPLACE, ST_INCR, ST_DECR, and ST_INVERT.

zpass expects one of six flags indicating which stencil operation should be performed should the stencil and z-buffer tests pass. Its value is not significant when the z-buffer is not enabled. The values are ST_KEEP, ST_ZERO, ST_REPLACE,
ST_INCR, ST_DECR, and ST_INVERT.

DESCRIPTION

**stencil** operates as a superior z-buffer test with a different algorithm. When **stencil** is enabled, each pixel write first tests the stencil bitplanes. Both the color and z-buffer bitplane writes, as well as the write of the stencil bitplanes, are conditioned by the stencil test. **stencil** operation can be enabled only if stencil bitplanes are present (see **stensize**). Stencil bitplanes are present only in the normal framebuffer, so **stencil** should be called only while draw mode is **NORMALDRAW**.

When the z-buffer is enabled, three test cases are distinguished:

- **fail**  Stencil test fails.
- **pass**  Stencil test passes, but z-buffer test fails.
- **zpass**  Stencil test passes, and z-buffer test passes.

(When the z-buffer is not enabled, only cases **fail** and **pass** are considered.) In all three cases the stencil bitplanes are updated with a potentially new value. This value is a function of the case. The user specifies, for each case, which of six possible values will be used:

- **ST_KEEP**  Keep the current value (no change).
- **ST_ZERO**  Replace with zero.
- **ST_REPLACE**  Replace with the reference value.
- **ST_INCR**  Increment by one (clamp to max).
- **ST_DECR**  Decrement by one (clamp to zero).
- **ST_INVERT**  Invert all bits.

Arguments **fail**, **pass**, and **zpass** are each specified as one of **ST_KEEP**, **ST_ZERO**, **ST_REPLACE**, **ST_INCR**, **ST_DECR**, and **ST_INVERT**.

**ref** is the reference value used by the function that determines whether the stencil test passes or fails. **func** specifies the comparison between **ref** and the current stencil plane value. This comparison function is specified with the flags:
SF_NEVER       Never pass.
SF_LESS        Pass if ref is less than stencil.
SF_LEQUAL      Pass if ref is less than or equal to stencil.
SF_EQUAL       Pass if ref is equal to stencil.
SF_GREATER     Pass if ref is greater than stencil.
SF_GEQUAL      Pass if ref is greater than or equal to stencil.
SF_NOTEQUAL    Pass if ref is not equal to stencil.
SF_ALWAYS      Always pass.

The stencil bitplanes are treated as an unsigned integer of planes bits, where planes is the value passed to stensize to allocate the stencil buffer.

mask is a field that specifies which stencil bitplanes are to be considered by the test. It does not affect which bitplanes are updated.

If the z-buffer is enabled, color and depth fields are drawn only in the zpass case (both the stencil and depth tests pass). If the z-buffer is not enabled, color is drawn only in the pass case. The zpass case is ignored.

SEE ALSO
drawmode, polymode, sclear, stensize, swritemask, zbuffer

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support stencil. Use getgdesc to determine whether stencil is supported.

stencil is supported only in the normal framebuffer, and is therefore effective only while draw mode is NORMALDRAW.

BUGS

IRIS-4D VGX models do not support stencil operation when afunction is enabled.
NAME

stensize — specify the number of bitplanes to be used as stencil planes

C SPECIFICATION

void stensize(planes)
long planes;

PARAMETERS

planes number of bitplanes to be allocated as stencil planes. Only
values 0 through 8 are accepted. The default is 0.

DESCRIPTION

stensize specifies an alternate configuration of the normal framebuffer in
which some bitplanes are used as a stencil. planes specifies the number
of bitplanes to be used for stenciling. The constraints on planes, as well
as the relationship of the stencil bitplanes to the other normal bitplanes,
are machine dependent. Call getgdesc(GD_BITS_STENCIL) to deter-
mine how many bitplanes are available for stencil operation.

stensize takes effect only after gconfig has been called. Because stencil
bitplanes are available only in the normal framebuffer, stensize should
be called only while draw mode is NORMALDRAW.

SEE ALSO

drawmode, gconfig, stencil

NOTES

This routine is available only in immediate mode.
IRIS-4D B, G, GT, and GTX models, and the Personal Iris, do not sup-
port stensize.

BUGS

IRIS-4D VGX machines without the optional alpha bitplanes allocate
stencil bitplanes from the least-significant z-buffer bitplanes. Z-buffer
operation compensates for this allocation automatically, so the
programmer is aware of the allocation only when z-buffer contents are read back using lrectread. Use getgdesc to determine whether your machine has alpha bitplanes.
NAME

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

C SPECIFICATION

void stepunit(xunit, yunit)
long xunit, yunit;

PARAMETERS

xunit expects the amount of change per unit in the x direction. The amount is measured in pixels.
yunit expects the amount of change per unit in the y direction. The amount is measured in pixels.

DESCRIPTION

stepunit specifies the size of the change in a graphics window in discrete steps of xunit and yunit. Call stepunit at the beginning of a graphics program; it takes effect when you call winopen. stepunit resizes graphics windows in units of a standard size (in pixels). If winopen is not called, stepunit is ignored.

SEE ALSO

winopen, fudge

NOTE

This routine is available only in immediate mode.
NAME

strwidth – returns the width of the specified text string

C SPECIFICATION

long strwidth(str)
String str;

PARAMETERS

str expects the string that is to be measured.

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.

SEE ALSO

gextlwidth, mapw, mapw2

NOTE

This routine is available only in immediate mode.
NAME

subpixel – controls the placement of point, line, and polygon vertices

C SPECIFICATION

void subpixel(b)
    Boolean b;

PARAMETERS

b expects either FALSE or TRUE.
    FALSE forces screen vertices to the centers of pixels (default).
    TRUE positions screen vertices exactly.

DESCRIPTION

subpixel controls the placement of point, line, and polygon vertices in screen coordinates. By default subpixel is FALSE, causing vertices to be snapped to the center of the nearest pixel after they have been transformed to screen coordinates. Vertex snapping introduces artifacts into the scan conversion of lines and polygons. It is especially noticable when points or lines are drawn smooth (see pntsmerc and linesmerc). Thus subpixel is typically set to TRUE while smooth points or smooth lines are being drawn.

In addition to its effect on vertex position, subpixel also modifies the scan conversion of lines. Specifically, non-subpixel positioned lines are drawn closed, meaning that connected line segments both draw the pixel at their shared vertex, while subpixel positioned lines are drawn half open, meaning that connected lines segments share no pixels. (Smooth lines are always drawn half open, regardless of state of subpixel.) Thus subpixel positioned lines produce better results when logicop or blendfunction are used, but will produce different, possibly undesirable results in 2-D applications where the endpoints of lines have been carefully placed.

For example, using the standard 2-D projection:
orthof2(left-0.5,right+0.5,bottom-0.5,top+0.5);
viewport(left,right,bottom,top);

subpixel positioned lines match non-subpixel positioned lines pixel for pixel, except that they omit either the right-most or top-most pixel. Thus the non-subpixel positioned line drawn from (0,0) to (0,2) fills pixels (0,0), (0,1), and (0,2), while the subpixel positioned line drawn between the same coordinates fills only pixels (0,0) and (0,1).

SEE ALSO

linesmooth, pntsMOOTH

NOTES

This routine does not function on IRIS-4D B or G models.

The IRIS-4D GT and GTX models do not implement subpixel positioned polygons. They also do not implement subpixel positioned non-smooth lines.

On the Personal Iris polygons are always subpixel positioned, regardless of the value of subpixel. Subpixel positioned non-smooth lines are not implemented.
NAME

swapbuffers – exchanges the front and back buffers of the normal framebuffer

C SPECIFICATION

void swapbuffers()

PARAMETERS

none

DESCRIPTION

swapbuffers causes the front and back buffers of the normal framebuffer to be exchanged during the next vertical retrace period. Once an image is fully drawn in the back buffer, swapbuffers displays it. swapbuffers is ignored when the normal framebuffer is in single buffer mode. It has no effect on the overlay, underlay, or popup framebuffers, regardless of the current draw mode.

To swap overlay or underlay buffers, or to swap buffers in more than one framebuffer simultaneously, you must use mswapbuffers.

SEE ALSO

doublebuffer, drawmode, mswapbuffers, swapinterval
NAME

swapinterval – defines a minimum time between buffer swaps

C SPECIFICATION

void swapinterval(i)
short i;

PARAMETERS

i expects the number of retrace to wait before swapping the front and back buffers. The default interval is 1.

DESCRIPTION

swapinterval defines a minimum number of retrace between buffer swaps. For example, for a swap interval of 5, the system refreshes the screen at least five times between successive buffer swaps. swapinterval changes frames at a steady rate if a new image can be created within one swap interval.

Like the swapbuffers and mswapbuffers commands that it affects, swapinterval is ignored by framebuffers in single buffer mode.

A single swap interval counter is shared by the normal, overlay, and underlay framebuffers. The interval is enforced between all buffer swap requests, regardless of which framebuffers are swapped.

SEE ALSO

doublebuffer, drawmode, mswapbuffers, swapbuffers

NOTE

This routine is available only in immediate mode.
NAME

swaptmesh — toggles the triangle mesh register pointer

C SPECIFICATION

void swaptmesh()

PARAMETERS

none

DESCRIPTION

The triangle mesh hardware stores two vertices. After each new vertex is specified (and a triangle comprising the new vertex and the two stored vertices is drawn), one of the stored vertices is replaced by the new vertex. The value of a two-value pointer determines which vertex is replaced. This pointer is toggled after each vertex, causing alternate stored vertices to be replaced. swaptmesh toggles the pointer without specification of a new vertex (and no triangle is drawn).

SEE ALSO

bgntmesh, v

NOTE

Operation is undefined if swaptmesh is called before the second vertex of a triangle mesh is specified.
NAME

swinopen — creates a graphics subwindow

C SPECIFICATION

long swinopen(parent)
long parent;

PARAMETERS

parent expects the GID (graphics window identifier) of the window (or subwindow) in which you want to open a subwindow. The GID is the returned function value of a previous call to either swinopen or winopen.

FUNCTION RETURN VALUE

The returned value of this function is either a −1 or the graphics window identifier for the subwindow just created. Use this value to identify this subwindow to other graphics routines.

A returned function value of −1 indicates that the system cannot create any more graphics windows.

DESCRIPTION

swinopen creates a graphics subwindow. The graphics state of the new subwindow is initialized to its defaults (see greset) and it becomes the current window.

Subwindows have no window borders or window manager function buttons. Window constraints do not apply to subwindows. A subwindow is repositioned automatically when its parent is moved, so that its origin with respect to the parent’s origin remains constant. Resizing the parent does not automatically resize a subwindow, but keeps the distance between the upper left hand corners constant. Imaging in a subwindow is limited (clipped) to the area of the parent window.

After calling swinopen, the application must call winposition to specify the location of the subwindow’s boundaries with respect to the origin of its parent window.
If *parent* is the GID of a subwindow, the parent of that subwindow becomes the parent of the new subwindow for the purpose of positioning the subwindow.

When using the DGL (Distributed Graphics Library), the graphics window identifier also identifies the graphics server associated with the window.

`sbinopen` queues the pseudo devices INPUTCHANGE and REDRAW.

SEE ALSO

greset, winclose, winget, winopen, winposition, winset
NAME

swritemask — specify which stencil bits can be written

C SPECIFICATION

void swritemask(mask)
unsigned long mask;

PARAMETERS

mask expects a mask whose least-significant bits are used to control writing of the stencil bitplanes. Bitplanes corresponding to 1’s in the mask can be written, those corresponding to 0’s are read-only.

DESCRIPTION

swritemask specifies which of the stencil bitplanes are written both during normal stencil operation and stencil clear (see sclear). Bits 0 through planes-1 are significant, where planes is the current size of the stencil buffer.

Because only the normal framebuffer includes stencil bitplanes, swritemask should be called only while draw mode is NORMALDRAW.

SEE ALSO

drawmode, sclear, stencil, stensize

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support stencil bitplanes, and therefore do not support swritemask. Use getgdesc to determine whether stencil bitplanes are supported.
NAME
t2d, t2f, t2i, t2s — specify a texture coordinate

C SPECIFICATION
void t2s(vector)
short vector[2];
void t2i(vector)
long vector[2];
void t2f(vector)
float vector[2];
void t2d(vector)
double vector[2];

PARAMETERS

vector expects a 2-element array containing s and t texture coordinates. Put the s coordinate in element 0 of the array, and the t coordinate in element 1.

DESCRIPTION
t sets the current texture coordinates, s and t. The specified texture coordinates remain valid until they are replaced. All draw modes share the same texture coordinates.

Using texgen it is possible for one or both of the texture coordinates to be replaced by a graphics system generated value. The coordinate or coordinates that are not being replaced continue to be specified by t.

Both s and t are transformed, prior to use, by the Texture matrix, which is modified while mmode is MTEXTURE.

Texture coordinates are ignored while texture mapping is not enabled.

SEE ALSO
mmode, texgen, texdef2d, texbind, tevdef, tevbind
NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support texture mapping. \texttt{t} is ignored by these machines. Use \texttt{getgdesc} to determine whether texture mapping is supported.

\texttt{t} cannot be used while \texttt{mmode} is \texttt{MSINGLE}.
NAME

tevbind – selects a texture environment

C SPECIFICATION

void tevbind(target, index)
long target, index;

PARAMETERS

target expects the texture resource to which the environment definition is to be bound. There is only one appropriate resource, TV_ENV0.

index expects the name of the texture environment that is being bound. Name is the index passed to tevdef when the environment was defined.

DESCRIPTION

tevbind specifies which of the previously defined texture mapping environments is to be the current environment. The texture environment defines how the results of the texture function are applied. Texture environments are defined using tevdef.

By default environment definition 0 is bound to TV_ENV0. Texture mapping is enabled when an environment definition other than 0 is bound to TV_ENV0, and a texture definition other than 0 is bound to TX_TEXTURE_0. (See texbind.)

SEE ALSO

t, tevdef, texdef2d, texbind

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support texture mapping. tevbind is ignored by these machines. Use getgdesc to determine whether texture mapping is supported.
tevbind cannot be used while mmode is MSINGLE.
NAME

tevdef – defines a texture mapping environment

C SPECIFICATION

void tevdef(index, np, props)
long index, np;
float props[];

PARAMETERS

index expects the name of the environment being defined. Index 0 is reserved as a null definition, and cannot be redefined.

np expects the number of symbols and floating point values in props, including the termination symbol TV_NULL. If np is zero, it is ignored. Operation over network connections is more efficient when np is correctly specified, however.

props expects the array of floating point symbols and values that define the texture environment. props must contain a sequence of symbols, each followed by the appropriate number of floating point values. The last symbol must be TV_NULL, which is itself not followed by any values.

DESCRIPTION

Evaluation of the texture function at a pixel yields 1, 2, 3, or 4 values, depending on the value of nc passed to tevdef2d when the currently bound texture was defined. Texture environment determines how these texture values are used, not how they are computed or filtered. tevdef defines an environment based on options specified in the props array. If no options are specified, a reasonable default environment is defined.

Before the options can be defined, several conventions must be established:

1. The color components of the incoming pixel (prior to texture mapping) are referred to as Rin, Gin, Bin, and Ain.
2. The components of the texture function (computed at each pixel) are referred to as I, R, G, B, and A, depending on the number of components in the currently bound texture. For example, the single value of a 1-component texture function is referred to as I, while the four components of a 4-component texture are referred to as A (value 0), B (value 1), G (value 2), and R (value 3). Refer to the texdef2d manual page for an explanation of how texture function values correspond to the image pixels used to define the texture.

```
0123 (texture function value)
```

```
1-component texture  I
2-component texture  AI
3-component texture  BGR
4-component texture  ABGR
```

3. The components of the outgoing color that results from application of the texture function to the incoming pixel color, based on the texture environment, are Rout, Gout, Bout, and Aout.

Texture environment options are specified as a list of symbols, each followed by the appropriate number of floating point values, in the props array. The last symbol must be TV_NULL.

**TV_MODULATE** is the default texture environment. It specifies an environment in which incoming color components are multiplied by texture values. No floating point values follow this token. The exact arithmetic for 1, 2, 3, and 4 component texture functions is:

```
1-component: Rout=Rin*I, Gout=Gin*I, Bout=Bin*I, Aout=Ain
2-component: Rout=Rin*I, Gout=Gin*I, Bout=Bin*I, Aout=Ain*A
3-component: Rout=Rin*R, Gout=Gin*G, Bout=Bin*B, Aout=Ain
4-component: Rout=Rin*R, Gout=Gin*G, Bout=Bin*B, Aout=Ain*A
```

**TV_BLEND** specifies a texture environment in which texture function values are used to blend between the incoming color and the current texture environment color constant: (Rcon,Gcon,Bcon,Acon). No floating point values follow this token. Only 1 and 2 component texture functions have defined behavior when this environment is specified. The exact arithmetic for these texture functions is:
1-component: \[ \text{Rout} = \text{Rin} (1-I) + \text{Rcon} \times I \]
\[ \text{Gout} = \text{Gin} (1-I) + \text{Gcon} \times I \]
\[ \text{Bout} = \text{Bin} (1-I) + \text{Bcon} \times I \]
\[ \text{Aout} = \text{Ain} \]

2-component: \[ \text{Rout} = \text{Rin} (1-I) + \text{Rcon} \times I \]
\[ \text{Gout} = \text{Gin} (1-I) + \text{Gcon} \times I \]
\[ \text{Bout} = \text{Bin} (1-I) + \text{Bcon} \times I \]
\[ \text{Aout} = \text{Ain} \times A \]

3-component: undefined
4-component: undefined

**TV_DECAL** specifies a texture environment in which texture function alpha is used to blend between the incoming color and the texture function color. No floating point values follow this token. Only 3 and 4-component texture functions have defined behavior when this environment is specified. Note that the 3-component version simply outputs the texture colors, because no alpha texture component is available for blending. The exact arithmetic is:

1-component: undefined
2-component: undefined
3-component: \[ \text{Rout} = R \]
\[ \text{Gout} = G \]
\[ \text{Bout} = B \]
\[ \text{Aout} = \text{Ain} \]
4-component: \[ \text{Rout} = \text{Rin} (1-A) + R \times A \]
\[ \text{Gout} = \text{Gin} (1-A) + G \times A \]
\[ \text{Bout} = \text{Bin} (1-A) + B \times A \]
\[ \text{Aout} = \text{Ain} \]

**TV_COLOR** specifies the constant color used by the TV_BLEND environment. Four floating point values, in the range 0.0 through 1.0, must follow this symbol. These values specify \text{Rcon}, \text{Gcon}, \text{Bcon}, and \text{Acon}. By default, all are set to 1.0.
Symbols TV_MODULATE, TV_BLEND, and TV_DECAL are exclusive; only one should be included in the props array. If none are included, TV_MODULATE is chosen by default.

The texture environment is used to apply the results of the texture function to pixel color data after shading, but before fog is blended. Conditional pixel writes based on pixel alpha are computed after texture and fog are applied. (See afunction.) This allows texture transparency to control the conditional writing of pixels.

Each time an index is passed to tevdef, the definition corresponding to that index is completely respecified. Do not attempt to change a portion of a texture environment definition.

SEE ALSO
afunction, scrsubdivide, t, tevbind, texbind, texdef2d, texgen

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support texture mapping. tevdef is ignored by these machines. Use getgdesc to determine whether texture mapping is supported.

IRIS-4D VGX models without alpha bitplanes do not fully support 4-component textures. When a 4-component texture is used, it is treated by the texture environment as though it were a 3-component texture. Use getgdesc(GD_BITS_NORM_SNG_ALPHA) to determine whether alpha bitplanes are available.

BUGS

IRIS-4D VGX models do not support simultaneous texture mapping and polygon antialiasing. (See polysmooth.)
NAME

texbind – selects a texture function

C SPECIFICATION

    void texbind(target, index)
    long target, index;

PARAMETERS

    target expects the texture resource to which the texture function
definition is to be bound. There is only one appropriate
resource, TX_TEXTURE_0.

    index expects the name of the texture function that is being bound.
Name is the index passed to texdef2d when the texture function
was defined.

DESCRIPTION

    texbind specifies which of the previously defined texture mapping func-
tions is to be the current texture function. The texture function defines
how texture coordinates s and t are converted into a 1, 2, 3, or 4 value
result. Texture functions are defined using texdef2d.

    By default texture function definition 0 is bound to TX_TEXTURE_0.
Texture mapping is enabled when an texture function definition other
than 0 is bound to TX_TEXTURE_0, and a texture environment
definition other than 0 is bound to TV_ENV0. (See tevbind.)

SEE ALSO

t, tevbind, tevdef, texdef2d

NOTES

    IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support
texture mapping. texbind is ignored by these machines. Use getgdesc
to determine whether texture mapping is supported.
texbind cannot be used while mmode is MSINGLE.
NAME

texdef2d – convert a 2-dimensional image into a texture

C SPECIFICATION

void texdef2d(index, nc, width, height, image, np, props)
long index, nc, width, height;
unsigned long *image;
long np;
float *props;

PARAMETERS

index expects the name of the texture function being defined. Index 0 is reserved as a null definition, and cannot be redefined.

nc expects the number of 8-bit components per image pixel. 1, 2, 3, and 4 component textures are supported.

width expects the width of image in pixels.

height expects the height of image in pixels.

image expects an array of 4-byte words containing the pixel data. This image is in the format returned by lrectread, and accepted by lrectwrite.

np expects the number of symbols and floating point values in props, including the termination symbol TX_NULL. If np is zero, it is ignored. Operation over network connections is more efficient when np is correctly specified, however.

props expects the array of floating point symbols and values that define the texture function. props must contain a sequence of symbols, each followed by the appropriate number of floating point values. The last symbol must be TX_NULL, which is itself not followed by any values.

DESCRIPTION

A texture, or texture function, is a mapping of the texture coordinates s and t into 1, 2, 3, or 4 values. texdef2d defines such a mapping, named index, based on an image and a set of options. Currently texdef2d is the
only command available to specify such a mapping. Future Graphics Library releases may provide alternate mechanisms, however.

The image accepted by texdef2d must be in the format defined by lrectread and pixmode, and must be packed with 8, 16, 24, or 32 bits per pixel. \textit{nc} specifies both the number of bits per pixel expected in \textit{image}, and the number of components, or values, that will be generated by the texture function.

<table>
<thead>
<tr>
<th>nc</th>
<th>components</th>
<th>bits per pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>32</td>
</tr>
</tbody>
</table>

A 2-component image, for example, is treated as two separate 1-component images. Each 1-component image is used to define one texture function output. The pixel byte with the lowest address is pixel component 0. The 1-component image defined by these bytes generates texture function output 0. Thus 32-bit image pixels packed in the usual ABGR manner generate four texture function outputs: alpha (output 0), blue (output 1), green (output 2), and red (output 3). Refer to the \texttt{tevdef} manual page for an explanation of how the 1, 2, 3, or 4 texture function outputs are used to modify pixel color.

The dimensions of \textit{image} are specified by \textit{width} and \textit{height}. These values must be positive, otherwise they are unconstrained. Note in particular that \textit{image} need not have dimensions that are a power of 2.

Each texture function output is a filtered sampling of the corresponding image component, where texture coordinate \textit{s} is used as the \textit{x} address, and texture coordinate \textit{t} is used as the \textit{y} address. Regardless of the dimensions of the image, it is mapped into \textit{st}-coordinates such that its lower-left corner is (0,0), and its upper-right corner is (1,1). The way that \textit{s} and \textit{t} map onto the image when they are out of the range 0.0 through 1.0 is specified in the \textit{props} array.

A useful texture function can be defined by simply passing an image and a null \textit{props} array to texdef2d. The options specified in the \textit{props} array, however, allow significant control over both texture mapping quality and performance. The following symbols are accepted in \textit{props}:
**TX_MINFILTER** specifies the filter function used to generate the texture function output when multiple *image* pixels correspond to one pixel on the screen. It is followed by a single symbol that specifies which minification filter to use.

**TX_POINT** selects the value of the image pixel nearest to the exact *s*,*t* mapping onto the texture.
**TX_BILINEAR** selects the weighted average of the values of the four image pixels nearest to the exact *s*,*t* mapping onto the texture.
**TX_MIPMAP_POINT** chooses a prefILTERED version of the image, based on the number of image pixels that correspond to one screen pixel, then selects the value of the pixel that is nearest to the exact *s*,*t* mapping onto that image.
**TX_MIPMAP_LINEAR** chooses the two prefILTERED versions of the image that have the nearest image-pixel to screen-pixel size correspondence, then selects the weighted average of the values of the pixel in each of these images that is nearest the exact *s*,*t* mapping onto that image.
**TX_MIPMAP_BILINEAR** chooses a prefILTERED version of the image, based on the number of image pixels that correspond to one screen pixel, then selects the weighted average of the values of the four pixels nearest to the exact *s*,*t* mapping onto that image.

The default minification filter is **TX_MIPMAP_LINEAR**. Prefiltered versions of the image, when required by the minification filter, are computed automatically by the Graphics Library.

**TX_MAGFILTER** specifies the filter function used to generate the texture function output when multiple screen pixels correspond to one *image* pixel. It is followed by a single symbol that specifies which magnification filter to use. The magnification filter symbols are:

**TX_POINT** selects the value of the image pixel nearest to the exact *s*,*t* mapping onto the texture.
**TX_BILINEAR** selects the weighted average of the values of the four image pixels nearest to the exact *s*,*t* mapping onto the
texture.
The default magnification filter is TX_BILINEAR. 

TX_WRAP specifies how texture coordinates outside the range 0.0 through 1.0 are handled.

TX_REPEAT uses the fractional parts of the texture coordinates.

TX_CLAMP clamps the texture coordinates to the range 0.0 through 1.0.

The default texture coordinate handling is TX_REPEAT.

TX_WRAP_S is like TX_WRAP, but it specifies behavior only for the s texture coordinate.

TX_WRAP_T is like TX_WRAP, but it specifies behavior only for the t texture coordinate.

TX_TILE specifies a subregion of an image to be turned into a texture. It is followed by four floating point coordinates that specify the x and y coordinates of the lower-left corner of the subregion, then the x and y coordinates of the upper-right corner of the subregion. The original texture image continues to be addressed in the range 0,0 through 1,1. However, the subregion occupies only a fraction of this space, and pixels that map outside the subregion are not drawn.

If the image (or the specified subregion) is larger than can be handled by the hardware, it is reduced to the maximum supported size automatically (with no indication other than the resulting visual quality). Because subregions are specified independently, they should all be the same size (otherwise some may be reduced and others not).

TX_TILE supports mapping of high-resolution images with multiple rendering passes. By splitting the texture into multiple pieces, each piece can be rendered at the maximum supported texture resolution. For example to render a scene with 2x texture resolution, texdef2d is called four times with different indices, one for each quadrant of the original texture. The scene is then drawn four times, each time calling texbind with the texture id of one of the four quadrants. In each pass, only the pixels whose texture coordinates map within that quadrant are drawn.
Pixels outside of this quadrant are effectively clipped.

SEE ALSO
afunction, scrsubdivide, t, tevbind, tevdef, texbind, texgen

NOTES
IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support texture mapping. texdef2d is ignored by these machines. Use getgdesc to determine whether texture mapping is supported.

It is acceptable to define a 4-component texture function on an IRIS-4D VGX system that does not have alpha bitplanes. However, this definition will be treated as a 3-component definition by tevdef. Use getgdesc(GD_BITS_NORM_ALPHA) to determine whether alpha bitplanes are available.

Points, lines, and characters, as well as polygons, are texture mapped. Filter selection and wrap modes are applicable to lines. Points are filtered by the magnification filter, assuming a 1-to-1 correspondence between texture pixel and screen pixel size. Characters use the same magnification filter, but are mapped assuming that both s and t are zero.

IRIS screen pixels have integer coordinates at their centers. Texture images, however, have integer coordinates (0 and 1) at their exact edges, not at the centers of pixels on their edges.

BUGS
IRIS-4D VGX models require that, when using TX_TILE, width and height of both the original image and the specified subregion be a power of 2. Also, texture coordinate t is always clamped to the range 0.0 through 1.0 in this case, regardless of the value of TX_WRAP_T.

IRIS-4D VGX models require that, when TX_WRAP_S is set to TX_CLAMP, TX_WRAP_T also be set to TX_CLAMP. Otherwise operation is undefined.
NAME

texgen – specify automatic generation of texture coordinates

C SPECIFICATION

void texgen(coord, mode, params)
long coord, mode;
float params[];

PARAMETERS

coord Expects the name of the texture coordinate whose generation is
to be defined, enabled, or disabled. One of:

TX_S: The s texture coordinate
TX_T: The t texture coordinate

mode Expects the mode of generation to be specified, or an indication
that generation is to be either enabled or disabled. One of the
symbolic constants:

TG_CONTOUR: Use the plane equation specified in params
to define a plane in eye-coordinates. Generate a texture coordi-
nate that is proportional to vertex distance from this plane.

TG_LINEAR: Use the plane equation specified in params to
define a plane in object-coordinates. Generate a texture coordi-
nate that is proportional to vertex distance from this plane.

TG_ON: Enable the (previously defined) replacement for the
specified texture coordinate.

TG_OFF: Disable replacement of the specified texture coordi-
nate (the default).

params Expects a 4-component plane equation when mode is
TG_CONTOUR or TG_LINEAR. Array element 0 is plane
equation component A, 1 is B, 2 is C, and 3 is D. The contents
of params are insignificant when mode is TG_ON or
TG_OFF.
DESCRIPTION

Texture coordinates $s$ and $t$ can be specified directly using the `t` command. It is also possible to have texture coordinates generated automatically as a function of object geometry. `texgen` specifies, enables, and disables such automatic generation. Either or both texture coordinates can be generated independently. Automatic texture coordinate generation is disabled by default.

`texgen` supports two generation algorithms. `TG_LINEAR` operates directly on object coordinates, and is therefore most useful for textures that are locked to objects, such as ground texture locked to a terrain, or metallic texture locked to a cylinder. `TG_CONTOUR` operates on eye-space coordinates. It supports motion of an object through a ‘field’ of texture coordinates.

Both modes `TG_LINEAR` and `TG_CONTOUR` define a texture coordinate generation function that is a linear function of distance from a plane. The plane equation is specified as a single, 4-component, vector in object coordinates.

$$ P_{\text{object}} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} $$

The `TG_LINEAR` plane equation remains in object-coordinates. The `TG_CONTOUR` plane equation is transformed by the ModelView matrix into eye-coordinates when it is defined:

$$ P_{\text{eye}} = M^{-1}_{\text{ModelView}} P_{\text{object}} $$

When a generation function has been defined for a texture coordinate, and `texgen` has been called with `TG_ON`, each vertex presented to the graphics system has that texture coordinate value replaced with the distance of the vertex from the defined plane. For example, when texture coordinate $s$ is generated by a `TG_LINEAR` function, the generation function is:

$$ s = V_{\text{object}} \cdot P_{\text{object}} = \begin{bmatrix} V_{\text{x}}, \ V_{\text{y}}, \ V_{\text{z}}, \ V_{\text{w}} \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} $$
Alternately, when \( t \) is generated by a TG\_CONTOUR function, the generation function is:

\[ t = V_{\text{eye}} \cdot P_{\text{eye}} \]

where

\[ V_{\text{eye}} = V_{\text{object}} \cdot M_{\text{ModelView}}, \]

and

\[ P_{\text{eye}} = M_{\text{ModelView}}^{-1} \cdot P_{\text{object}} \]

Note that the ModelView matrix that modifies the plane equation is the ModelView matrix in effect when texgen was called, while the ModelView matrix that modifies the vertex coordinates is the matrix used to transform that vertex.

texgen generation functions remain valid until they are redefined. They are enabled and disabled without redefinition by calls to texgen with modes TG\_ON and TG\_OFF. texgen definition has no effect on the enable mode of the texture generation function.

When enabled, texgen replaces \( s \), \( t \), or both each time a vertex command is received. A texture coordinate that is not being generated continues to be specified by \( t \) commands. Texture coordinate are transformed by the Texture matrix (see mmode) following coordinate replacement by texgen.

SEE ALSO

mmode, t, texdef2d, texbind, tevdef, tevbind

NOTES

IRIS-4D G, GT, and GTX models, and the Personal Iris, do not support texture mapping. texgen is ignored by these machines. Use getgdesc to determine whether texture mapping is supported.

texgen cannot be used while mmode is MSINGLE.
NAME

textcolor – sets the color of text in the textport

C SPECIFICATION

void textcolor(tcolor)
Colorindex tcolor;

PARAMETERS

tcolor expects an index into the current color map.

DESCRIPTION

textcolor sets the color of all the text in the textport of the calling process. If the calling process was invoked from a wsh window, this window is used for its textport; otherwise, the process does not have a textport and this routine does nothing.

SEE ALSO

pagecolor


NOTES

This routine is available only in immediate mode.

A process launched from 4Sight or The IRIS WorkSpaceTM will not have a textport. Therefore, we do not recommend the use of this routine in new development.
NAME

textinit – initializes the textport

C SPECIFICATION

void textinit()

PARAMETERS

none

DESCRIPTION

textinit initializes the textport of the calling process to its default size, location, textcolor, and pagecolor. If the calling process was invoked from a wsh window, this window is used for its textport; otherwise, the process does not have a textport and this routine does nothing.

SEE ALSO

pagecolor, textcolor, textport, tpon

NOTES

This routine is available only in immediate mode.

A process launched from 4Sight or The IRIS WorkSpace™ will not have a textport. Therefore, we do not recommend the use of this routine in new development.
NAME

textport – positions and sizes the textport

C SPECIFICATION

void textport(left, right, bottom, top)
Screencoord left, right, bottom, top;

PARAMETERS

left    expects x screen coordinate for the left side of the textport.
right   expects x screen coordinate for the right side of the textport.
bottom  expects y screen coordinate for the bottom of the textport.
top     expects y screen coordinate for the top of the textport.

DESCRIPTION

textport positions and sizes the textport of the calling process to the
specified rectangle. If the calling process was invoked from a wsh win-
dow, this window is used for its textport; otherwise, the process does not
have a textport and this routine does nothing.

SEE ALSO

textinit, tpon

NOTES

This routine is available only in immediate mode.

A process launched from 4Sight or The IRIS WorkSpace™ will not have
a textport. Therefore, we do not recommend the use of this routine in
new development.
NAME

tie — ties two valuators to a button

C SPECIFICATION

void tie(b, v1, v2)
Device b, v1, v2;

PARAMETERS

b   expects a button.

v1  expects a valuator.

v2  expects a valuator.

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. The valuators v1 and v2 need not be (and probably should not be) queued.

You can tie one valuator to a button by calling tie with v2 set to NULLDEV. You can untie a button by calling tie with both v1 and v2 set to NULLDEV. v1 appears before v2 in the event queue; b precedes both v1 and v2.

SEE ALSO

gobutton

NOTES

This routine is available only in immediate mode.

The symbol NULLDEV is defined in <gl/device.h>.
NAME

tpon, tloff – control the visibility of the textport

C SPECIFICATION

void tpon()
void tloff()

PARAMETERS

none

DESCRIPTION

tpon pops the textport of the calling process, bringing it to the front of
any windows that conceal it. tloff pushes the textport down behind all
other windows, effectively hiding it. If the calling process was invoked
from a wsh window, this window is used for its textport; otherwise, the
process does not have a textport and this routine does nothing.

SEE ALSO

textinit, textport

NOTES

This routine is available only in immediate mode.
A process launched from 4Sight or The IRIS WorkSpace™ will not have
a textport. Therefore, we do not recommend the use of these routines in
new development.
NAME

translate – translates graphical primitives

C SPECIFICATION

void translate(x, y, z)
Coord x, y, z;

PARAMETERS

x  expects the x coordinate of a point in object space.
y  expects the y coordinate of a point in object space.
z  expects the z coordinate of a point in object space.

DESCRIPTION

translate moves the object space origin to a point specified in the
current object coordinate system. The translate routine 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

mmode, popmatrix, pushmatrix, rotate, scale
NAME

underlay – allocates bitplanes for display of underlay colors

C SPECIFICATION

void underlay(planes)
long planes;

PARAMETERS

planes expects the number of bitplanes to be allocated for underlay colors. Valid values are 0 (the default), 2, 4, and 8.

DESCRIPTION

The IRIS physical framebuffer is divided into four separate GL framebuffers: normal, popup, overlay, underlay. Because a single physical framebuffer is used to implement the four GL framebuffers, bitplanes must be allocated among the GL framebuffers. underlay specifies the number of bitplanes to be allocated to the underlay framebuffer. underlay does not take effect immediately. Rather, it is considered only when gconfig is called, at which time all requests for bitplane resources are resolved.

While only one of the four GL framebuffers can be drawn to at a time (see drawmode), all four are displayed simultaneously. The decision of which to display at each pixel is made based on the contents of the four framebuffers at that pixel location, using the following hierarchical rule:

if the popup pixel contents are non-zero
then display the popup bitplanes
else if overlay bitplanes are allocated AND the overlay pixel contents are non-zero
then display the overlay bitplanes
else if the normal pixel contents are non-zero OR no underlay bitplanes are allocated

Version 4.0 - 1 - April 1990
then  display the normal bitplanes
else  display the underlay bitplanes

Thus images drawn into the overlay framebuffer appear over images in the normal framebuffer, and images drawn into the underlay framebuffer appear under images in the normal framebuffer. Popup images appear over everything else.

The default configuration of the underlay framebuffer is 0 bitplanes. To make a change to this configuration other than to change the bitplane size, the drawing mode must be UNDERDRAW. For example, the underlay framebuffer can be configured to be double buffered by calling doublebuffer while draw mode is UNDERDRAW.

On models that cannot support overlay and underlay bitplanes simultaneously, calling underlay with a non-zero argument forces overlay to zero. When simultaneous overlay and underlay operation is supported, calling underlay may have no effect on the number of overlay bitplanes.

SEE ALSO
doublebuffer, drawmode, gconfig, getgdesc, singlebuffer, underlay

NOTES

This routine is available only in immediate mode.

IRIS-4D G, GT, and GTX models, and the Personal Iris, support only single buffered, color map mode underlay bitplanes.

The Personal Iris supports 0 or 2 underlay bitplanes. There are no overlay or underlay bitplanes in the minimum configuration of the Personal Iris.

IRIS-4D GT and GTX models support 0, 2, or 4 underlay bitplanes. Because 4-bitplane allocation reduces the popup framebuffer to zero bitplanes, however, its use is strongly discouraged. The window manager cannot operate properly when no popup bitplanes are available.

IRIS-4D VGX models support 0, 2, 4, or 8 underlay bitplanes, either single or double buffered, in color map mode only. The 4 and 8 bitplane allocations utilize the alpha bitplanes, which must be present, and which therefore are unavailable in draw mode NORMALDRAW.
BUGS

The Personal Iris does not support shade model GOURAUD in the underlay framebuffer.
NAME

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

C SPECIFICATION

void unqdevice(dev)
Device dev;

PARAMETERS

dev expects a device identifier

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

NOTE

This routine is available only in immediate mode.
NAME

v2d, v2f, v2i, v2s, v3d, v3f, v3i, v3s, v4d, v4f, v4i, v4s – transfers a 2-D, 3-D, or 4-D vertex to the graphics pipe

C SPECIFICATION

void v2s(vector);
short vector[2];

void v2i(vector);
long vector[2];

void v3s(vector);
short vector[3];

void v3i(vector);
long vector[2];

void v4s(vector);
short vector[4];

void v4i(vector);
long vector[4];

void v2f(vector);
float vector[2];

void v2d(vector);
double vector[2];

void v3f(vector);
float vector[3];

void v3d(vector);
double vector[2];

void v4f(vector);
float vector[4];

void v4d(vector);
double vector[4];

PARAMETERS

vector expects a 2, 3, or 4 element array, depending on whether you call the v2, v3, or v4 version of the routine. The elements of the array are the coordinates of the vertex (point) that you want to transfer to the graphics pipe. Put the x coordinate in element 0, the y coordinate in element 1, the z coordinate in element 2 (for v3 and v4), and the w coordinate in element 3 (for v4).

DESCRIPTION

v transfers a single 2-D (v2), 3-D (v3), or 4-D (v4) vertex to the graphics pipe. The coordinates are passed to v as an array. Separate subroutines are provided for 16-bit integers (s), 32-bit integers limited to a signed 24-bit range (i), 32-bit IEEE single precision floats (f), and 64-bit IEEE double precision floats (d). The z coordinate defaults to 0.0 if not specified. w defaults to 1.0.
The Graphics Library subroutines bgnpoint, endpoint, bgnline, endline, bgnclosedline, endclosedline, bgnpolygon, endpolygon, bgntmesh, endtmesh, bgnqstrip, and endqstrip determine how the vertex is interpreted. For example, vertices specified between bgnpoint and endpoint draw single pixels (points) on the screen. Likewise, those specified between bgnline and endline draw a sequence of lines (with the line stipple continued through internal vertices). Closed lines return to the first vertex specified, producing the equivalent of an outlined polygon.

Vertices specified when none of bgnpoint, bgnline, bgnclosedline, bgnpolygon, bgntmesh, and bgnqstrip are active simply set the current graphics position. They do not have any effect on the frame buffer contents. (Refer to the pages for bgnpoint, bgnline, bgnclosedline, bgnpolygon, bgntmesh, and bgnqstrip for their effect on the current graphics position.)

SEE ALSO
bgnclosedline, bgnline, bgnpoint, bgnpolygon, bgntmesh, bgnqstrip
NAME

`videocmd` – initiates a command transfer sequence on an optional video peripheral

C SPECIFICATION

```c
void videocmd(cmd)
long cmd;
```

PARAMETERS

`cmd` expects a command value which initiates a command transfer sequence on a video peripheral. The valid command tokens are:

- `VP_INITNTSC_COMP` initialize the optional Live Video Digitizer for a composite NTSC video source.
- `VP_INITNTSC_RGB` initialize the Live Video Digitizer for a RGB NTSC video source.
- `VP_INITPAL_COMP` initialize the Live Video Digitizer for a composite PAL video source.
- `VP_INITPAL_RGB` initialize the Live Video Digitizer for a RGB PAL video source.

DESCRIPTION

`videocmd` allows you to initialize the Live Video Digitizer peripheral board. Four command tokens are recognized; these initialize the board for either an NTSC video source (composite or RGB) or a PAL video source (composite or RGB).

SEE ALSO

- `getmonitor`, `getothermonitor`, `setmonitor`, `setvideo`
NOTES

This routine is available only in immediate mode.

The Live Video Digitizer is available as an option for IRIS-4D GTX models only.

The symbolic constants named above are defined in the file `<gl/vp1.h>`.
NAME

viewport – allocates an area of the window for an image

C SPECIFICATION

void viewport(left, right, bottom, top)
Screencoord left, right, bottom, top;

PARAMETERS

left  expects x location (in pixels) of left side of viewport.
right expects x location (in pixels) of right side of viewport.
bottom expects y location (in pixels) of bottom of viewport.
top   expects y location (in pixels) of top of viewport.

DESCRIPTION

viewport specifies, in pixels, the area of the window that displays an image. The viewport locations are specified relative to the lower-left corner of the window. 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

NOTE

On the Personal Iris, if left is greater than right or bottom is greater than top, the screen displays a reflected image.
NAME

winattach – obsolete routine

C SPECIFICATION

long winattach() 

PARAMETERS

none

DESCRIPTION

This routine is obsolete and does not function. Currently, there is no replacement.

NOTE

This routine is available only in immediate mode.
NAME

winclose – closes the identified graphics window

C SPECIFICATION

```c
void winclose(gwid)
long gwid;
```

PARAMETERS

`gwid` expects the identifier for the graphics window that you want closed.

DESCRIPTION

`winclose` closes the graphics window associated with identifier `gwid`. The identifier for a window is the function return value from the call to `winopen` that created the window.

When using the Distributed Graphics Library (DGL), the graphics window identifier also identifies the graphics server associated with the window. The DGL directs all subsequent Graphics Library input and output to the server associated with `gwid`.

If the window being closed is on a screen for which the `getgdesc` inquiry `GD_SCRNTYPE` returns `GD_SCRNTYPE_NOWM`, `winclose` leaves the image undisturbed.

SEE ALSO

`getgdesc`, `winopen`

NOTE

This routine is available only in immediate mode.
NAME

winconstraints – binds window constraints to the current window

C SPECIFICATION

void winconstraints()

PARAMETERS

none

DESCRIPTION

winconstraints binds the currently specified constraints to the current graphics window. (Logically, because this assumes the existence of a current graphics window, you must have previously called winopen.) Prior to calling winconstraints, you can set the the values of the window constraints by using the following commands: minsize, maxsize, keepaspect, prefsize, iconsize, noborder, noport, stepunit, fudge, and imakebackground. Note the absence from this list of prefposition; the position of a window can not be constrained.

After binding these constraints to a window, winconstraints resets the window constraints to their default values, if any.

SEE ALSO

fudge, keepaspect, iconsize, imakebackground, maxsize, minsize, noborder, noport, prefposition, prefsize, stepunit, winopen

NOTE

This routine is available only in immediate mode.
NAME

windepth – measures how deep a window is in the window stack

C SPECIFICATION

long windepth(gwid)
long gwid;

PARAMETERS

gwid expects the window identifier for the window you want to test.

FUNCTION RETURN VALUE

The returned value of this function is a number that you can use to
determine that stacking order of windows on the screen.

DESCRIPTION

windepth returns a number which can be compared against the win-
depth return value for other windows to determine the stacking order of
a programs windows on the screen.

When using the Distributed Graphics Library DGL), the graphics win-
dow identifier also identifies the graphics server associated with the win-
dow. The DGL directs all subsequent Graphics Library input and output
to the server associated with gwid.

SEE ALSO

winpush, winpop

NOTE

This routine is available only in immediate mode.
NAME

window – defines a perspective projection transformation

C SPECIFICATION

void window(left, right, bottom, top, near, far)
Coord left, right, bottom, top, near, far;

PARAMETERS

left  expects x coordinate of left side of viewing volume.
right expects x coordinate of right side of viewing volume.
bottom expects y coordinate of bottom of viewing volume.
top   expects y coordinate of top of viewing volume.
near  expects the z coordinate of the near clipping plane.
far   expects the z coordinate of the far clipping plane.

DESCRIPTION

window specifies the position and size of the rectangular viewing frustum closest to the eye (in the near clipping plane), and the location of the far clipping plane. All objects contained within this volume are projected in perspective onto the screen area that viewport defines.

When the system is in single matrix mode, window loads a matrix onto the transformation stack, replacing the current top matrix. When the system is in viewing, projection, or texture matrix mode, window replaces the current Projection matrix and leaves the ModelView matrix stack and the Texture matrix unchanged.

SEE ALSO

mmode, ortho, perspective, viewport
NAME

winget — returns the identifier of the current graphics window

C SPECIFICATION

`long winget()`

PARAMETERS

`none`

FUNCTION RETURN VALUE

The returned value for this function is the identifier of the current graphics window.

DESCRIPTION

`winget` returns the identifier of the current graphics window. The current graphics window is the window to which the system directs the output from graphics routines.

SEE ALSO

`winset`

NOTE

This routine is available only in immediate mode.
NAME

winmove – moves the current graphics window by its lower-left corner

C SPECIFICATION

void winmove(orgx, orgy)
long orgx, orgy;

PARAMETERS

orgx expects the x coordinate of the location to which you want to move the current graphics window.

orgy expects the y coordinate of the location to which you want to move the current graphics window.

DESCRIPTION

winmove moves the current graphics window so that its origin is at the screen coordinates (in pixels) specified by orgx, orgy. The origin of the current graphics window is its lower-left corner. winmove does not change the size and shape of the window.

SEE ALSO

winposition

NOTE

This routine is available only in immediate mode.
NAME

`winopen` — creates a graphics window

C SPECIFICATION

```c
long winopen(name)
String name;
```

PARAMETERS

`name` expects the window title that is displayed on the left hand side of the title bar for the window. If you do not want a title, pass a zero-length string.

FUNCTION RETURN VALUE

The returned value for this function is the graphics window identifier for the window just created. Use this value to identify the graphics window to other windowing functions. Only the lower 16 bits are significant, since a graphics window identifier is the value portion of a REDRAW event queue entry. If no additional windows are available, this function returns -1.

DESCRIPTION

`winopen` creates a graphics window as defined by the current values of the window constraints on the currently selected screen. This new window becomes the current window. If this is the first time that your program has called `winopen`, the system also initializes the Graphics Library.

Except for size and location, the system maintains default values for the constraints on a window. You can change these default window constraints if you call the routines `minsize`, `maxsize`, `keepaspect`, `prefsize`, `prefposition`, `stepunit`, `fudge`, `iconsizes`, `noborder`, `noproject`, `imakebackground`, and `foreground` before you call `winopen`. If the a window's size and location (or both) are left unconstrained, the system allows the user to place and size the window.
The selected screen defaults to the screen with the input focus at the time the first Graphics Library routine is called. You can change it using the routine scrmselect.

winopen sets the graphics state of the new window (this includes window constraints) to its default values; there are listed in the table below. It also queues the pseudo devices INPUTCHANGE and REDRAW.

When using the Distributed Graphics Library (DGL), the window identifier also identifies the window's graphics server. The DGL directs all graphics input and output to the current window's server; subsequent Graphics Library subroutines are executed by the window's server.

<table>
<thead>
<tr>
<th>State</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acsize</td>
<td>0</td>
</tr>
<tr>
<td>afunction</td>
<td>AF_ALWAYSS</td>
</tr>
<tr>
<td>backbuffer</td>
<td>FALSE</td>
</tr>
<tr>
<td>backface</td>
<td>FALSE</td>
</tr>
<tr>
<td>blendfunction</td>
<td>BF_ONE, BF_ZERO</td>
</tr>
<tr>
<td>buffer mode</td>
<td>single</td>
</tr>
<tr>
<td>character position</td>
<td>undefined</td>
</tr>
<tr>
<td>clipplane</td>
<td>CP_OFF</td>
</tr>
<tr>
<td>color</td>
<td>0</td>
</tr>
<tr>
<td>color mode</td>
<td>single color map</td>
</tr>
<tr>
<td></td>
<td>(cmode and onemap)</td>
</tr>
<tr>
<td>concave</td>
<td>FALSE</td>
</tr>
<tr>
<td>curveprecision</td>
<td>undefined</td>
</tr>
<tr>
<td>depth range</td>
<td>Zmin, Zmax</td>
</tr>
<tr>
<td>depthcue</td>
<td>FALSE</td>
</tr>
<tr>
<td>drawmode</td>
<td>NORMALDRAW</td>
</tr>
<tr>
<td>feedback mode</td>
<td>off</td>
</tr>
<tr>
<td>fogvertex</td>
<td>FG_OFF</td>
</tr>
<tr>
<td>font</td>
<td>0</td>
</tr>
<tr>
<td>frontbuffer</td>
<td>TRUE</td>
</tr>
<tr>
<td>frontface</td>
<td>FALSE</td>
</tr>
<tr>
<td>full screen mode</td>
<td>off</td>
</tr>
<tr>
<td>State</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>glcompat</td>
<td></td>
</tr>
<tr>
<td>GLCOLDPOLYGON</td>
<td>1</td>
</tr>
<tr>
<td>GLCZRANGEMAP</td>
<td>1 (B and G models)</td>
</tr>
<tr>
<td></td>
<td>0 (other models)</td>
</tr>
<tr>
<td>graphics position</td>
<td>undefined</td>
</tr>
<tr>
<td>linesmooth</td>
<td>SML_OFF</td>
</tr>
<tr>
<td>linestack</td>
<td>0 (solid)</td>
</tr>
<tr>
<td>linewidth</td>
<td>1</td>
</tr>
<tr>
<td>lmapcolor</td>
<td>LMC_COLOR</td>
</tr>
<tr>
<td>lmded</td>
<td></td>
</tr>
<tr>
<td>LIGHTn</td>
<td>0</td>
</tr>
<tr>
<td>LMODEL</td>
<td>0</td>
</tr>
<tr>
<td>MATERIAL</td>
<td>0</td>
</tr>
<tr>
<td>logicop</td>
<td>LO_SRC</td>
</tr>
<tr>
<td>lsrepeat</td>
<td>1</td>
</tr>
<tr>
<td>mapcolor</td>
<td>no entries changed</td>
</tr>
<tr>
<td>matrix</td>
<td></td>
</tr>
<tr>
<td>ModelView</td>
<td>undefined</td>
</tr>
<tr>
<td>Projection</td>
<td>undefined</td>
</tr>
<tr>
<td>Single</td>
<td>ortho2 matching window size</td>
</tr>
<tr>
<td>Texture</td>
<td>undefined</td>
</tr>
<tr>
<td>ndmode</td>
<td>MSINGLE</td>
</tr>
<tr>
<td>name stack</td>
<td>empty</td>
</tr>
<tr>
<td>ndmode</td>
<td>N/AUTO</td>
</tr>
<tr>
<td>normal vector</td>
<td>undefined</td>
</tr>
<tr>
<td>overlay</td>
<td>2</td>
</tr>
<tr>
<td>patchbasis</td>
<td>undefined</td>
</tr>
<tr>
<td>patchcurves</td>
<td>undefined</td>
</tr>
<tr>
<td>patchprecision</td>
<td>undefined</td>
</tr>
<tr>
<td>pattern</td>
<td>0 (solid)</td>
</tr>
<tr>
<td>pick mode</td>
<td>off</td>
</tr>
<tr>
<td>picksizes</td>
<td>10×10</td>
</tr>
<tr>
<td>pixmode</td>
<td>standard</td>
</tr>
<tr>
<td>pntsmooth</td>
<td>SMP_OFF</td>
</tr>
<tr>
<td>polymode</td>
<td>PYM_FILL</td>
</tr>
<tr>
<td>State</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>polysmooth</td>
<td>PYSM_OFF</td>
</tr>
<tr>
<td>readsource</td>
<td>SRC_AUTO</td>
</tr>
<tr>
<td>rectzoom</td>
<td>1.0, 1.0</td>
</tr>
<tr>
<td>RGB color</td>
<td>all components 0</td>
</tr>
<tr>
<td></td>
<td>(when RGB mode is entered)</td>
</tr>
<tr>
<td>RGB shade range</td>
<td>undefined</td>
</tr>
<tr>
<td>RGB writemask</td>
<td>all components 0xFF</td>
</tr>
<tr>
<td></td>
<td>(when RGB mode is entered)</td>
</tr>
<tr>
<td>scrbox</td>
<td>SB_RESET</td>
</tr>
<tr>
<td>scrmask</td>
<td>size of window</td>
</tr>
<tr>
<td>scrsubdivide</td>
<td>SS_OFF</td>
</tr>
<tr>
<td>select mode</td>
<td>off</td>
</tr>
<tr>
<td>shade range</td>
<td>0, 7, Zmin, Zmax</td>
</tr>
<tr>
<td>shademodel</td>
<td>GOURAUD</td>
</tr>
<tr>
<td>stencil</td>
<td>disabled</td>
</tr>
<tr>
<td>stensize</td>
<td>0</td>
</tr>
<tr>
<td>swritemask</td>
<td>all planes enabled</td>
</tr>
<tr>
<td>tevbind</td>
<td>0 (off)</td>
</tr>
<tr>
<td>texbind</td>
<td>0 (off)</td>
</tr>
<tr>
<td>texgen</td>
<td>TG_OFF</td>
</tr>
<tr>
<td>underlay</td>
<td>0</td>
</tr>
<tr>
<td>viewport</td>
<td>size of window</td>
</tr>
<tr>
<td>writemask</td>
<td>all planes enabled</td>
</tr>
<tr>
<td>zbuffer</td>
<td>FALSE</td>
</tr>
<tr>
<td>zdraw</td>
<td>FALSE</td>
</tr>
<tr>
<td>zfunction</td>
<td>ZF_LEQUAL</td>
</tr>
<tr>
<td>zsource</td>
<td>ZSRC_DEPTH</td>
</tr>
<tr>
<td>zwritemask</td>
<td>all planes enabled</td>
</tr>
</tbody>
</table>

Notes

- Font 0 is a Helvetica-like font.
- $Z_{\text{min}}$ and $Z_{\text{max}}$ are the minimum and maximum values that you can store in the $z$-buffer. These depend on the graphics hardware and are returned by getgdesc(GD_ZMIN) and getgdesc(GD_ZMAX).
- On IRIS-4D B and G models, `winopen` also sets `lsbackup(FALSE)` and `resetls(TRUE)`.

SEE ALSO

foreground, fudge, iconsize, imakebackground, keepaspect, minsize, maxsize, noborder, noport, prefsize, prefposition, scrmselect, stepunit, winclose

4Sight User’s Guide, “Using the GL/DGL Interfaces”.

NOTE

This routine is available only in immediate mode.
NAME

winpop – moves the current graphics window in front of all other windows

C SPECIFICATION

void winpop()

PARAMETERS

none

DESCRIPTION

When more than one window tries to occupy the same space on the screen, the system stacks them on top of each other—thus obscuring (either partially or completely) the underlying graphics window or windows.

Use winpop to take the current graphics window from anywhere in the stack of windows and place it on top.

SEE ALSO

winpush

NOTE

This routine is available only in immediate mode.
NAME

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

C SPECIFICATION

void winposition(x1, x2, y1, y2)
long x1, x2, y1, y2;

PARAMETERS

x1 expects the x screen coordinate (in pixels) of the first corner of the new location for the current graphics window. The first corner of the new window is the corner diagonally opposite the second corner.

x2 expects the x screen coordinate (in pixels) of the second corner of the new location for the current graphics window.

y1 expects the y screen coordinate (in pixels) of the first corner of the new location for the current graphics window.

y2 expects the y screen coordinate (in pixels) of the second corner of the new location for the current graphics window.

DESCRIPTION

winposition moves and reshapes the current graphics window to match the screen coordinates x1, x2, y1, y2 (calculated in pixels). This differs from prefposition because the reshaped window is not fixed in size and shape, and can be reshaped interactively.

SEE ALSO

prefposition, prefsize, winmove

NOTE

This routine is available only in immediate mode.
NAME

\texttt{winpush} – places the current graphics window behind all other windows

C SPECIFICATION

\texttt{void winpush()}

PARAMETERS

\textit{none}

DESCRIPTION

When more than one window tries to occupy the same space on the screen, the system stacks them on top of each other—thus obscuring (either partially or completely) the underlying graphics window or windows.

Use \texttt{winpush} to take the current graphics window from anywhere in the stack of windows and push it to the bottom.

SEE ALSO

\texttt{winpop}

NOTE

This routine is available only in immediate mode.
NAME

`winset` - sets the current graphics window

C SPECIFICATION

```c
void winset(gwid)
long gwid;
```

PARAMETERS

`gwid` expects a graphics window identifier.

DESCRIPTION

`winset` takes the graphics window associated with identifier `gwid` and makes it the current window. The system directs all graphics output to the current graphics window.

When using the Distributed Graphics Library (DGL), the graphics window identifier also identifies the graphics server associated with the window. The DGL directs all subsequent Graphics Library input and output to the server associated with `gwid`.

SEE ALSO

`winget`

NOTE

This routine is available only in immediate mode.
NAME

wintitle — adds a title bar to the current graphics window

C SPECIFICATION

void wintitle(name)
String name;

PARAMETERS

name expects the title you want displayed in the title bar of the current graphics window.

DESCRIPTION

wintitle adds a title to the current graphics window. Use wintitle("") to clear the title.

SEE ALSO

winopen

NOTE

This routine is available only in immediate mode.
NAME

wmpack - specifies RGBA writemask with a single packed integer

C SPECIFICATION

    void wmpack(pack)
    unsigned long pack;

PARAMETERS

    pack    expects a packed integer containing the RGBA (red, green, blue, alpha) values you want to assign as the current write mask. Expressed in hexadecimal, the format of the packed integer is 0xaabbggrr, where:

    aa      is the alpha value,
    bb      is the blue value,
    gg      is the green value, and
    rr      is the red value.

    RGBA component values range from 0 to 0xFF (255).

DESCRIPTION

wmpack sets the red, green, blue, and alpha write mask components of the currently active GL framebuffer, one of normal, popup, overlay, or underlay as specified by drawmode. The current framebuffer must be in RGB mode for the wmpack command to be applicable. All drawing into the color bitplanes of the current framebuffer is masked by the current write mask. Write mask components are retained in each draw mode, so when a draw mode is re-entered, the red, green, blue, and alpha masks are reset to the last values specified in that draw mode.

Each write mask component is an 8-bit mask, which allows changes only to bitplanes corresponding to ones in the mask. For example, wmpack(0xFF0000F0) allows changes only to the 4 most significant bits of red, and to all the bits of alpha.

It is an error to call wmpack while the current framebuffer is in color map mode.
The write mask components of all framebuffers in RGB mode are set to 0xFF when gconfig is called.

SEE ALSO
cpack, drawmode, gRGBmask, RGBmode

NOTE
Because only the normal framebuffer currently supports RGB mode, wmpack should be called only while draw mode is NORMALDRAW. Use getgdesc to determine whether RGB mode is available in draw mode NORMALDRAW.
NAME

writemask – grants write permission to bitplanes

C SPECIFICATION

void writemask(wtm)
Colorindex wtm;

PARAMETERS

wtm expects a mask whose bits control which bitplanes are available for drawing and which are read only.
The mask contains one bit per available bitplane. If a bit is set in the writemask, the system writes the current color index into the corresponding bitplane. If a bit is set to zero in the writemask, the corresponding bitplane is read-only.

DESCRIPTION

Use writemask to reserve bitplanes for special purposes. When the writemask marks a biplane as read-only, that bitplane is write protected from ordinary drawing routines.

Use RGBwritemask in RGB mode.

SEE ALSO

color, drawmode, RGBwritemask
NAME

`writepixels` — paints a row of pixels on the screen

C SPECIFICATION

```c
void writepixels(n, colors)
short n;
Colorindex colors[];
```

PARAMETERS

- `n` expects the number of pixels you want to paint.
- `colors` expects an array of color indices. The system reads `n` elements from this array and writes a pixel of the appropriate color for each.

DESCRIPTION

`writepixels` paints a row of pixels on the screen in color map mode. The starting location is the current character position. The system updates the current character position to one pixel to the right of the last painted pixel. 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. The current character position becomes undefined if the new position is outside the viewport.

The system must be in color map mode for `writepixels` to function correctly.

SEE ALSO

`lrectwrite`

NOTES

`writepixels` should not be used in new development. Rather, pixels should be written using the high-performance `lrectwrite` command.
This routine is available only in immediate mode.
NAME

writeRGB – paints a row of pixels on the screen

C SPECIFICATION

void writeRGB(n, red, green, blue)
short n;
RGBvalue red[], green[], blue[];

PARAMETERS

n     expects the number of pixels that you want to paint.
red   expects an array containing red values for the pixels you paint.
      You need a red value for each pixel you paint.
green expects an array containing green values for the pixels you paint.
      You need a green value for each pixel you paint.
blue  expects an array containing blue values for the pixels you paint.
      You need a blue value for each pixel you paint.

DESCRIPTION

writeRGB paints a row of pixels on the screen in RGB mode. The
starting location is the current character position. The system updates
the current character position to one pixel to the right of the last painted
pixel. 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. The current character position becomes undefined if the new
position is outside the viewport.

writeRGB supplies a 24-bit RGB value (8 bits for each color) for each
pixel. This value is written directly into the bitplanes.

SEE ALSO

lrectwrite
NOTES

`writeRGB` should not be used in new development. Rather, pixels should be written using the high-performance `irectwrite` command.

This routine is available only in immediate mode.

When there are only 12 color bitplanes available, the lower 4 bits of each color are ignored.
NAME

xfpt, xfpti, xfpts, xfpt2, xfpt2i, xfpt2s, xfpt4, xfpt4i, xfpt4s – multiplies a point by the current matrix in feedback mode

C SPECIFICATION

void xfpt(x, y, z)
Coord x, y, z;

void xfpti(x, y, z)
Icoord x, y, z;

void xfpts(x, y, z)
Scoord x, y, z;

void xfpt2(x, y)
Coord x, y;

void xfpt2i(x, y)
Icoord x, y;

void xfpt2s(x, y)
Scoord x, y;

void xfpt4(x, y, z, w)
Coord x, y, z, w;

void xfpt4i(x, y, z, w)
Icoord x, y, z, w;

void xfpt4s(x, y, z, w)
Scoord x, y, z, w;

PARAMETERS

x  expects the x coordinate of a point.

y  expects the y coordinate of a point.

z  expects the z coordinate of a point. Used only by the 3-D and 4-D versions of the routines; 0.0 is assumed by the others.

w  expects the w coordinate of a point. Used only by the 4-D version of the routines; 1.0 is assumed by the others.
DESCRIPTION

\texttt{xfpt} multiplies the specified point \((x, y, 0.0, 1.0)\), \((x, y, z, 1.0)\), or \((x, y, z, w)\) by the current matrix in the Geometry Pipeline. The 4-D result is not clipped or scaled, and is placed in the feedback buffer.

SEE ALSO


NOTES

This routine is available only in feedback mode; otherwise it is ignored.

This routine functions only on IRIS-4D B and G models, and we advise against its use for new development.

The processor can access full words only on full-word boundaries. \texttt{xfpt} does not guarantee such alignment. See the \textit{Graphics Library Programming Guide}, Feedback Mode, for information on successful alignment.
NAME

`zbuffer` — enable or disable z-buffer operation in the current framebuffer

C SPECIFICATION

```c
void zbuffer(bool)
Boolean bool;
```

PARAMETERS

`bool` expects one of two possible values:

- `TRUE` enables z-buffer operation.
- `FALSE` disables z-buffer operation.

DESCRIPTION

`zbuffer` turns z-buffer mode off or on for the current framebuffer, one of normal, popup, overlay, and underlay, as specified by `drawmode`. The z-buffer is a bitplane bank that is associated with a single framebuffer, and that stores a depth value for each pixel in that framebuffer. When z-buffer operation is enabled, the depth value associated with each incoming pixel is compared to the depth value stored in the framebuffer at that pixel location. The comparison function is specified by `zfunction`. If the comparison passes, the incoming pixel color is written into the color bitplane bank or banks, and the incoming pixel depth is written into the z-buffer bitplanes. The current z write mask controls which z-buffer bitplanes are written with new depth data.

If the comparison fails, no change is made to the contents of either the color bitplane banks or the z-buffer bitplane bank. In some cases, however, a change is made to the contents of the stencil bitplanes.

By default z-buffer operation is disabled in all framebuffers.

SEE ALSO

drawmode, getzbuffer, lsetdepth, stencil, zclear, zdraw, zfunction, zsource, zwritemask
NOTE

On some models z-buffer hardware is optional. Call getgdesc(GD_BITS_NORM_ZBUFFER) to determine whether z-buffer hardware is available.

Currently z-buffer operation is supported only in the normal framebuffer. To insure compatibility with future releases of the GL, make calls to zbuffer only while draw mode is NORMALDRAW.

BUG

IRIS-4D GT and GTX models accept z-buffer commands, and support z-buffer operation using the normal z-buffer, when the draw mode is PUPDRAW, OVERDRAW, and UNDERDRAW. This operation is incorrect and will be changed in a future release of the Graphics Library.
NAME

zclear – initializes the z-buffer of the current framebuffer

C SPECIFICATION

void zclear()

PARAMETERS

none

DESCRIPTION

zclear sets the z-buffer in the area of the viewport to
getgdesc(GD_ZMAX), the largest positive z value supported. Typically zclear is called prior to rendering each frame. If you intend to
clear the color bitplanes as well as the z-buffer, or if you require control
of the value written to the z-buffer, call czclear instead of zclear.

Because only the normal framebuffer includes a z buffer, zclear should
be called only while draw mode is NORMALDRAW. Also, the current
z writemask controls which z-buffer bitplanes are modified during
zclear execution, and screenmask, when it is set to a subregion of the
viewport, bounds the cleared region. Other drawing modes, including
polygon fill pattern, stenciling, texture mapping, writemask, and z
buffering, have no effect on the operation of zclear.

After zclear executes, the graphics position is undefined.

SEE ALSO

drawmode, getgdesc, scrmask, setpattern, stencil, texbind, wmpack, writemask, zbuffer, zwritemask
NAME

zdraw – enables or disables drawing to the z-buffer

C SPECIFICATION

void zdraw(b)
Boolean b;

PARAMETERS

b expects one of two possible values:

TRUE enables drawing of colors into the z-buffer.
FALSE disables drawing of colors into the z-buffer.

DESCRIPTION

When zbuffer is TRUE, depth values are drawn into the z-buffer as a side effect of drawing to the front or back bitplane buffers. When zbuffer is FALSE, however, it is possible to treat the z-buffer as a third color bitplane buffer. zdraw enables or disables drawing of color values into the z-buffer.

By default, and after each call to gconfig, zdraw is FALSE. All combinations of values for backbuffer, frontbuffer, and zdraw are valid while the normal framebuffer is in double buffer mode. While the normal framebuffer is in single buffer mode, backbuffer is ignored, and frontbuffer can be disabled only while zdraw is enabled.

Because only the normal framebuffer includes a z-buffer, zdraw is significant only while the normal framebuffer is enabled for drawing (see drawmode). zdraw should not be called while drawing to the overlay, underlay, or pop-up framebuffers.

SEE ALSO

backbuffer, drawmode, frontbuffer, gconfig, zbuffer
NOTE

On the Personal Iris, calling zdraw(TRUE) selects the z-buffer as the
destination of the pixel writing routines: writepixels, writergb,
rectwrite, lrectwrite, and rectcopy. Geometric drawing routines (lines,
polygons, etc.) cannot draw into the z-buffer even when zdraw is
enabled. These commands will continue to draw into the front and back
buffers (as selected) when zdraw is on.

On the Personal Iris, when zdraw is on it is not possible to write pixels
into the frame buffer, regardless of the settings of frontbuffer and
backbuffer.

On all machines, operation while both zdraw and zbuffer are TRUE is
undefined.

BUGS

On the Personal Iris, when zdraw is enabled, geometric drawing com-
mands (lines, polygons, etc.) will update depth values into the z-buffer.

IRIS-4D VGX models do not support zdraw while the normal frame-
buffer is configured with stencil bitplanes. (see stensize.)
NAME

zfunction – specifies the function used for z-buffer comparison by the current framebuffer

C SPECIFICATION

void zfunction(func)
long func;

PARAMETERS

func expects one of eight possible flags used when comparing z values. The available flags are:

ZF_NEVER, the z-buffer function never passes.
ZF_LESS, the z-buffer function passes if the incoming pixel z value is less than the z value stored in the z-buffer bitplanes.
ZF_EQUAL, the z-buffer function passes if the incoming pixel z value is equal to the z value stored in the z-buffer bitplanes.
ZF_LEQUAL, the z-buffer function passes if the incoming pixel z value is less than or equal to the z value stored in the z-buffer bitplanes. (This is the default value.)
ZF_GREATER, the z-buffer function passes if the incoming pixel z value is greater than the z value stored in the z-buffer bitplanes.
ZF_NOTEQUAL, the z-buffer function passes if the incoming pixel z value is not equal to the z value stored in the z-buffer bitplanes.
ZF_GEQUAL, the z-buffer function passes if the incoming pixel z value is greater than or equal to the z value stored in the z-buffer bitplanes.
ZF_ALWAYS, the z-buffer function always passes.

DESCRIPTION

zfunction specifies the function used to compare each incoming pixel z value with the z value present in the z-buffer bitplanes. For example, if func is ZF_LESS and the incoming pixel z value is less than the z value
in the z-buffer bitplanes, the comparison passes. Refer to the zbuffer manual page for an explanation of z-buffer operation in the cases of z function pass and failure.

A separate zfunction mode is retained by each of the framebuffers: normal, popup, overlay, and underlay. The current draw mode determines which z function value is used, and which is modified by zfunction.

SEE ALSO
drawmode, zbuffer, zsource

NOTES
This subroutine is available only in immediate mode.

Currently z-buffer operation is supported only in the normal framebuffer. To insure compatibility with future releases of the GL, make calls to zfunction only while draw mode is NORMALDRAW.

On the Personal Iris, if you use zfunction with czelear you can increase the speed of buffer clearing.
NAME

zsourceme selects the source for z-buffering comparisons

C SPECIFICATION

void zsource(src)
long src;

PARAMETERS

src expects one of two possible values:

ZSRC_DEPTH, z-buffering is done by depth comparison
    (default).

ZSRC_COLOR, z-buffering is done by color comparison.

DESCRIPTION

By default z-buffer comparisons are done on depth data. However, in
certain cases, it can be useful to z-buffer by comparing color values,
especially the color index values generated by the linesmooth and
pntsmooth hardware. When the src parameter is ZSRC_DEPTH, the z-
buffer operation is normal. When the src parameter is ZSRC_COLOR,
however, source and destination color values are compared to determine
which pixels the system draws. In this mode, the zbuffer is not updated
when a pixel is written.

A separate zsource mode is retained by each of the framebuffers: nor-
mal, popup, overlay, and underlay. The current draw mode determines
which zsource mode is used, and which is modified by zsource.

SEE ALSO

drawmode, gversion, linesmooth, pntsmooth, zbuffer, zfunction

NOTES

This subroutine is available only in immediate mode.
This subroutine does not function on IRIS-4D B or G models.

Currently z-buffer operation is supported only in the normal frame-buffer. To insure compatibility with future releases of the GL, make calls to zsource only while draw mode is NORMALDRAW.

BUGS

IRIS-4D GT and GTX models support zsource(ZSRC_COLOR) only for non-subpixel positioned lines drawn after a linesmooth(SML_ON) call.

On early serial numbers of the Personal Iris, ZSRC_DEPTH is the only supported setting for this routine. For compatibility, they accept the call zsource(ZSRC_COLOR), but it has the same effect as calling zfunction(ZF_ALWAYS), which turns off z value comparison. This allows the unrestricted drawing of color values into the front and back buffers and depth values into the z-buffer. Use gversion to determine which type of Personal Iris you have.

IRIS-4D VGX models support zsource(ZSRC_COLOR) only in color map mode. Stencil operation is undefined in this case.

This routine is of limited utility, and we do not recommend the use of it.
NAME

**zwritemask** – specifies a write mask for the z-buffer of the current framebuffer

C SPECIFICATION

```c
void zwritemask(mask)
unsigned long mask;
```

PARAMETERS

*mask* expects a mask indicating which z-buffer bitplanes are read only and which can be written to. Z-buffer bitplanes that correspond to zeros in the mask are read only. Z-buffer bitplanes that correspond to ones in the mask can be written.

DESCRIPTION

*zwritemask* specifies a mask used to control which z-buffer bitplanes are written, and which are read only. A separate mask is maintained by each of the framebuffers, normal, popup, overlay, and underlay. The mask affects both writes to the z-buffer that are the result of z-buffer pixel operation, and writes resulting from *zclear* operation.

*zwritemask* is ignored while drawing directly to the z-buffer, as when *zdraw* is TRUE. In this case the current writemask applies to the z-buffer as well as to the color bitplanes.

SEE ALSO

wm_pack, writemask, zbuffer, zdraw

NOTES

This subroutine is available only in immediate mode.

Currently z-buffer operation is supported only in the normal framebuffer. To insure compatibility with future releases of the GL, make calls to *zwritemask* only while draw mode is **NORMALDRAW**.
BUGS

This subroutine does not function on IRIS-4D B or G models.

IRIS-4D GT and GTX models, and the Personal Iris, currently support a subset of z write mask functionality. Specifically, zwritemask either enables or disables the writing of all z-buffer bitplanes. The mask passed to zwritemask will disable z-buffer writes if it is zero; otherwise all z-buffer bitplanes are written. To assure upward compatibility with the IRIS-4D VGX and other future models, call zwritemask with mask equal to either 0 or 0xFFFFFFFF when all-or-nothing update is desired.
<table>
<thead>
<tr>
<th>Key Word</th>
<th>Man Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGL client buffer</td>
<td>gflush</td>
</tr>
<tr>
<td>DGL connection to a graphics</td>
<td>dglopen</td>
</tr>
<tr>
<td>DGL server connection</td>
<td>dgclose</td>
</tr>
<tr>
<td>GL framebuffer is drawable</td>
<td>drawmode</td>
</tr>
<tr>
<td>Geometry Pipeline is empty</td>
<td>finish</td>
</tr>
<tr>
<td>Geometry Pipeline -</td>
<td>passthrough</td>
</tr>
<tr>
<td>NURBS surface definition</td>
<td>bgnsurface</td>
</tr>
<tr>
<td>NURBS surface definition</td>
<td>endsurface</td>
</tr>
<tr>
<td>NURBS surface</td>
<td>hurbssurface</td>
</tr>
<tr>
<td>NURBS surface trimming loop</td>
<td>bgntrim</td>
</tr>
<tr>
<td>NURBS surface trimming loop</td>
<td>endtrim</td>
</tr>
<tr>
<td>NURBS surfaces display property</td>
<td>getnurbsproperty</td>
</tr>
<tr>
<td>NURBS surfaces /a piecewise</td>
<td>pwlcurve</td>
</tr>
<tr>
<td>NURBS surfaces - sets a property</td>
<td>setnurbsproperty</td>
</tr>
<tr>
<td>NURBS trimming curve</td>
<td>hurbscurve</td>
</tr>
<tr>
<td>Pipeline is empty</td>
<td>finish</td>
</tr>
<tr>
<td>RGB color values</td>
<td>gRGBcolor</td>
</tr>
<tr>
<td>RGB colors used for depth-cueing</td>
<td>lRGBrander</td>
</tr>
<tr>
<td>RGB mode</td>
<td>RGBcolor</td>
</tr>
<tr>
<td>RGB (or RGBA) values for the/</td>
<td>c3f</td>
</tr>
<tr>
<td>RGB values for a color map entry</td>
<td>getmcolor</td>
</tr>
<tr>
<td>RGB writemask</td>
<td>gRGBmask</td>
</tr>
<tr>
<td>accumulation buffer /of bitplanes</td>
<td>acbuf</td>
</tr>
<tr>
<td>accumulation buffer</td>
<td>acsize</td>
</tr>
<tr>
<td>all bitplanes</td>
<td>singlebuffer</td>
</tr>
<tr>
<td>all geometry is clipped</td>
<td>cliplane</td>
</tr>
<tr>
<td>all other windows /the current</td>
<td>winpop</td>
</tr>
<tr>
<td>all other windows - places the</td>
<td>winpush</td>
</tr>
<tr>
<td>allocates an area of the window</td>
<td>viewport</td>
</tr>
<tr>
<td>allocates and initializes a</td>
<td>newpup</td>
</tr>
<tr>
<td>allocates bitplanes for display</td>
<td>overlay</td>
</tr>
<tr>
<td>allocates bitplanes for display</td>
<td>underlay</td>
</tr>
<tr>
<td>alpha test function</td>
<td>afuction</td>
</tr>
<tr>
<td>antialiasing of lines</td>
<td>linesmooth</td>
</tr>
<tr>
<td>antialiasing of points</td>
<td>ptsmooth</td>
</tr>
<tr>
<td>antialiasing of polygons</td>
<td>polynomials</td>
</tr>
<tr>
<td>arc, arci, arcs - draw a circular</td>
<td>arc</td>
</tr>
<tr>
<td>arc, arci, arcs - draw a filled circular</td>
<td>arcf, arcf, arci, arci, arcs -</td>
</tr>
<tr>
<td>arc, arci, arcs - draw a filled circular</td>
<td>draw a circular arci, arci, arcs -</td>
</tr>
<tr>
<td>arc, arci, arcs - draw a filled circular</td>
<td>draw a circular arci, arci, arcs -</td>
</tr>
<tr>
<td>- specifies the aspect ratio of a graphics</td>
<td>keepaspect</td>
</tr>
<tr>
<td>fog density for per-vertex</td>
<td>fogvertex</td>
</tr>
<tr>
<td>atmospheric effects - specify</td>
<td>fogvertex</td>
</tr>
<tr>
<td>attaches the cursor to two</td>
<td>attachcursor</td>
</tr>
<tr>
<td>attaches the input focus to a</td>
<td>scrmattach</td>
</tr>
<tr>
<td>- pops the attribute stack</td>
<td>popattributes</td>
</tr>
<tr>
<td>- pushes down the attribute stack</td>
<td>pushattributes</td>
</tr>
<tr>
<td>off - turns backfacing polygon removal on and</td>
<td>backface</td>
</tr>
<tr>
<td>- returns whether backfacing polygons will appear</td>
<td>getbackface</td>
</tr>
<tr>
<td>process from being put into the background</td>
<td>prevents a graphical</td>
</tr>
</tbody>
</table>
- sets the color of the textport
- registers the screen
- sets current
- defines a
- selects a
bounding box and minimum/ bbox2i,
- rings the keyboard
of the beep of the keyboard
current window -
access to a subset of available
- clears the color
colors - allocates
colors - allocates
- returns the number of available
the/ - specify the number of
- writes and displays all
planes - specify the number of
- grants write permission to
- controls screen
- sets the screen
- specifies a window without any
/bbox2s - culs and prunes to
back the current computed screen
/- culs and prunes to bounding
current computed screen bounding
- control the screen
the lights on the dial and button
- sets the dial and button
- operate on the accumulation component in the accumulation and disable drawing to the back and disable drawing to the front
- flushes the DGL client array of pixels into the frame
- sets the display mode to double array of pixels into the frame
- defines a minimum time between
- indicates which
- exchanges the front and back
- sets the lights on the dial and
- sets the dial and
- returns the state of a
ties two valuators to a values for/ c3i, c3s, c4f, c4i,
- returns the
cmov2s - returns the current
cmov2s - updates the current
- returns the current
- returns the cursor
- returns the character
menu entry - sets the display
background ........................................ pagecolor
background process ................................ imakebackground
basis matrices ..................................... patchbasis
basis matrix ....................................... defbasis
basis matrix used to draw curves .......... curvebasis
bbox2s - culs and prunes to ............. bbox2,
bbox2s - culs and prunes to ............. bbox2,
bell ............................................. ringbell
bell - sets the duration ................. setbell
binds window constraints to the ........ winconstraints
bitplanes - grants write ................. RGBwritemask
bitplanes and the z-buffer/ ............ czeqler
bitplanes for display of overlay ........ overlay
bitplanes for display of underlay ......... underlay
bitplanes ..................................... getplanes
bitplanes per color component in ........ acsize
bitplanes ..................................... singlebuffer
bitplanes to be used as stencil ........... stensize
bitplanes ..................................... writemask
blanking ..................................... blankscreen
blanking timeout ......................... blanktime
borders ......................................... noborder
bounding box and minimum pixel/ ....... bbox2,
bounding box - read ..................... getscrbox
box and minimum pixel radius .......... bbox2,
box - read back the ................. getscrbox
box ........................................... scrbox
box - sets ............................ setblights
box text display ......................... dbox
buffer ......................................... scbuf
buffer /of bitplanes per color ........ acsize
buffer - enable ......................... backbuffer
buffer - enable ......................... frontbuffer
gflush ........................................ gflush
buffer - draws a rectangular ........... lrectwrite
buffer mode ............................ doublebuffer
buffer - draws a rectangular ........... rectwrite
buffers are enabled for writing ........ getbuffer
buffers of the normal framebuffer ...... swapbuffers
button box ................................ setblights
button box text display .................. dbox
text button ................................ getbutton
tie ........................................... tie
c4s - sets the RGB (or RGBA) .......... c3f,
character characteristics ................ getdescender
color height in the current ............ getheight
color height in the current ............ getheight
color position /cmov2i, ............... cmov,
character position ........................ getepos
characteristics ............................ getdescender
characteristics ............................ getdescender
characteristics of a given pop up ........ setupump
- sets the cursor characteristics .................................. setcursor
- draws a string of raster characters on the screen .................. charstr
queue - checks the contents of the event .......................... qtest
circf - circes - draws a filled circle ............................ circe,
circe, circes - outlines a circle ................................ circef,
circe, circefs - draws a filled circle ................................ circe,
circe, specified value - clear the stencil planes to a ................. sclear
the z-buffer simultaneously - clears the color bitplanes and ........ czclear
- clears the viewport ............................................. clear
- flushes the DGL against which all geometry is .................. gflush
window -
cmovi, cmovs, cmov2, cmov2i, cmov2s - updates the current/ .... cmov,
simultaneously - clears the color bitplanes and the z-buffer ........ czclear
active - change the effect of color commands while lighting is ... lmcolor
the number of bitplanes per color component in the/ - specify .... acsize
- returns the current color ........................................ getcolor
depth-cueing - sets range of color in RGB mode .................. RGBcolor
display mode that bypasses the color index in the current draw ....... color,
maps - organizes the color indices used for ........................... Ishaderange
- organizes the color map - sets a rendering and ................. RGBmode
rate - changes a color map as a number smaller ................. multimap
a copy of the RGB values for a color map as one large map .......... onemap
- changes a color map entry at a selectable ....................... blink
returns the number of the current color map entry - gets ............ getncolor
do not change a color map entry .................................. mapcolor
mode - sets the color map mode as the current ...................... cmode
- sets the color map mode ........................................ getcmode
- returns the current color map ramp for gamma .................... gammaramp
correction - defines a color maps at a specified rate .............. cyclemap
- cycles between color maps provided by multimap .................. setmap
- selects one of the small color of the text in the textport ........ textcolor
do not change a color value for a pixel .......................... blendfunction
do not change a color values ..................................... gRGBcolor
or RGBA) values for the current color values for a pixel .......... c3f,
integer - specifies RGBA color with a single packed 32-bit .......... cpack
do not change a color - sets the color index in .................... color,
the current draw mode object - compacts the memory storage of an .... compactify
do not change a color - controls compatibility modes ............... glcompat
do not change a color - allows the system to draw concave polygons .......... concave
into a texture - convert a 2-dimensional image ...................... texdef2d
screen into a line in 3-D world coordinates /maps a point on the ...... mapw
on the screen into 2-D world coordinates - maps a point ........... mapw2
the viewer's position in polar coordinates - defines ............... polarview

Version 4.0 - 3 - April 1990
world space to absolute screen coordinates - map
automatic generation of texture coordinates - specify
an optional zoom - texgen
copies a rectangle of pixels with rectcopy
copies and writes - specifies rectzoom
copy of a transformation matrix getmatrix
copy of the RGB values for a getmcolor
copy of the dimensions of the getviewport
correction - defines gammaramp
create a window that occupies the gbegi
create a window that occupies the ginit
creates a graphics subwindow swinopen
creates a graphics window winopen
creates a new tag within an newtag
creates an event queue entry qenter
creates an object makeobj
culls and prunes to bounding box bbox2,
current RGB color values gRGBcolor
current RGBA color values gRGBmask
current basis matrices patchbasis
current character position cmov, current character position getcpos
current color getcolor
current color in RGB mode RGBcolor
current color map getmap
returns the number of the current color map mode getcmmode
the RGB (or RGBA) values for the current color vector /c4s - sets c3f,
box - read back the current computed screen bounding getscrbox
returns the current display mode getdisplaymode
returns the current display monitor getmonitor
sets the type of the current draw mode color getcolor
sets the current drawing mode getdrawmode
disable z-buffer operation in the current framebuffer - enable or zbuffer
for z-buffer comparison by the current framebuffer / used zclear
mask for the z-buffer of the current framebuffer / a write zwritemask
initialize the z-buffer of the current graphics position getgps
z-buffer comparison by the current graphics position to a/ move,
for z-buffer comparison by the current graphics window behind winpush
current graphics window by its current graphics window windowmove
current graphics window. icontitle
of all other windows - moves the current graphics window in front winpop
current graphics window /sets the reshapeviewport
current graphics window current graphics window / changes winposition
of all other windows - moves the current graphics window windowget
current graphics window / changes current graphics window winset
viewport to the dimensions of the current graphics window wintitle
do not hallucinate.
current hitcode gethitcode
current linestyle current linestyle
sets a repeat factor for the current linestyle I is repeat
- multiplies a point by the current linewidth getwidth
coordinates - map current matrix in feedback mode xfpt,
- returns the current matrix mode ....................... getmmode
- sets the current matrix mode ....................... mmnmode
- sets color map mode as the current mode .......... cmode
- deletes a tag from the current object .............. deltag
whether a tag exists in the current object .......... istag
- returns the index of the maximum character height in the current object .......... getheight
- returns the current raster font number .......... getfont
- returns the current screen mask .................. getscmask
- returns the current shading model ............... getsm
- returns the current state of a valuator .......... getvaluator
- has no function in the current system ............. getlbackup
- premultiplies the current transformation matrix .... multmatrix
- returns the current value of a trimmed NURBS ...... getnurbsproperty
- returns the current viewport - gets ............... getviewport
returns the screen upon which the current window appears - ............ getscwm
- binds window constraints to the current window .... winconstraints
- returns the current writemask ..................... getwritemask
- returns the cursor characteristics ............... getcursor
- sets the cursor characteristics ................ setcursor
- sets the origin of a cursor ....................... curorigin
- defines the type and/or size of a cursor .......... curtype
- defines a cursor glyph ............................ defcursor
- attaches the cursor to two valuators .............. attachcursor
curoff - control draws a curve ....................... crv
/a piecewise linear trimming draws a curve for NURBS surfaces .............. pwlcurve
of line segments used to draw a curve - controls nurbcurve Nunes
- draws a curve segment ............................. curveit
curve segment - sets number ....................... curveprecision
draws a curve segment ............................. crv
- draws a series of curve segments .................. rcrv
draws a series of curve segments .................. rcrv
- deallocates a menu ............................... freepup
defines a basis matrix ............................. defbasis
gamma correction - defines a color map ramp for ....... gammaramp
defines a cursor glyph ............................. defcursor
defines a linestyle ............................... definestyle
defines a menu .................................... defpup
buffer swaps - defines a minimum time between ......... swapinterval
transformation - defines a perspective projection ....... perspective
transformation - defines a perspective projection ....... window
clipping mask - defines a raster font .................. defrasterfont
environment - defines a rectangular screen .......... scrmask
defines a texture mapping .......................... tevdef
light source, or lighting/ - defines or modifies a material .......... Imdef
defines patterns ................................... defpattern
cursor - defines the type and/or size of .......... curtype
polar coordinates - defines the viewer’s position in ....... polarview
delimit a NURBS surface ................................... bgnsurface
- closes an object
- delimits a NURBS surface
- opens an object
  open object -
  - indicates whether
    - turns
    - sets the
      - depth range ...
      - Isetdepth
      - depth-cue mode is on or off ...
      - getdc
      - depth-cue mode on and off ...
      - depthc
      - depth-cueing - sets ...
      - LRGrange
      - depth-cueing - sets ...
      - shaderange
      - description ...
      - getdesc
      - descriptor of the event queue ...
      - qgetfd
      - dimensions of the current ...
      - reshapeviewport
      - dimensions of the current ...
      - getviewport
      - dimensions of the picking region ...
      - picksize
      - buffer - enable and
        - enable and buffer - enable and
        - current framebuffer - enable or
given pop up menu / - sets the
  - sets the
  - returns the file
  - sets the viewport to the
    viewport - gets a copy of the
    - sets the
      - buffer - enable and
        - enable and
        - current framebuffer - enable or
given pop up menu / - sets the
  - sets the
  - returns the file
  - sets the
  - returns the current
    - color map - sets a rendering and
      - mode - sets the
        - returns the type of the current
          - allocates bitplanes for
            - sets a property for the
              - allocates bitplanes for
                - value of a trimmed NURBS surfaces
    - writes and
      - menu -
        - sets the display mode to
          - arci, arcs -
            - number of line segments used to
              - arci, arcs -
                - rectangle sboxi, sboxfs -
                  - sboxi, sbox -
                    - allows the system to
                      - selects a basis matrix used to
                        - the color index in the current
                          - rrd2, rrd2i, rrd2s - relative
                            - rpdr2i, rpdr2s - relative polygon
drawi, draws, draw2, draw2i, draw2s -
circli, circls -
polfs, polf2i, polf2s, polf2s -
draws, draw2, draw2i, draw2s -
definition ................................................ closeobj
definition ................................................ endsurface
definition for editing ................................... editobj
deletes a tag from the current ..................... deltag
deletes an object ....................................... delobj
deletes routines from an object ..................... objdelete
depth range ........................................... Isetdepth
depth-cue mode is on or off ......................... getdc
depth-cue mode on and off ......................... depthc
depth-cueing - sets ................................. LRGrange
depth-cueing - sets ................................. shaderange
description ......................................... getdesc
descriptor of the event queue .................... qgetfd
dimensions of the current ......................... reshapeviewport
dimensions of the current ......................... getviewport
dimensions of the picking region ................ picksize
disable drawing to the back ....................... backbuffer
disable drawing to the front ...................... frontbuffer
disable z-buffer operation in the ............... zbuffer
display characteristics of a ...................... setup
display ............................................... dbtext
display lights ....................................... lampon,
display list ......................................... maketag
display list routines with new object ............ objreplace
display mode ........................................ getdisplaymode
display mode that bypasses the ................. RGBmode
display mode to double buffer .................... doublebuffer
display monitor ..................................... getmonitor
display of overlay colors ......................... overlay
display of trimmed NURBS surfaces .............. setnurbsproperty
display of underlay colors ....................... underlay
display property /the current ................... getnurbsproperty
displays all bitplanes ............................. singlebuffer
displays the specified pop-up .................... dopup
double buffer mode .................................. doublebuffer
draw a circular arc ................................ arc,
draw a filled circular arc ....................... arcf,
draw a curve segment - sets ...................... curveprecision
draw a filled screen-aligned .................... sboxf,
draw a screen-aligned rectangle .............. sbox,
draw concave polygons ............................ concave
draw curves ......................................... curvebasis
draw mode color - sets ............................ color,
draw mode colors - sets .......................... colors,
draw rdri, rds, .................................... rrd,
draw rpdr, rpdrs, rpdr2s ......................... rpdr,
draw2s - draws a line ............................. draw,
draw2s - draws a line ............................. draw,
draws a curve ...................................... crv
draws a curve segment ............................. curveit
draws a filled circle ............................. circf,
draws a filled polygon ............................ polf,
draws a filled polygon polf, .................... polf,

- ends full-screen mode endfullscrn
- defines a color map ramp for gamma correction gammaramp
specify a plane against which all - defines a cursor cliплrane
- exists glyph defcursor
version information - returns graphics gexit
a 2-D, 3-D, or 4-D vertex to the graphics hardware and library gversion
- gets the current graphics pipe /v4s - transfers v2d,
graphics position getpос
point /move2s - moves the current graphics position to a specified move,
graphics server dglopen
- opens a DGL connection to a graphics state greset
graphics subwindow swinopen
- resets
- creates a
- gets
windows - places the current
graphics system description getdesc
corner - moves the current
graphics window behind all other winpush
discrete/ - specifies that a
graphics window by its lower-left winmove
fudge values that are added to a
graphics window change size in stepunit
- returns the position of a
graphics window - specifies fudge
the icon title for the current
graphics window window
- returns the size of a other/ - moves the current
the icon title for the current
graphics window in front of all winpop
- specifies the aspect ratio of a
graphics window keepaspect
- specifies the maximum size of a graphics window maxsize
- specifies the minimum size of a
graphics window minsize
specifies the preferred size of a
graphics window - specifies the
- to the dimensions of the current
graphics window /the viewport reshapeviewport
- closes the identified
graphics window winclose
the identifier of the current
graphics window - returns winget
- creates a
graphics window winopen
size and position of the current
graphics window changes the
- sets the current winposition
graphics window
- adds a title bar to the current
information - returns graphics
- gets video
graphics window gversion
- set video
hardware registers getvideo
hardware registers setvideo
- returns the maximum character
go height in the current raster font getheight
text height in the current raster font
- returns the current getheight
code gheightcode
- sets the
+ code to zero clearheightcode
icon size of a window iconsize
graphics window - assigns the
icon title for the current icontitle
integer for use as an object genobj
- gets the identifier - returns a unique
object - returns the identifier of the current winget
- convert a 2-dimensional
texdef2d
- returns the an area of the window for an
image - allocates viewport
colorf - sets the color index in the current draw mode color,
- returns the index of the current pattern getpattern
menu - allocates and
initializes a structure for a new newpup
- initializes the name stack initnames
- initializes the textport textinit

Version 4.0 - 8 - April 1990
current framebuffer - initializes the z-buffer of the ............... zclear
sequence on an optional video/ - initiates a command transfer ................ videocmd
   a specified location - inserts routines in an object at ............... objinsert
      - rings the keyboard bell ................................ ringbell
the duration of the beep of the keyboard bell - sets ............... setbell
      elkoff - control keyboard click ................................ clkon,
lampoff - control the light source, or lighting model ...................... lmdef
   - selects a new material, light source, or lighting model .............. lmbind
defines or modifies a material, effect of color commands while lighting is active - change the ..................... lmcolor
    a new material, light source, or lighting model - selects .............. lmbind
    a material, light source, or lighting model /or modifies .............. lmdef
   - control the keyboard display lights lampoff ......................... lampon,
    - sets the lights on the dial and button box .......................... setblights
delimit the vertices of a closed line - ..................................... bgnclosedline
draw2, draw2i, draw2s - draws a line drawi, draws, ......................... draw,
delimit the vertices of a closed line - ..................................... endclosedline
delimit the vertices of a line line ........................................ endline
maps a point on the screen into a line in 3-D world coordinates - ........ mapw
curve segment - sets number of line segments used to draw a .............. curveprecision
surfaces - describes a piecewise linear trimming curve for NURBS ........... pwcurve
defines a linestyle .................................................. deflinestyle
   - returns the current linestyle ......................................... getlinestyle
   - returns the current linestyle ......................................... getlinestyle
   - returns the current linestyle ......................................... getlinestyle
   - returns the current linestyle ......................................... getlinestyle
   - returns the current linestyle ......................................... getlinestyle
numbers a routine in the display list ..................................... maketag
   - reads a list of evaluators at one time ............................... getdev
   - overwrites existing display list routines with new ones .............. objreplace
    - loads a name onto the name stack ................................. loadname
    - loads a transformation matrix ....................................... loadmatrix
delimit a NURBS surface trimming loop ........................................... bgntrim
delimit a NURBS surface trimming loop ........................................... endtrim
mode that bypasses the color map /sets a rendering and display .......... RGBmode
   - organizes the color map as a number of smaller maps .............. multimap
   - organizes the color map as one large map .............................. onemap
   - changes a color map entry at a selectable rate ....................... blink
    of the RGB values for a color map entry - gets a copy .............. getncolor
    - changes a color map entry ............................................ mapcolor
   the number of the current color map - returns .......................... getmap
   - sets color map mode as the current mode ............................. cmode
   - returns the current color map mode ................................. getcmode
   the color map as one large map - organizes ............................. onemap
   - defines a color map ramp for gamma correction ....................... gammaramp
screen coordinates - map world space to absolute ....................... screenspace
   - defines a texture mapping environment ................................ tevdef
current/ - specifies a write mask for the z-buffer of the ............... zwritemask
getscrmask
- defines scrmask
matrices patchbasis
defbasis
matrix getmatrix
matrix in feedback mode /xfp4s - xft
matrix getmode
matrix mode
matrix mode multimatrix
matrix - premultiplies popmatrix
matrix stack pushmatrix
matrix used to draw curves curvebasis
memory - chunksize
memory - reads a rectangular Irectread
memory - reads a rectangular rectread
memory storage of an object compactify
menu addtopup
defpup
menu dopup
menu entry - sets the display setupup
menu freepup
menu - allocates and newpup
mesh - delimit bgntmesh
mesh - delimit endmesh
mesh register pointer swaptmesh
mirrors objects scale
mode RGBcolor
mode as the current mode. cmode
mode colorf - sets the color,
mode - sets doublebuffer
mode endfeedback
mode endfullscrn
mode endpick
mode endselect
mode feedback
mode getcmmode
getdisplaymode
getdrawmode
mode getmmode
getresetsls
mode gselect
mode is on or off getdcm
mode mmode
mode on and off depthcuel
mode parameters pixmode
mode pick
mode - selects one of the small setmap
mode that bypasses the color map RGBmode
mode to double buffer mode doublebuffer
mode /xfp4s multiplies a point xft,
the type of the current display
- sets the
  
  mv2, mv2i, mv2s - relative
  rpv2i, rpv2s - relative polygon
  movei, moves, move2, move2i,
  /moves, move2, move2i, move2s -
  by its lower-left corner -
  in front of all other windows -
  the small color maps provided by
  - pops a
  - pushes a new
  - loads a
  - initializes the
  - loads a name onto the
  - pops a name off the
  - pushes a new name on the
  the front and back buffers of the
  
  /- specifies a
  - specify renormalization of
  - inserts routines in an
    - calls a function from within an
      - draws an instance of an
        compacts the memory storage of an
        - closes an
        - opens an
        - deletes an
        a tag from the current open
        - returns whether an
          identifier of the currently open
          a unique integer for use as an
            a tag exists in the current open
            - creates an
            - deletes routines from an
              - creates a new tag within an
                - specifies minimum
  
  monitor - returns ................................. getmonitor
  monitor type .................................... setmonitor
  move rmvi, rmvs, .............................. mv,
  move rpvni, rpvns, rpvv2, .................... rpv,
  move2s - moves the current/ ........................ move,
  moves the current graphics/ ........................ move,
  moves the current graphics window ............ winmove
  moves the current graphics window ............ winpop
  multimap mode - selects one of .................. setmap
  name off the name stack ........................ popname
  name on the name stack ........................ pushname
  name onto the name stack ........................ loadname
  name stack ..................................... initnames
  name stack ..................................... loadname
  name stack ..................................... popname
  name stack ..................................... pushname
  normal framebuffer - exchanges .................. swapbuffers
  
  normal ....................................... n3f
  normals ....................................... nmode
  object at a specified location .................. objinsert
  object .......................................... callfunc
  object .......................................... callobj
  object - ....................................... compactify
  object definition ................................ closeobj
  object definition for editing .................... editobj
  object .......................................... delobj
  object - deletes ............................... deltag
  object exists ................................... isobj
  object - returns the ............................ getopenobj
  object identifier - returns .................... genobj
  object - returns whether ....................... istag
  object .......................................... makeobj
  object .......................................... objdelete
  object relative to an existing/ .................. newtag
  object size in memory ........................... chunksize
  - obsolete routine ............................ RGBcursor
  - obsolete routine ............................ RGBrange
  - obsolete routine ............................ endpupmode
  - obsolete routine ............................ gRGBcursor
  - obsolete routine ............................ getdepth
  - obsolete routine ............................ getothermonitor
  - obsolete routine ............................ getport
  - obsolete routine ............................ getshade
  - obsolete routine ............................ ismex
  - obsolete routine ............................ normal
  - obsolete routine ............................ pupmode
  - obsolete routine ............................ setdepth
  - obsolete routine ............................ setsdepth
  - obsolete routine ............................ shadersrange
  - obsolete routine ............................ smoothline
  - obsolete routine ............................ spclos
backfacing polygon removal on and
    - turns depth-cue mode on and
        polygon removal on and
        whether depth-cue mode is on or
        whether z-buffering is on or
            - turns
                - pops a name
            /a command transfer sequence
    turns backfacing polygon removal
        - turns depth-cue mode
            on and off
        turns frontfacing polygon removal
            whether depth-cue mode is
                - returns whether z-buffering is
                    - operate
                        - sets the lights
                    - pushes a new name
                    a string of raster characters
            coordinates - maps a point
                world coordinates - maps a point
                    - paints a row of pixels
                    - paints a row of pixels
                        graphics server -
                            editing -
                            number of smaller maps -
                            large map -
                            projection transformation
                                ortho2 - define an orthographic
                                    orthographic projection/
                                outlines a circle
                                    outlines a polygon polyi,
                                    outlines a rectangular region
                                bitplanes for display of
                                    overlay colors - allocates
                                    overwrites existing display list
                                    screens with new ones -
                                    screen -
                                    screen -
                                    - alter the operating
                                    - specify pixel transfer mode
                                        - draws a surface
                                            of curves used to represent a
                                                at which curves are drawn in a
                                                    - draws a rational surface
                                                        rectangles - selects a
                                                            returns the index of the current
                                                                - selects a linestyle
                                                                    - defines
                                                                        of a/ pdfi, pdrs, pdrl2, pdr2i,
                                                                        sequence on an optional video
                                                                        transformation - defines a
                                                                            transformation - defines a
                                                                                for NURBS surfaces - describes a
- obsolete routine .............................................. winattach
- turns ......................................................... backface
- turns ......................................................... backfacing
turns frontfacing ............................................ frontface
- indicates ..................................................... getdcn
- returns ....................................................... getbuffer
- picking mode ................................................... endpick
- selecting mode ............................................... endselect
- off the name stack ........................................ popname
- on an optional video peripheral .......................... videocmd
- on and off ................................................... backface
- on and off ................................................... depthcue
- on and off ................................................... frontface
- on or off ..................................................... getdcn
- on or off ..................................................... getbuffer
- on the accumulation buffer ................................ acbuf
- on the dial and button box ................................. setdblights
- on the name stack ........................................... pushname
- on the screen - draws ...................................... charstr
- on the screen into 2-D world ............................ mapw2
- on the screen into a line in 3-D ....................... mapw
- on the screen ............................................... writeRGB
- on the screen ............................................... writepixels
- opens a DGL connection to a ............................ dglopen
- opens an object definition for .......................... editobj
- organizes the color map as a ............................ multimap
- organizes the color map as one ........................... onemap
- ortho2 - define an orthographic
    orthographic projection/
    outlines a circle
    outlines a polygon polyi,
    outlines a rectangular region
    overlay colors - allocates
    overwrites existing display list
    paints a row of pixels on the
    paints a row of pixels on the
    parameters of the stencil
    parameters
    patch ....................................................... patch
    patch - sets the number ................................ patchcurves
    patch - sets the precision ............................. patchprecision
    pattern ..................................................... setpattern
    pattern ..................................................... getpattern
    pattern ..................................................... setlinestyle
    patterns .................................................... defpattern
    pdr2s - specifies the next point ........................ pdr,
    peripheral /a command transfer ........................ videocmd
    perspective projection ................................... perspective
    perspective projection ................................. window
    piecewise linear trimming curve ..................... pwlcuvr
a blended color value for a
/the zoom for rectangular
to bounding box and minimum
- specify
specifies a logical operation for	polygon vertices - controls the
is clipped - specify a
bitplanes to be used as stencil
- clear the stencil
of a/ pmvi, pmvs, pmv2, pmv2i,
pnti, pnts, pnt2, pnt2i,
/xftp4i, xftp4s - multiplies a
graphics position to a specified
pdr2i, pdr2s - specifies the next
pmv2s - specifies the first
world coordinates - maps a
in 3-D world/ - maps a
pnt2, pnt2i, pnt2s - draws a
of vertex routines as
of vertex routines as
- specify antialiasing of
defines the viewer’s position in
polfi, polfs, polf2, polf2i,
polyi, polys, poly2, poly2i,
- delimit the vertices of a
rdrl2, rdrl2i, rdrl2s - relative
- delimit the vertices of a
rpmv2, rpmv2i, rpmv2s - relative
- closes a filled
- specifies the next point of a
- specifies the first point of a
polf2i, polf2s - draws a filled
poly2i, poly2s - outlines a
- turns backfacing
- turns frontfacing
splf2s - draws a shaded filled
the placement of point, line, and
pixel - computes .................................. blendfunction
pixel copies and writes ................................ rectzoom
pixel radius / - call and prunes ................. box2,
pixel transfer mode parameters ....................... pixmode
pixel writes - ....................................... logicop
placement of point, line, and ....................... subpixel
plane against which all geometry ............... clipplane
planes - specify the number of ....................... stensize
planes to a specified value ......................... sclear
pmv2s - specifies the first point ................. pmvi,
pnt2s - draws a point ................................ pnt,
point by the current matrix in/ .................... xftp,
point /move2s - moves the current ............... move,
point of a polygon /pdrs, pdr2, ............... pdr,
point of a polygon /pmv2, pmv2i, ............... pmvi,
point on the screen into 2-D ....................... mapw2
point on the screen into a line ................... mapw
point pnti, pnts, ................................. pnt,
points /the interpretation ......................... bgnpoint
points /the interpretation ......................... endpoint
points ............................................. pnts
smooth polar coordinates - ......................... polarview
polf2s - draws a filled polygon ................... polf,
poly2s - outlines a polygon ....................... poly,
polygon ............................................ bgnpolygon
polygon draw rdrl, rdrls, ......................... rdrl,
polygon endpolygon ................................ endpolygon
polygon move rpmvi, rpmvs, ...................... rpmvi,
polygon ............................................ polcs
polygon /pdrs, pdr2, pdr2i, pdr2s .............. pdr,
polygon /pmvsi, pmvsi, pmv2i, pmv2s ........ pmvi,
polygon polfi, polfs, polf2, ...................... polf,
polygon polyi, polys, poly2, ...................... poly,
polygon removal on and off ....................... backface
polygon removal on and off ....................... frontface
polygon /splfis, splf2, splf2i, .................. spfl,
polygon vertices - controls ....................... subpixel
- pops a name off the name stack ............... popname
- pops the attribute stack ....................... popattrib
stack -
- pops the transformation matrix ............... popmatrix
- pops the viewport stack ....................... popviewport
- pop-up menu .................................. addtopop
- display the specified
- updates the current character ..................
- returns the current character ................. getcpos
- gets the current graphics ..................... getcpos
- defines the viewer’s .......................... winposition
- returns the window - changes the size and
- moves the current graphics ..................
rot - rotate graphical
- translates graphical
- prevents a graphical
  - registers the screen background
orthogonal2 - define an orthogonal
- defines a perspective
- defines a perspective
trimmed NURBS surfaces - sets a
  a trimmed NURBS surfaces display
bbox2i, bbox2s - calls and
  stack -
  -
  matrix stack -
  -
  - reads multiple entries from the
  - creates an event
  - administers event
  the file descriptor of the event
  the first entry in the event
  - empties the event
  checks the contents of the event
  from making entries in the event
  specified device is enabled for
to bounding box and minimum pixel
  - sets the depth
  depth-cueing - sets the
  depth-cueing - sets
  - assigns an initial value and a
  - draws a string of
  - defines a
  strings - selects a
  character height in the current
  - returns the current
  a color map entry at a selectable
  between color maps at a specified
  - specifies the aspect
  - draws a
  - draws a
  rdri, rdrs, rdr2, rdr2i,
screen bounding box -
for pixels that various routines
optional zoom - copies a
sboxs - draw a screen-aligned
- draw a filled screen-aligned
rectfi, rectfs - fills a
CPU memory - reads a
CPU memory - reads a
the frame buffer - draws a
the frame buffer - draws a
writes - specifies the zoom for
recti, rect - outlines a
primitives ........................................... rotate,
primitives ........................................... translate
process from being put into the / foreground
process ................................................... imakebackground
projection transformation ................................ ortho,
projection transformation ................................ perspective
projection transformation ................................ window
property for the display of ................................ setnurbsproperty
property /the current value of ................................ getnurbsproperty
prunes to bounding box and/ ................................bbox2,
pushes a new name on the name ................................ pushname
pushes down the attribute stack ................................ pushattributes
pushes down the transformation ................................ pushmatrix
pushes down the viewport stack ................................ pushviewport
queue ................................................. blkqread
queue entry ......................................... qenter
queue .................................................. qcontrol
queue - returns ...................................... qgetfd
queue - reads ........................................ qread
queue .................................................. qreset
queue .................................................. qtest
queue /the specified device ................................ unqdevice
queueing - returns whether the ................................ isqueued
radius /bbox2s - calls and prunes ................................ bbox2,
rangle ............................................... lsetdepth
range of RGB colors used for ................................ IRGBRange
range of color indices used for ................................ Ishaderrange
range to a evaluator ..................................... setevaluator
raster characters on the screen ................................ charstr
raster font ............................................. defrasterfont
raster font for drawing text ................................ font
raster font /returns the maximum ................................ getheight
raster font number ..................................... getfont
rate - changes .......................................... blink
rate - cycles .......................................... cyclemap
ratio of a graphics window ................................ keepsaspect
rational curve ......................................... rcrv
rational surface patch .................................. rpatch
rdр2s - relative draw ................................... rdr,
read back the current computed ................................ getscreenbox
read - sets the source ................................... readscreen
rectangle of pixels with an ................................ rectcopy
rectangle sboxi, ........................................ sbox,
rectangle sboxf, ........................................ sboxf,
rectangular area ....................................... rectf,
rectangular array of pixels into ................................ Irectread
rectangular array of pixels into ................................ Irectread
rectangular array of pixels into ................................ Irectwrite
rectangular array of pixels into ................................ Irectwrite
rectangular pixel copies and ................................ rectcopy
rectangular pixel copies and ................................ rectzoom
rectangular region ...................................... rect,
- defines a rectangular screen clipping mask .......... scrmask
  rectf, rects - fills a rectangular area .......... rectf,
  region recti, rects - outlines a rectangular .......... rect,
- toggles the triangle mesh swapmesh
- get video hardware getvideo
- set video hardware setvideo
- process
  rdri, rdrs, rdr2, rdr2i, rdr2s -
  mvvi, mvvs, mvv2, mvv2i, mvv2s -
  rpdr, rpdr2, rpdr2i, rpdr2s -
  rpmmv, rpmmv2, rpmmv2i, rpmmv2s -
  /a new tag within an object
  bypasses the color map - sets a
  - control the
- returns the state of linestyle
- waits for a vertical
  mvvi, mvv2, mvv2i, mvv2s -
  rpdr, rpdr2, rpdr2i, rpdr2s -
  rpmmv, rpmmv2, rpmmv2i, rpmmv2s -
  screen-aligned rectangle sboxf,
  rectangle sboxi,
- registers the
  - controls
  - sets the
- read back the current computed
  - control the
  of raster characters on the
- defines a rectangular
- map world space to absolute
- a program write to the entire
  a window that occupies the entire
  a window that occupies the entire
  - maps a point on the
  - maps a point on the
  - returns the current
  - attaches the input focus to a
  that a program does not need
  placed - selects the
  window appears - returns the
  - paints a row of pixels on the
  - paints a row of pixels on the
  - draws a series of curve ...
  - draws a series of curve ...
  segment - sets number of line
  - turns off
  - puts the system in
  draw curves
  - selects a linestyle pattern

Version 4.0 - 15 - April 1990
source, or lighting model - selects a new material, light .......... Imbind
polygons and rectangles - selects a pattern for filling .......... setpattern
text strings - selects a raster font for drawing .......... font
- selects a texture environment .......... tevbind
- selects a texture function .......... texbind
maps provided by multimap mode - selects one of the small color .......... setmap
windows are placed - selects the screen upon which new .......... scmsselect
- selects the shading model .......... shademodel
z-buffering comparisons - selects the source for .......... zsource
drawable - selects which GL framebuffer is .......... drawmode
- closes the DGL server connection .......... dlgclose
a DGL connection to a graphics server - opens .......... dglopen
of trimmed NURBS surfaces - sets a property for the display .......... setnurbsproperty
that bypasses the color map - sets a rendering and display mode .......... RGBmode
current linestyle - sets a repeat factor for the .......... lrepeat
current mode. - sets color map mode as the .......... cmode
- sets current basis matrices .......... patchbasis
to draw a curve segment - sets number of line segments used .......... curveprecision
for depth-cueing - sets range of color indices used .......... Ishaderrorange
current draw mode color - sets the RGB (or RGBA) values for .......... c3f,
textport - sets the color in the .......... color,
background - sets the color of the textport .......... pagecolor
mode - sets the current color in RGB .......... RGBcolor
- sets the current graphics window .......... winset
- sets the current matrix mode .......... mmode
- sets the cursor characteristics .......... setcursor
- sets the depth range .......... Isedepth
display - sets the dial and button box text .......... dbtext
picking region - sets the dimensions of the .......... picksize
of a given pop up menu entry - sets the display characteristics .......... setupup
buffer mode - sets the display mode to double .......... doublebuffer
the keyboard bell - sets the duration of the beep of .......... setbell
- sets the hitcode to zero .......... clearhitcode
button box - sets the lights on the dial and .......... set.dblights
- sets the monitor type .......... setmonitor
represent a patch - sets the number of curves used to .......... patchcures
- sets the origin of a cursor .......... curorigin
curves are drawn in a patch - sets the precision at which .......... patchprecision
for depth-cueing - sets the range of RGB colors used .......... IRGBrange
- sets the screen blanking timeout .......... blanktime
various routines read - sets the source for pixels that .......... readsource
dimensions of the current/ - sets the viewport to the .......... reshapeviewport
- returns the current shading model .......... getsm
- selects the shading model .......... shademodel
- specifies RGBA color with a single packed 32-bit integer .......... cpakc
- specifies RGBA writemask with a single packed integer .......... wpakc
Pipeline - passes a single token through the Geometry .......... passthrough
routines read - sets the source for pixels that various .......... readsource
comparisons - selects the source for z-buffering .......... zsource
splf, splfs, splf2, splf2i, splf2s - draws a shaded filled/ .......... splf,
- delimit the vertices of a line .................................... bglinline
- delimit the vertices of a line .................................... endl ine
- delimit the vertices of a polygon ................................. bgnpolygon
- delimit the vertices of a polygon ................................. endpolygon
- delimit the vertices of a quadrilateral strip ..................... bgnsstrip
- delimit the vertices of a quadrilateral strip ..................... endqstrip
- delimit the vertices of a triangle mesh .......................... bgntmesh
- delimit the vertices of a triangle mesh .......................... endlmesh

of point, line, and polygon
- defines a viewing transformation ............................... lookat
- clears the viewport ................................................... clear

of the dimensions of the current
viewport - gets a copy .............................................. getviewport
- pops the viewport stack ............................................. popviewport
- pushes down the viewport stack .................................... pushviewport
viewport to the dimensions of the ............................... reshapeviewport

window appears - returns ........................................... getwscrn
window behind all other windows ................................. winpush
window by its lower-left corner .................................. winmove
window change size in discrete/ ................................. stepunit
window constraints to the current ............................... winconstraints

window cursor off ..................................................... curson,
window for an image .................................................. viewport
window - specifies fudge values ................................... fudge
window - returns ....................................................... getorigin
window ................................................................. getsiz e
window ................................................................. iconsize
window in front of all other/ ......................................... winpop
window is in the window stack ...................................... wind eath
window - specifies ..................................................... keepaspect
window - specifies ..................................................... maxsize
window - specifies ..................................................... minsize
window - specifies the preferred ................................. preposition
window - specifies ..................................................... predim
window /to the dimensions ........................................... reshapeviewport

how deep a window is in the
screen - create a window ............................................. gbegin
screen - create a window ............................................. ginit

- closes the identified graphics
window constraints to the current
of the current graphics
- creates a graphics window.......................................... winopen
position of the current graphics
- sets the current graphics

- specifies a title bar to the current graphics
- specifies a on the screen into a line in 3-D
a point on the screen into 2-D
coordinates - map
available biplanes - grants
the current/ - specifies a
- grants

write access to a subset of ......................................... RGBwritemask
write mask for the z-buffer of .................................... zwritemask

Version 4.0 - 19 -

April 1990
- allows a program
- returns the current RGB
- returns the current integer - specifies RGBA
- a logical operation for pixel for rectangular pixel copies and which buffers are enabled for
/xfpt2i, xfpt2s, xfpt4, xfpt4i,
- specifies the function used for framebuffer - initializes the
- specifies a write mask for the framebuffer - enable or disable the color bitplanes and the or disables drawing to the
- selects the source for
- returns whether and writes - specifies the of pixels with an optional write to the entire screen ................. fullscrm
writemask .................................................. gRGBmask
writemask .................................................. getwritemask
writemask with a single packed .................. wmpack
writes and displays all biplanes ................ singlebuffer
writes - specifies ................................... logicop
writes - specifies the zoom ........................ rectzoom
writing - indicates ................................. getbuffer
xfpt4s - multiplies a point by/ .................. xfpt,
z-buffer comparison by the/ ...................... zfunction
z-buffer of the current ......................... zclear
z-buffer of the current/ .......................... zwritemask
z-buffer operation in the current .............. zbuffer
z-buffer simultaneously - clears ............. czclear
z-buffer - enables ................................. zdraw
z-buffering comparisons ....................... zsource
z-buffering is on or off ....................... getzbuffer
zoom for rectangular pixel copies ............ rectzoom
zoom - copies a rectangle ...................... rectcopy
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