GL3.2 for AIX: Graphics Library (GL) Technical Reference (POWER-based Systems only)

GL3.2 for AIX: Graphics Library (GL) Technical Reference (POWER-based Systems only)

First Edition (October 1994)

Before using the information in this book, read the general information in Notices.

This edition applies to the GL3.2 Version 4.1 for AIX licensed program and to all subsequent releases of this product until otherwise indicated in new editions.

Contents

About This Book

GL3.2 for AIX: Graphics Library (GL) Technical Reference (POWER-based Systems Only) provides information on calls and subroutines for the Graphics Library (GL) and example programs that illustrate using basic GL subroutines.

Who Should Use This Book

This book is intended for programmers who develop 3-D applications. To use this book effectively, you should be familiar enough with the C programming language to write, compile, and link a program that prints `Hello, World!` on the screen. You should also be familiar with the GL programming interface. Use this book as a reference companion to the *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* book.

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Related Publications

The following books contain information about or related to programming graphics:

- *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*
- *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*
- *AIX 5L Version 5.1 AIXwindows Programming Guide*
- *AIX 5L for POWER-based Systems OpenGL 2.1 Reference Manual*
- *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- AIXwindows
- POWERstation
- RS/6000
- SNA

Microsoft is a trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be the trademarks or service marks of others.

Chapter 1. GL Subroutines

This chapter describes the subroutines for GL. These subroutines are organized alphabetically.

addtopup Subroutine

Purpose

Adds items to an existing pop-up menu.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void addtopup
(Int32  popup,
Char8 *  string,
Int32  argument)
```

FORTRAN Syntax

SUBROUTINE ADDTOP(*popup*, *string*, *length*, *argument***)**

INTEGER*4 *popup*

CHARACTER*(*) *string***(*)**

INTEGER*4 *length*

INTEGER*4 *argument*

Description

The **addtopup** subroutine adds items to the bottom of an existing pop-up menu. You can build a menu by using a call to the **newpup** subroutine to create a menu, followed by a call to the **addtopup** subroutine for each menu item that you want to add to the menu.

To activate and display the menu, submit the menu to the **dopup** subroutine.

Note: This subroutine cannot be used to add to a display list.

Parameters

popup Specifies the identifier of the menu to which to add. The menu identifier is the returned function value of the menu creation call to either the **newpup** or **defpup** subroutines.

string Specifies the pointer to the text to add as a menu item. There are seven menu item type flags for optional pairing with each menu item:

%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** flag. You must specify the invoked routine in the *argument* 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 the **%F** flag. Produces unpredictable results in FORTRAN.

%f Invokes a routine when this particular menu item is selected. You must specify the invoked routine in the *argument* 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 the **%f** flag. If you have also used the **%F** flag within this menu, then the result of the **%f** flag is passed as a parameter of the **%F** flag. Produces unpredictable results in FORTRAN.

%l Adds a line under the current entry. This is useful in providing visual clues for grouping 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 *argument* parameter.

%n Like the **%f** flag, this flag invokes a routine when the user selects this menu item. However, the **%n** flag differs from the **%f** flag in that it ignores the routine (if any) specified by the **%F** flag. 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 the **%f** flag. Produces unpredictable results in FORTRAN.

%x*n* Assigns a numeric value to this menu item. This value overrides the default position-based value assigned to this menu item (third item=3). You must enter the numeric value as the part of the text string specified by the *n* flag. Do not use the *argument* parameter to specify the numeric value.

Notes:

1. With the | (vertical bar) delimiter, you can specify multiple menu items in a text string. However, because there is only one *argument* parameter, the text string can contain no more than one item type that references the *argument* parameter.
2. In FORTRAN programming language, there is no pointer to a function. This restriction causes the **%f**, **%n**, and **%F** flags to produce unpredictable results.

length Specifies the length of the text string (for FORTRAN syntax).

argument Specifies the command or submenu to assign to the menu item. There can be only one *argument* parameter for each call to the **addtopup** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a pop-up menu with the **defpup** subroutine.

Displaying a pop-up menu with the **dopup** subroutine.

Deallocating a pop-up menu and its data structures with the **freepup** subroutine.

Allocating and initializing a structure for a new pop-up menu with the **newpup** subroutine.

Enabling or disabling a given pop-up entry with the **setpup** subroutine.

Graphics Library Overview and Creating and Managing Pop-Up Menus.

---

arc Subroutine

Purpose

Draws a circular arc.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

FORTRAN Syntax

```
SUBROUTINE ARC(x, y, radius, startang, endang)
REAL x, y, radius
INTEGER*4 startang, endang
SUBROUTINE ARCI(x, y, radius, startang, endang)
INTEGER*4 x, y, radius, startang, endang
SUBROUTINE ARCS(x, y, radius, startang, endang)
INTEGER*2 x, y, radius, startang, endang
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference. Also, the FORTRAN INTEGER*2 version, the **ARCS** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked

by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

Description

The **arc** subroutine draws a circular arc in the x-y plane (z = 0), using the current line attributes: color, linestyle, linewidth and writemask. 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.

All of the routines listed in the syntax are functionally the same. They differ only in the type declarations for the coordinates. After the **arc** subroutine executes, the graphics position is left undefined.

The syntax for each of the subroutine forms is the same except for the first argument. They differ only in that **arc** expects real coordinates, **arci** expects integer coordinates, and **arcs** expects short integer coordinates.

Parameters

x	Specifies the *x* coordinate of the center of the arc, which is the center of the circle that would contain the arc.
y	Specifies the *y* coordinate of the center of the arc, which is the center of the circle that would contain the arc.
radius	Specifies the length of the radius of the arc, which is the radius of the circle that would contain the arc.
startang	Specifies the measure of the start angle of the arc, which is measured in tenths of a degree from the positive x-axis.
endang	Specifies the measure of the end angle of the arc, which is measured in tenths of a degree from the positive x-axis.

Example

The example C language program **tpbig.c** uses the **arc** subroutine and the **arci** subroutine to draw two circular arcs in the default linestyle.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a filled circular arc with the **arcf** subroutine.

Drawing a circle with the **circ** subroutine.

Drawing a filled circle with the **circf** subroutine.

Drawing a curve with the **crv** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Drawing Rectangles, Circles, Arcs, and Polygons.

arcf Subroutine

Purpose

Draws a filled, pie-slice-shaped, circular arc.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

FORTRAN Syntax

```
SUBROUTINE ARCF(x, y, radius, startang, endang)
REAL x, y, radius
INTEGER*4 startang, endang

SUBROUTINE ARCFI(x, y, radius, startang, endang)
INTEGER*4 x, y, radius, startang, endang

SUBROUTINE ARCFS(x, y, radius, startang, endang)
INTEGER*2 x, y, radius, startang, endang
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference. Also, the FORTRAN INTEGER*2 version, the **ARCFS** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

Description

The **arcf** subroutine draws a circular filled arc in the *x*-*y* plane (*z* = *0*) and fills the arc with the current pattern. The filled area is bounded by the arc and by the start and end radii. The subroutine uses the current area attributes: texture, color, and writemask. To draw a filled arc in a plane other than the *x*-*y* plane, define the filled arc in the *x*-*y* plane and then rotate or translate the arc.

The syntax for each of the subroutine forms is the same except for the first argument. They differ only in that the **arcf** subroutine expects real coordinates for the first argument, the **arcfi** subroutine expects integer coordinates, and the **arcfs** subroutine expects short integer coordinates. After the **arcf** subroutine executes, the graphics position is undefined.

Parameters

x	Specifies the *x* coordinate of the center of the filled arc, which is the center of the circle that would contain the arc.
y	Specifies the *y* coordinate of the center of the filled arc, which is the center of the circle that would contain the arc.
radius	Specifies the length of the radius of the filled arc, which is the radius of the circle that would contain the arc.
startang	Specifies the measure of the start angle of the filled arc, which is measured from the positive x-axis.
endang	Specifies the measure of the end angle of the filled arc, which is measured from the positive x-axis.

Example

The example C language program **tpbig.c** uses the **arcfi** subroutine to draw a scoop of ice cream.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a circular arc with the **arc** subroutine.

Drawing a circle with the **circ** subroutine.

Drawing a filled circle with the **circf** subroutine.

Drawing a curve with the **crv** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

attachcursor Subroutine

Purpose

Couples cursor position to valuator device.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

FORTRAN Syntax

SUBROUTINE ATTACH (*vx***, *vy***) INTEGER*4** *vx*, *vy*

Description

The **attachcursor** subroutine couples the pointer location to the indicated valuators. As the values generated by the valuators change, the cursor position is updated to reflect the new readings. The values are interpreted as screen coordinates (as measured in pixels) from the lower-left hand corner of the screen. Currently, the only devices supported by the **attachcursor** subroutine are the MOUSEX, MOUSEY, GHOSTX and GHOSTY devices. This subroutine cannot be used to add to a display list.

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when started from a FORTRAN program, although it accepts short integers when started from a C program. The C and FORTRAN syntax shown here reflect this difference.

Parameters

vx Token representing the valuator device that controls the horizontal location of the cursor. By default, the x position of the cursor is attached to the MOUSEX device.

vy Token representing the valuator device that controls the vertical location of the cursor. By default, the y position of the cursor is attached to the MOUSEY device.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Tying valuators to a button with the **tie** subroutine.

Filtering valuator motion with the **noise** subroutine.

Setting the minimum, maximum, and current value of a valuator with the **setvaluator** subroutine.

Graphics Library Overview, Creating a Cursor, and Creating and Managing Windows.

---

backbuffer Subroutine

Purpose

Enables or disables drawing into the back buffer.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void backbuffer(Int32 *bool***)**

FORTRAN Syntax

SUBROUTINE BACKBU(*bool***)**
LOGICAL *bool*

Description

The **backbuffer** subroutine enables or disables drawing into the back frame buffer. By default, drawing into the back frame buffer is enabled. In common usage, drawing is done to the back buffer, after which a call to the **swapbuffers** subroutine is made to exchange buffers. The **backbuffer** subroutine can be used to override this default.

This routine is useful only in double buffer mode, and is ignored in single buffer mode.

Parameter

bool Specifies a value for the state of the back frame buffer. The settings for the *bool* parameter are:
True = drawing into the back buffer is enabled.
False = drawing into the back buffer is disabled.

The **gconfig** subroutine sets the back buffer to True.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 withAIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Enabling drawing into the front buffer with the **frontbuffer** subroutine.

Finding out which buffers are enabled for writing with the **getbuffer** subroutine.

Exchanging the front and back buffers with the **swapbuffers** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Configuring the Frame Buffer, Creating Animated Scenes.

---

backface Subroutine

Purpose

Enables and disables backfacing polygon removals.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void backface(Int32  bool)
```

FORTRAN Syntax

```
SUBROUTINE BACKFA(bool)
LOGICAL bool
```

Description

The **backface** subroutine allows or suppresses the display of backfacing filled polygons. With the **frontface** subroutine, it is useful for drawing polygons that have different colors on each side. The **frontface** subroutine can also be useful for performing a primitive kind of hidden surface removal.

A backfacing polygon is defined as a polygon whose vertices are in clockwise order in screen coordinates. When backfacing polygon removal is on, the system displays only polygons whose vertices are in counterclockwise order.

For programs representing solid objects as collections of polygons, this subroutine can be used to help remove hidden surfaces. The **backface** subroutine works best for simple convex objects that do not obscure other objects.

For complicated objects, this routine alone may not remove all hidden surfaces. To remove hidden surfaces for more complicated objects or groups of objects, use the **zbuffer** subroutine. The **zbuffer** subroutine checks the relative distances of the figure from the viewer (z values) and draws only the nearest figures.

If both frontfacing and backfacing polygon removal is enabled, no filled polygons are displayed. (An exception is made if the graphics pipeline cannot determine if a polygon is front- or backfacing because it is degenerate or otherwise exceptional.)

Notes:

1. 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 the **backface** subroutine to do the opposite of what is intended.
2. If a polygon is extremely small or has degenerate vertices, the graphics pipeline cannot determine if the polygon is backfacing. In such cases, the polygon is rendered by default.

Parameter

bool Specifies a value for the state of backfacing polygon removal. The settings for the *bool* parameter are:

False = backfacing polygon removal disabled.
True = backfacing polygon removal enabled.

Example

The example C language program **backface.c** uses the **backface** subroutine to draw a cube two ways to demonstrate the difference between allowing or suppressing the display of backfacing polygons.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows Desktop/3D Feature.

On the POWERgraphics GTO adapter, either frontfacing polygon removal or backfacing polygon removal can be specified, but not both simultaneously. Enabling backfacing culling disables frontfacing culling.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Initiating z-buffer mode with the **zbuffer** subroutine.

Graphics Library Overview and Removing Hidden Surfaces.

---

bbox2 Subroutine

Purpose

Specifies the bounding box and minimum pixel radius.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

FORTRAN Syntax

```
SUBROUTINE BBOX2(xmin, ymin, x1, y1, x2, y2)
INTEGER*4 xmin, ymin
REAL x1, y1, x2, y2

SUBROUTINE BBOX2I(xmin, ymin, x1, y1, x2, y2)
INTEGER*4 xmin, ymin, x1, y1, x2, y2

SUBROUTINE BBOX2S(xmin, ymin, x1, y1, x2, y2)
INTEGER*2 xmin, ymin, x1, y1, x2, y2
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference. Also, the FORTRAN INTEGER*2 version, the **BBOX2S** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

Description

The **bbox2** subroutine controls the execution of routines in a GL object by performing the graphical functions known as culling and pruning.

The **bbox2** subroutine calculates the bounding box, transforms it to screen coordinates, and compares it to the viewport. If the bounding box is completely outside the viewport, the routines between the call to the **bbox2** subroutine and the end of the object are ignored.

If the bounding box is within the viewport, the system compares it with the minimum feature size. If the box is too small in both the *x* and *y* dimensions, the rest of the routines in the object are ignored.

Overuse of the **bbox2** subroutine can impair performance, so it is best reserved for complicated object definitions.

Note: This subroutine can be used only to add to a display list.

Parameters

xmin	Specifies the width, in pixels, of the smallest displayable feature.
ymin	Specifies the height, in pixels, of the smallest displayable feature.
x1	Specifies the *x* coordinate of a corner of the bounding box.
y1	Specifies the *y* coordinate of a corner of the bounding box.
x2	Specifies the *x* coordinate of a corner of the bounding box. This corner must be diagonally opposite the corner (*x1, y1*).
y2	Specifies the *y* coordinate of a corner of the bounding box. This corner must be diagonally opposite the corner (*x1, y1*).

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Creating an object with the **makeobj** subroutine.

Graphics Library Overview, Creating Objects (Display Lists), and Using Viewports and Screenmasks.

---

bgnclosedline or endclosedline Subroutine

Purpose

Begins or ends interpretation of vertex routines as closed line vertices.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void bgnclosedline( )
void endclosedline( )
```

FORTRAN Syntax

```
SUBROUTINE BGNCLO
SUBROUTINE ENDCLO
```

Description

The **bgnclosedline** subroutine marks the start of a group of vertex (begin-end style) subroutines to be interpreted as points on a closed line. It is the same as the **bgnline** subroutine, except that it connects the last vertex to the first. For example,

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

draws the outline of a triangle.

The group of begin-end style subroutines terminates with the **endclosedline** subroutine. After the **endclosedline** subroutine, the system draws a line from the final vertex back to the initial vertex, and the current graphics position is left undefined.

All line segments use the current linestyle and linewidth. The **bgnclosedline** subroutine resets the linestyle counter; that is, the first pixel drawn uses the first bit of the linestyle, and so on. The linestyle counter is *not* reset on subsequent vertices; instead, the pattern continues around corners, until the **endclosedline** subroutine is reached. To reset the pattern, call the **endclosedline** subroutine and then the **bgnclosedline** subroutine again.

If the color changes between a pair of vertices, the color of the line segment is 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 gamma ramp.

Between calls to the **bgnclosedline** and **endclosedline** subroutines, you can issue calls to the following GL subroutines only:

- **c**
- **color**
- **cpack**
- **lmbind**
- **lmcolor**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

Use the **lmdef** and **lmbind** subroutines to respecify only materials and their properties.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GTO and the 3D Color Graphics Processor support a maximum of 255 vertices between the **bgnclosedline** and **endclosedline** subroutines.

The POWERgraphics Gt4 and GXT1000 support an arbitrary number of vertices between the **bgnclosedline** and **endclosedline** subroutines.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing vertex-based lines with the **bgnline** subroutine.

Selecting a linestyle pattern with the **setlinestyle** subroutine.

Transferring a vertex to the graphics pipe with the **v** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes.

---

bgnline or endline Subroutine

Purpose

Begins or ends interpretation of vertex routines as line vertices.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void bgnline( )
void endline( )
```

FORTRAN Syntax

```
SUBROUTINE BGNLIN
SUBROUTINE ENDLIN
```

Description

The **bgnline** subroutine begins the interpretation of vertex (begin-end style) subroutines as line vertices. It is like the **bgnclosedline** subroutine, except that the last vertex does not connect to the first vertex.

Vertices specified after the **bgnline** subroutine and before the **endline** subroutine are interpreted as endpoints of a series of line segments. Use the **v** subroutine 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. All segments use the current linestyle and linewidth.

After the **endline** subroutine executes, the current graphics position is undefined.

All line segments use the current linestyle and linewidth. The **bgnline** subroutine resets the linestyle counter; that is, the first pixel drawn uses the first bit of the linestyle, and so on. The linestyle counter is *not* reset on subsequent vertices; instead, the pattern continues around corners, until the **endline** subroutine is reached. To reset the pattern, call the **endline** subroutine and then the **bgnline** subroutine again.

If the color changes between a pair of vertices, the color of the line segment is 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 gamma ramp>.

Between calls to the **bgnline** and **endline** subroutines, you can issue calls to the following Graphics Library subroutines only:

- **c**
- **color**
- **cpack**
- **lmbind**
- **lmcolor**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

Use the **lmdef** and **lmbind** subroutines to respecify only materials and their properties.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER Gt4, POWER Gt4x, and POWER GXT1000 adapters support an arbitrary number of vertices between the **bgnline** and **endline** subroutines.

The POWERgraphics GTO and the 3-D Color Graphics Processor support a maximum of 255 vertices between the **bgnline** and **endline** subroutines.

The POWER Gt4x will restart the linestyle stipple pattern periodically if a large number of vertices is issued between a **bgnline()** and **endline** pair. Such a reset will not occur within the first 255 vertices but will occur sometime between the first 400 and 2000 vertices and will then restart at multiples of this amount. The actual point at which the restart occurs depends on whether normals, colors, and/or materials have been specified in addition to the vertex data.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing closed line vertices with the **bgnclosedline** subroutine.

Setting the current color in RGB mode with the **c** subroutine.

Selecting the shading model with the **shademodel** subroutine.

Transferring a vertex to the graphics pipe with the **v** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes.

---

bgnpoint or endpoint Subroutine

Purpose

Begins or ends interpretation of vertex subroutines as points.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

FORTRAN Syntax

```
SUBROUTINE BGNPOI
SUBROUTINE ENDPOI
```

Description

The **bgnpoint** subroutine marks the beginning of a list of vertex (begin-end style) subroutines to interpret as points. Use the **endpoint** subroutine to mark the end of the list. For each vertex, the system draws a one-pixel point into the frame buffer. After the **endpoint** subroutine completes, the current graphics position is undefined. Use the **v** subroutine to specify a vertex.

Between the **bgnpoint** and **endpoint** subroutines, you can issue only the following Graphics Library subroutines:

- **c**
- **color**
- **cpack**
- **lmbind**
- **lmcolor**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

Use the **lmbind** and **lmdef** subroutines to respecify only materials and their properties.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER Gt4 and POWER Gt4xThe POWER Gt4, POWER Gt4x, and POWER GXT1000 adapters support an arbitrary number of vertices between the **bgnpoint** and **endpoint** subroutines.

The POWERgraphics GTO and the 3D Color Graphics Processor support a maximum of 255 vertices between the **bgnpoint** and **endpoint** subroutines.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current color in RGB mode with the **c** subroutine.

Transferring a vertex to the graphics pipe with the **v** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes.

---

bgnpolygon or endpolygon Subroutine

Purpose

Begins or ends interpretation of vertex subroutines as polygon vertices.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void bgnpolygon( )
void endpolygon( )
```

FORTRAN Syntax

```
SUBROUTINE BGNPOL
SUBROUTINE ENDPOL
```

Description

The **bgnpolygon** subroutine begins the interpretation of vertex (begin-end style) subroutines as polygon vertices. Vertices specified after the **bgnpolygon** subroutine and before the **endpolygon** subroutine form a single polygon.

Self-intersecting polygons (other than four-point bowties) may render incorrectly. To force the system to render concave polygons correctly, call the **concave** subroutine with the parameter set to True.

Between the **bgnpolygon** and **endpolygon** subroutines, you can issue only the following Graphics Library subroutines:

- **c**
- **color**
- **cpack**
- **lmbind**
- **lmcolor**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

After the **endpolygon** subroutine completes, the current graphics position is undefined.

Use the **lmbind** and **lmdef** subroutines to respecify only materials and their properties.

Use the **v** subroutine to specify a vertex.

Use the **backface** subroutine to eliminate backfacing polygons. Polygons with vertices specified in clockwise order are not drawn.

> **Attention:** Calling **concave(TRUE)** guarantees that all polygons are drawn correctly, but may cause degradation of performance. Do not set **concave(TRUE)** if you plan to draw only convex polygons.

Example

The example C language program **cylinder2.c** uses the **bgnpolygon** subroutine to define the beginning of a description of a polygon.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER Gt4 and POWER Gt4x adapters support only a limited number of vertices between the **bgnpolygon** and **endpolygon** subroutines. The actual limit can vary from less than 200 vertices to more than 400 vertices, depending on whether or not colors, normals, or material properties are specified between the begin and end pair. If only vertices are specified between a begin and end pair, then at least 255 vertices are supported. Specifying normals, colors, or material properties decreases the overall maximum number of vertices.

The POWERgraphics GTO and the 3D Color Graphics Processor support a maximum of 255 vertices between the **bgnpolygon** and **endpolygon** subroutines.

The POWERgraphics GXT1000 supports an arbitrary number of vertices between the **bgnpolygon** and **endpolygon** subroutines, if the concave flag has been set to FALSE. If concave polygon checking is enabled, then a maximum of 255 vertices is supported.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allowing the system to draw concave polygons with the **concave** subroutine.

Transferring a vertex to the graphics pipe with the **v** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes.

---

bgnsurface or endsurface Subroutine

Purpose

Delimit a NURBS surface definition.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void bgnsurface( )
void endsurface( )
```

FORTRAN Syntax

```
SUBROUTINE BGNSUR
SUBROUTINE ENDSUR
```

Description

The **bgnsurface** and **endsurface** subroutines mark the beginning and end, respectively, of a Non-Uniform Rational B-Spline (NURBS) surface definition. Between calls to these two subroutines, only those subroutines that define the surface and provide the trimming information can be invoked. They are:

- **bgntrim**
- **endtrim**
- **nurbscurve**
- **nurbssurface**
- **pwlcurve**

The NURBS surface definition must consist of exactly one call to the **nurbssurface** subroutine (to define the shape of the surface), followed by a list of zero or more trimming loop definitions (to define the boundaries of the surface). Each trimming loop definition consists of one call to the **bgntrim** subroutine, one or more calls to the **pwlcurve** or **nurbscurve** subroutines, and one call to the **endtrim** subroutine.

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; otherwise much surface information is lost. When using a lighting model, define or modify materials and their properties with the **lmdef** and **lmbind** subroutines.

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

```
bgnsurface(...);
    nurbssurface(...);
    bgntrim();
        pwlcurve(...);
    endtrim();
    bgntrim();
        nurbscurve(...);
        nurbscurve(...);
    endtrim();
endsurface();
```

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Marking the beginning and end of a NURBS surface trimming loop with the **bgntrim** and **endtrim** subroutines.

Returning the current value of a trimmed NURBS surface display property with the **getnurbsproperty** subroutine.

Controlling the shape of a NURBS trimming curve with the **nurbscurve** subroutine.

Controlling the shape of a NURBS surface with the **nurbssurface** subroutine.

Describing a piecewise linear trimming curve for NURBS surfaces with the **pwlcurve** subroutine.

Setting a property for the display of trimmed NURBS with the **setnurbsproperty** subroutine.

Graphics Library Overview, Creating Lighting Effects, and Drawing NURBS Curves and Surfaces.

---

bgntmesh or endtmesh Subroutine

Purpose

Begins or ends interpretation of vertex subroutines as triangle mesh vertices.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void bgntmesh( )
void endtmesh( )
```

FORTRAN Syntax

```
SUBROUTINE BGNTME
SUBROUTINE ENDTME
```

Description

The **bgntmesh** subroutine begins the system interpretation of vertex (begin-end style) subroutines as triangle mesh vertices, which are used to define a mesh of triangles.

The graphics pipeline maintains two vertex registers. The first and second vertices are loaded into the registers. When the third vertex routine is executed, the system draws a triangle through the vertices and replaces the older of the register vertices with the third vertex.

For each new vertex subroutine, the system draws a triangle through the new vertex and the stored vertices, then replaces the older stored vertex with the new vertex.

To replace the more recent of the stored vertices, call the **swaptmesh** subroutine. For example, the code sequence:

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

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

```
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 the **swaptmesh** subroutine can be called.

Between the **bgntmesh** and **endtmesh** subroutines, you can issue only the following Graphics Library subroutines:

- **c**
- **color**
- **cpack**
- **lmbind**
- **lmcolor**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

Use the **lmbind** and **lmdef** subroutines to respecify only materials and their properties.

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER Gt4 and POWER Gt4x adapters support only a limited number of vertices between the **bgntmesh** and **endtmesh** subroutines. The actual limit can vary from less than 200 vertices to more than 400 vertices, depending on whether or not colors, normals, or material properties are specified between the begin and end pair. If only vertices are specified between a begin and end pair, then at least 255 vertices are supported. Specifying normals, colors, material properties, or the **swaptmesh** subroutine decreases the overall maximum. If all attributes are specified, only 198 vertices are supported.

The POWERgraphics GTO and the 3D Color Graphics Processor support a maximum of 255 vertices between the **bgntmesh** and **endtmesh** subroutines.

The POWERgraphics GXT1000 supports an arbitrary number of vertices between the **bgntmesh** and **endtmesh** subroutines.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current color in RGB mode with the **c** subroutine.

Toggling the triangle mesh register pointer with the **swaptmesh** subroutine.

Transferring a vertex to the graphics pipe with the **v** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, and Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

bgntrim or endtrim Subroutine

Purpose

Delimit a NURBS surface trimming loop.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

FORTRAN Syntax

```
SUBROUTINE BGNTRI
SUBROUTINE ENDTRI
```

Description

The **bgntrim** and **endtrim** subroutines mark the beginning and 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 Non-Rational B-Spline (NURBS) surface. 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, in a definition for NURBS surface that resembles a rectangle with a hole punched out, there are two trimming loops. One loop defines the outer edge of the rectangle. The other trimming loop defines the hole punched out of the rectangle. The definitions of each of these trimming loops is 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 (as defined by the **pwlcurve** subroutine), or as a single NURBS curve (as defined by the **nurbscurve** subroutine), or as a combination of both in any order. The only GL subroutines that can appear in a trimming loop definition (between the **bgntrim** and **endtrim** subroutines) are the **pwlcurve** and **nurbscurve** subroutines.

The following code fragment defines 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 counterclockwise trimming loop and outside for a clockwise trimming loop. For the rectangle mentioned previously, the trimming loop for the outer edge of the rectangle should run counterclockwise, 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. In other words, 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 endpoints 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. In other words, 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.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Marking the beginning and end of a NURBS surface definition with the **bgnsurface** and **endsurface** subroutines.

Returning the current value of a trimmed NURBS surfaces display property with the **getnurbsproperty** subroutine.

Controlling the shape of a NURBS trimming curve with the **nurbscurve** subroutine.

Controlling the shape of a NURBS surface with the **nurbssurface** subroutine.

Describing a piecewise linear trimming curve for NURBS surfaces with the **pwlcurve** subroutine.

Setting a property for the display of trimmed NURBS with the **setnurbsproperty** subroutine.

Graphics Library Overview and Drawing NURBS Curves and Surfaces.

---

blankscreen Subroutine

Purpose

Turns screen refresh on and off.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void blankscreen(Int32  bool)
```

FORTRAN Syntax

```
SUBROUTINE BLANKS(bool)
LOGICAL bool
```

Description

The **blankscreen** subroutine turns screen refresh on and off. Normally, the screen is refreshed 60 times a second. If the screen is not regularly refreshed, it goes blank. The screen refresh is turned on or off immediately upon invocation of this subroutine.

The action of this subroutine is not affected by the **blanktime** subroutine. The **blankscreen** subroutine affects the entire screen, not just an individual window.

Notes:

1. This subroutine cannot be used to add to a display list.
2. This call is intended for use by a window manager. If a window manager is already running, it is possible that this call will be overridden by the window manager.

Parameter

bool Specifies a value for the screen display. The settings for the *bool* parameter are:
True = Display stops and screen turns black *immediately*.
False = Display is restored.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the screen-blanking time out with the **blanktime** subroutine.

Creating and Windows, Understanding the Hardware Used by GL, Understanding Understanding Windows and Input Control.

---

blanktime Subroutine

Purpose

Sets the screen-blanking timeout.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void blanktime(Int32  nframe)
```

FORTRAN Syntax

```
SUBROUTINE BLANKT(nframe)
INTEGER*4 nframe
```

Description

The **blanktime** subroutine changes the amount of time the system waits before it blanks the screen to protect the color display. The default delay before the screen turns black is 15 minutes after the last input. This subroutine can also disable the screen-blanking feature.

To calculate the value of the *nframe* parameter, multiply the desired blanking delay period (in seconds) by 60. For example, when *nframe* is 18000, the blanking delay period is 5 minutes. If there are 60 frames per second, *nframe* is 60 times the number of seconds that the system waits before blanking the screen. To disable screen blanking, use 0 (zero) as the value for *nframe.*

Notes:

1. This subroutine cannot be used to add to a display list.
2. This call is intended for use by a window manager. If a window manager is already running, it is possible that this call will be overridden by the window manager.

Parameters

nframe Specifies the number of frames after which to blank the screen. This subroutine assumes 60 frames per second.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning screen refresh off or on with the **blankscreen** subroutine.

Creating and Managing Windows, Understanding the Hardware Used by GL, Windows and Input Control.

---

blendfunction Subroutine

Purpose

Specifies the function to be used for alpha blending.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void blendfunction(Int32  sfactor, Int32  dfactor)
```

FORTRAN Syntax

```
SUBROUTINE BLENDF(sfactor, dfactor)
INTEGER*4 sfactor, dfactor
```

Description

In RGB mode, the system draws pixels using a function that blends the current (destination) RGBA values of the pixel with the RGBA values to be superimposed on that pixel (the source values).

Most often, blending is simple: the source RGBA values replace the destination RGBA values of the pixel. However, if a colored transparent primitive is drawn on top of another primitive, then the RGBA values of the new primitive must be blended with the RGBA values of the underlying primitive. (The transparency or opacity of a primitive can be stored as an alpha value.)

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

```
Rblended = (Rsource * sfactor) + (Rdestination * dfactor)
Gblended = (Gsource * sfactor) + (Gdestination * dfactor)
Bblended = (Bsource * sfactor) + (Bdestination * dfactor)
Ablended = (Asource * sfactor) + (Adestination * dfactor)
```

where R is a red value, G is a green value, B is a blue value, and A is an alpha value. Blending is available with or without z-buffer mode. Blending works properly only in RGB mode. In color map mode, the results are unpredictable.

Blending is effectively deactivated by setting the *sfactor* parameter to BF_ONE and *dfactor* to BF_ZERO (the default values). RGB mode fill rates are significantly higher when blending is effectively deactivated.

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. These default values can be changed with the **readsource** subroutine.

Blending factors use RGBA values converted to percentages of maximum value (255 in current hardware). To improve performance, conversion calculations are approximate. However, 0 converts exactly to 0.0, and maximum value converts exactly to 1.0.

Symbolic Constants		
C	**FORTRAN**	**Description**
BF_ZERO	BFZERO	0
BF_ONE	BFONE	1
BF_DC	BFDC	(destination RGBA)/(maximum value)
BF_SC	BFSC	(source RGBA)/(maximum value)

BF_MDC	BFMDC	1 - (destination RGBA)/(maximum value)
BF_MSC	BFMSC	1 - (source RGBA)/(maximum value)
BF_SA	BFSA	(source alpha)/(maximum value)
BF_MSA	BFMSA	1 - (source alpha)/(maximum value)
BF_DA	BFDA	(destination alpha)/(maximum value)
BF_MDA	BFMDA	1 - (destination alpha)/(maximum value)

Parameters

sfactor Specifies a symbolic constant that identifies the blending factor by which to scale contributions from source (incoming) pixel RGBA (red, green, blue, alpha) values.

dfactor Specifies a symbolic constant that identifies the blending factor by which to scale contributions from destination (current) pixel RGBA values.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 supports source alpha blending. However, the GXT1000 does not support destination alpha blending. Destination blending is available using OpenGL.

When rendering semi-transparent polygons, it is recommended that z-buffering should be enabled, with the the z compare function set to ZF_LESS. Because the POWERgraphics GTO graphics adapter always renders ″fat″ triangles (that is, non-point sampled triangles whose upper and left edges are one pixel wider than those of a point-sampled triangle), the edges of bordering triangles are drawn twice, and thus double-blended, unless z-buffering is used to eliminate the double-rendering of edges. The POWERgraphics GTO graphics adapter forms all polygons into triangles; thus, the internal form of the polygon's edges become visible unless z-buffering is used to eliminate the edges.

The POWERgraphics GTO graphics adapter supports transparency by implementing a limited subset of the **blendfunction** subroutine functionality. The BF_SA and BF_MSA alpha-blending modes are supported when used in combination.

Semi-transparent lines and polygons are supported (this class includes circles, arcs, spline curves and NURBS surfaces); semi transparent blits (including raster text) are not supported. The alpha-blending factor must be a constant for the entire primitive; the interpolation of alpha values from vertex to vertex is not supported. Alpha values can be set with the **cpack**, **c4[isf]**, or **lmdef** subroutines.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.

/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying RGBA color with a single, packed 32-bit integer using the **cpack** subroutine.

Specifying the source for pixels to be read with the **readsource** subroutine.

Writemasks and Logical Operations, and Working in Color Map and RGB Modes.

blink Subroutine

Purpose
Changes a color map entry at a selectable rate.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
void blink(lnt16 *rate*, **Colorindex** *color*, **lnt16** *red*, **lnt16** *green*, **lnt16** *blue*)

FORTRAN Syntax
```
SUBROUTINE BLINK(rate, color, red, green, blue)
INTEGER*4 rate, color, red, green, blue
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when started from a FORTRAN program, although it accepts short integers when started from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description
The **blink** subroutine alternates the color located at index *color* 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.

Up to 20 colors can blink simultaneously, each at a different rate. The blink rate is changed by calling the **blink** subroutine with the same *color* parameter and a different *rate* parameter.

For example, if the *rate* parameter is 3, the color changes (blinks) every third vertical retrace. Its value alternates between the original value and the new value supplied by the parameters *red*, *green*, and *blue*.

The length of time between retraces varies according to the monitor used. When using a 60Hz monitor, 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, set the *color* parameter to the color map entry for which to stop blinking and call the **blink** subroutine with the *rate* parameter set to 0.

To terminate all blinking colors simultaneously, call the **blink** subroutine with the *rate* parameter set to -1. When *rate* is -1, the other parameters are ignored.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters
rate Specifies the number of vertical retraces per blink. On a standard 60 Hz monitor, there are 60 vertical retraces per second.

color Specifies the index into the current color map. The color defined at that index is the color that is blinked (alternated).

red Specifies the red value of the alternate color that blinks against the color selected from the color map by the *color* parameter.

green Specifies the green value of the alternate color that blinks against the color selected from the color map by the *color* parameter.

blue Specifies the blue value of the alternate color that blinks against the color selected from the color map by the *color* parameter.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER GXT1000 adapter does not support the **blink** subroutine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current color in color map mode with the **color** subroutine.

Changing a color map entry to a specified RGB value with the **mapcolor** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Working in Color Map and RGB Modes, Creating a Cursor, Creating Animated Scenes.

---

blkqread Subroutine

Purpose

Reads multiple entries from the event queue.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 blkqread(Int16 * data, Int16  number)
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION BLKQRE(data, number)
INTEGER*2 data(number)
INTEGER*4 number
```

Note: For FORTRAN users, this function accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **blkqread** subroutine reads multiple entries from the event queue and stores them in the buffer designated by the *data* parameter.

The returned value of the function is the number of queue entries actually read into the *data* buffer. This function fills the *data* buffer alternatively with device numbers and device values. Thus, the number of entries read is never more than the value or the *number* parameter divided by two.

Note: This subroutine cannot be used to add to a display list.

Parameters

data	Specifies a pointer to the buffer that is to receive the queue information.
number	Specifies the length of the buffer.

Return Value

The number of 16-bit words of data actually read into the buffer pointed to by the *data* parameter. This number is twice the number of complete queue entries read because each queue entry consists of two 16-bit words.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling an input device for event queuing with the **qdevice** subroutine.

Reading the first entry in the event queue with the **qread** subroutine.

Graphics Library Overview, Controlling Queues and Devices.

---

c Subroutine

Purpose

Sets the current color in RGB mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

	3-D	4-D
Int16	**void c3s(Int16** *vector*[3])	**void c4s(Int16** *vector*[4])
Int32	**void c3i(Int32** *vector*[3])	**void c4i(Int32** *vector*[4])
float	**void c3f(float** *vector*[3])	**void c4f(float** *vector*[4])

FORTRAN Syntax

	3-D	4-D
INTEGER*2	**SUBROUTINE C3S(***vector***)** INTEGER*2 *vector*(3)	**SUBROUTINE C4S(***vector***)** INTEGER*2 *vector*(4)
INTEGER*4	**SUBROUTINE C3I(***vector***)** INTEGER*4 *vector*(3)	**SUBROUTINE C4I(***vector***)** INTEGER*4 *vector*(4)
FLOAT	**SUBROUTINE C3F(***vector***)** REAL *vector*(3)	**SUBROUTINE C4F(***vector***)** REAL *vector*(4)

Description

The **c** subroutine changes the current RGBA (red, green, blue, alpha) color. Array components 0, 1, 2, and 3 are red, green, blue, and alpha, respectively. In the three-component cases, alpha defaults to 1.0 (float) or 255 (integer).

Floating-point components range from 0.0 through 1.0. Integer components range from 0 through 255.

Note: This subroutine is available only in RGB mode.

Parameter

vector Specifies, for the **c4** subroutines, a four-element array containing RGBA (red, green, blue, and alpha) values. For **c3** subroutines, 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.

Example

The example C language program **cylinder1.c** calls the **c3f** subroutine, with a three element array initialized with all zeros as the *vector* parameter, to clear the screen to black.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

If your system does not have alpha bitplanes, set the alpha values to zero.

Related Information

Setting the current color in color map mode with the **color** subroutine.

Specifying RGBA color with a single packed 32-bit integer using the **cpack** subroutine.

Returning the current color in RGB mode with the **gRGBcolor** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Graphics Library Overview, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes.

---

callobj Subroutine

Purpose

Draws an instance of an object (display list).

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void callobj(Int32  object)
```

FORTRAN Syntax

```
SUBROUTINE CALLOB(object)
INTEGER*4 object
```

Description

The **callobj** subroutine draws an instance of a previously defined display list (object). If the subroutine specifies an undefined object, the system ignores the routine.

Global state attributes are not saved before a call to the **callobj** subroutine. Thus, if you change a variable within an object, such as color, the change can affect the caller as well.

Use the **pushattributes** and **popattributes** subroutines to preserve global state attributes across calls. Also, the object may execute transformations that change the matrix stack. Use the **pushmatrix** and **popmatrix** subroutines to restore the state of the matrix stack.

> **Note:** This editing subroutine can be added to a display list.

Parameter

object Specifies the identifier of the object to be drawn.

Example

The example C language program **depthcue.c** uses the **callobj** subroutine, after specifying a rotation transformation, to rotate a graphical object (a cube).

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Creating an object with the **makeobj** subroutine.

Popping the attribute stack with the **popattributes** subroutine.

Popping the transformation matrix stack with the **popmatrix** subroutine.

Saving the global state attributes with the **pushattributes** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Graphics Library Overview, Creating Objects (Display Lists), Setting Drawing Attributes, and Working with Coordinate Systems.

---

charstr Subroutine

Purpose

Draws a string of raster characters on the screen.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void charstr(Char8 * string)
```

FORTRAN Syntax

```
SUBROUTINE CHARST( string,  length)
CHARACTER*(*) string
INTEGER*4 length
```

Description

The **charstr** subroutine draws a string of text, using the current raster font. After each character is drawn, the character width is added to the current character position. The text string is drawn in the current raster font and color, using the current writemask.

Characters that are not defined in the current raster font are treated as having zero size and are therefore ignored.

The **charstr** subroutine supports both single- and double-byte raster character renderings.

If the current font is a double-byte font, this subroutine expects the first two bytes to represent the first character, the second two bytes to represent the next character, and so on. Double-byte fonts are useful in languages with extremely large character sets such as Japanese and Chinese.

If the current font is a single-byte font, each byte represents one character. The ASCII code set is an example of a single-byte font.

It is the user's responsibility to determine if the currently bound font is a single- or double-byte font and to pass the appropriate string. To determine the font type, use the **getfonttype** subroutine.

Character Clipping

The beginning of the string is positioned at the current character position. The string is drawn from left to right, parallel to the surface of the screen. The string is fine clipped to the rectangular area defined with the **scrnmask** subroutine. The current color and writemask are used. Characters not defined in the current raster font are assumed to have zero width and are not drawn.

Strings drawn by the **charstr** subroutine are subject to clipping and z-buffering. The algorithmic details of this process are as follows:

1. Trivial accept/trivial reject test. If the start point of the test string (after transformation by all matrices) is *not* within the NDC unit cube (-w<x,y,z<+w< x,y,z < +1.0) then the *entire* string is trivially rejected (not drawn).
2. If the preceding test passes, then z-buffering proceeds on a pixel-by-pixel basis. At the start point, the string is mapped to window-relative pixel coordinates. The z-value of the reference point is used as the z-value of the entire string. Then, for every pixel of every letter in the string:
   a. If that pixel is outside of the screenmask, the pixel is not drawn.
   b. If the pixel is inside the screenmask, then use z-compare. If the z-test passes, then update the frame buffer, *and* update the z-buffer (for that pixel only).

Glyph Lookup for SBCS and DBCS Strings

Each character of the string should be thought of as a *glyph index*. That is, each byte (for an SBCS font) or each pair of bytes (for a DBCS font) serves as an index into a table of raster glyphs. The table of glyphs is the table that was loaded with the **defrasterfont** subroutine or the **loadXfont** subroutine. The mapping of glyph indices to glyphs is a constant, and is locale-independent. There is no processing or translation that is applied to the glyph index before the raster is looked up. The *raw* value is used unmodified. A single null byte is sufficient to indicate end-of-string for ordinary (single-byte) fonts; a null short integer (two consecutive null bytes) is required to terminate a DBCS string.

Because the **charstr** routine does not perform any translation and conversion, it is left up to the application to make sure that appropriate fonts are loaded, and that multibyte text strings are converted into sequences of glyph indices. Some tools for performing such conversions are provided by the base operating system.

> **Note:** The default font (font index 0) is locale-dependent; during initialization, a different default font is loaded, depending on the locale. The default font is always chosen from the iso8859 family of fonts.

Parameters

string	Specifies a pointer to the variable containing the string.
length	Specifies the length (number of characters) of the string.

Example

The example C language programs **curved.c** and **font3.c** use the **charstr** subroutine to draw a character string in the current raster font.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Determining the font format with the **getfonttype** subroutine.

Determining the font name with the **getfontencoding** subroutine.

Moving the current character position with the **cmov** subroutine.

Defining bitmaps for a raster font with the **defrasterfont** subroutine.

Selecting a raster font with the **font** subroutine.

Returning the width of the specified text string with the **strwidth** subroutine.

National Language Support Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Creating Text Characters.

---

chunksize Subroutine

Purpose

Specifies the amount of memory to be allocated for a display list.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void chunksize(Int32  chunk)
```

FORTRAN Syntax

```
SUBROUTINE CHUNKS(chunk)
INTEGER*4 chunk
```

Description

The **chunksize** subroutine gives the system a hint about the appropriate amount of memory to be allocated when compiling a display list. The system may, on occasion, override the hint.

As you add primitives to a display list, the *chunk* parameter is the unit size (in bytes) by which the memory allocated to the display list grows. The default chunk size is 1024 bytes. A chunk size that is much smaller than the final size of the display list leads to inefficiencies due to fragmentation. A chunk size that is larger than the final display list contains unused, and therefore wasted, memory.

Most subroutines add from 4 to 28 bytes to the display list; subroutines that accept arrays as parameters (for example, the **poly** subroutine and **polf** subroutine) typically add to the display list in proportion to the length of the array. Some experimentation may be necessary to determine the optimal chunk size for an application.

Note: This editing subroutine itself cannot be added to a display list.

Parameter

chunk Specifies the minimum memory size to allocate for an object.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Unfragmenting and compacting the memory storage of an object with the **compactify** subroutine.

Initializing the system with the **ginit** subroutine.

Creating an object with the **makeobj** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

circ Subroutine

Purpose

Outlines a circle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

FORTRAN Syntax

```
SUBROUTINE CIRC(x, y, radius)
REAL x, y, radius
SUBROUTINE CIRCI(x, y, radius)
INTEGER*4 x, y, radius
SUBROUTINE CIRCS(x, y, radius)
INTEGER*2 x, y, radius
```

> **Note:** For FORTRAN users, the INTEGER*2 version, the **CIRCS** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

Description

The **circ** subroutine draws an unfilled circle in the *x*-*y* plane ($z = 0$). The system draws the circle using the current line attributes: color, linestyle, linewidth, repeat factor, and writemask.

To create a circle that does not lie in the *x*-*y* plane, draw the circle in the *x*-*y* plane, then rotate or translate the circle. Circles rotated outside the 2-D *x*-*y* plane appear as ellipses.

All of the routines listed in the syntax are functionally the same. They differ only in the type declarations for the coordinates.

Parameters

x	Specifies the *x* coordinate of the center of the circle specified in modeling coordinates.
y	Specifies the *y* coordinate of the center of the circle specified in modeling coordinates.
radius	Specifies the length of the radius of the circle.

Example

The example C language program **boxcirc.c** uses the **circi** subroutine to draw a red circle.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a circular arc with the **arc** subroutine.

Drawing a filled circular arc with the **arcf** subroutine.

Drawing a filled circle with the **circf** subroutine.

Drawing a curve with the **crv** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

circf Subroutine

Purpose

Draws a filled circle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

FORTRAN Syntax

```
SUBROUTINE CIRCF(x, y, radius)
REAL x, y, radius
SUBROUTINE CIRCFI(x, y, radius)
INTEGER*4 x, y, radius
SUBROUTINE CIRCFS(x, y, radius)
INTEGER*2 x, y, radius
```

> **Note:** For FORTRAN users, the INTEGER*2 version, the **CIRCFS** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

Description

The **circf** subroutine draws a filled circle in the *x-y* plane (*z* = zero). The system draws the circle using the current area attributes: color, repeat factor, and writemask. To create a filled circle that does not lie in the *x-y* plane, draw the circle in the *x-y* plane, then rotate or translate the circle. Filled circles rotated outside the 2-D *x-y* plane appear as filled ellipses.

All of the routines listed in the syntax are functionally the same. They differ only in the type declarations for the coordinates.

Parameters

x	Specifies the *x* coordinate of the center of the circle specified in modeling coordinates.
y	Specifies the *y* coordinate of the center of the circle specified in modeling coordinates.
radius	Specifies the length of the radius of the circle.

Example

The example C language program **tpbig.c** uses the **circf** subroutine to draw a scoop of cherry ice cream.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a circular arc with the **arc** subroutine.

Drawing a filled circular arc with the **arcf** subroutine.

Drawing a circle with the **circ** subroutine.

Drawing a curve with the **crv** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

clear Subroutine

Purpose

Clears to the screenmask.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void clear( )
```

FORTRAN Syntax

```
SUBROUTINE CLEAR
```

Description

The **clear** subroutine sets the screen area inside the current screenmask to the current color, using the current writemask and pattern. Only the portion of the current screenmask that is inside the current window is actually cleared.

By default, the screenmask is exactly the same size as the window. Use the **scrmsk** subroutine to change the size of the screenmask.

Note: The **viewport** subroutine resets the screenmask to be precisely the same size as the viewport.

Example

The example C language program **boxcirc.c** uses the **clear** subroutine to clear the viewport before drawing a box and a circle.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Clearing the color bitplanes and the z-buffer simultaneously with the **czclear** subroutine.

Drawing a filled screen-aligned rectangle with the **sboxf** subroutine.

Clearing the z-buffer with the **zclear** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Clearing, Resetting, and Initializing GL, Setting Drawing Attributes, and Using Viewports and Screenmasks.

---

clkoff or clkon Subroutine

Purpose

Turns on or off the keyboard click.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void clkoff( )
void clkon( )
```

FORTRAN Syntax

```
SUBROUTINE CLKOFF
SUBROUTINE CLKON
```

Description

The **clkoff** subroutine turns off the keyboard click, and the **clkon** subroutine turns it on.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning off the keyboard display lights with the **lampon** or **lampoff** subroutine.

Ringing the keyboard bell with the **ringbell** subroutine.

Setting the duration of the keyboard bell sound with the **setbell** subroutine.

Graphics Library Overview and Using the Keyboard.

---

closeobj Subroutine

Purpose

Closes an object.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void closeobj( )
```

FORTRAN Syntax

```
SUBROUTINE CLOSEO
```

Description

The **closeobj** subroutine closes an open object definition. Use the **makeobj** subroutine to open a definition for a new object. All display list routines between the **makeobj** subroutine and the **closeobj** subroutine become part of the object definition.

Use the **editobj** subroutine to open an existing object for editing and the **closeobj** subroutine to end the editing session.

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

Note: This subroutine, *itself*, cannot be used to add to a display list.

Example

The example C language program **depthcue.c** uses the **closeobj** subroutine to specify the end of a graphical object definition.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Opening an object for editing with the **editobj** subroutine.

Creating an object with the **makeobj** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

cmode Subroutine

Purpose

Sets color map mode as the current mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void cmode( )
```

FORTRAN Syntax

```
SUBROUTINE CMODE
```

Description

The **cmode** subroutine sets color map mode as the current mode. This mode is the default.

If your workstation has more than 12 standard bitplanes, you can write color indexes between 0 and 4095 into the standard bitplanes. The drawing mode must be set to NORMALDRAW to write into the standard bitplanes (NORMALDRAW is the default drawing mode). The system must be in color map mode in order to draw into the overlay or underlay bitplanes.

You must call the **gconfig** subroutine after the **cmode** subroutine in order to have the **cmode** subroutine take effect. The system does not enter color map mode until the **gconfig** subroutine is called.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 will ignore the **cmode** subroutine when applied to windows created with the **winX** subroutine. The **cmode** subroutine can still be used with windows created with the **winopen** subroutine. See the **winX** subroutine for more detail.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Choosing a set of bitplanes for drawing with the **drawmode** subroutine.

Configuring the system with the **gconfig** subroutine.

Setting a display mode that bypasses the color map with the **RGBmode** subroutine.

Working in Color Map and RGB Modes.

---

cmov Subroutine

Purpose

Updates the current text character position.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

void cmov2(Coord  x, Coord  y)
```

```
void cmov2i(Icoord  x, Icoord  y)

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

FORTRAN Syntax

```
SUBROUTINE CMOV(x, y, z)
REAL x, y, z
SUBROUTINE CMOVI(x, y, z)
INTEGER*4 x, y, z
SUBROUTINE CMOVS(x, y, z)
INTEGER*2 x, y, z
SUBROUTINE CMOV2(x, y)
REAL x, y
SUBROUTINE CMOV2I(x, y, z)
INTEGER*4 x, y
SUBROUTINE CMOV2S(x, y, z)
INTEGER*2 x, y
```

> **Note:** For FORTRAN users, the INTEGER*2 versions of this subroutine, **CMOVS** and **CMOV2S**, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking one of the short versions of this subroutine with an integer constant will result in unexpected behavior.

All of the functions are essentially 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.

Description

The **cmov** subroutine moves the current character position to a specified point (just as the **move** subroutine sets the current graphics position). The **cmov** subroutine transforms the specified modeling coordinates into screen coordinates, which become the new character position. If the transformed point is outside the viewport, the character position is undefined.

The **cmov** subroutine does not affect the current graphics position.

Parameters

x Specifies the *x* location of the point (in modeling coordinates) to which to move the current character position.

y Specifies the *y* location of the point (in modeling coordinates) to which to move the current character position.

z Specifies the *z* location of the point (in modeling coordinates) to which to move the current character position. (This parameter is not used by the 2-D subroutines.)

Example

The example C language program **curved.c** uses the **cmov2** subroutine to move the character position to a specific location before drawing text.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.

/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a string of raster characters on the screen with the **charstr** subroutine.

Moving the current graphics position with the **move** subroutine.

Graphics Library Overview, Creating Text Characters, Using Viewports and Screenmasks, and Working with Coordinate Systems.

---

color or colorf Subroutine

Purpose

Sets the current color in color map mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void  color(Colorindex  color)

void  colorf(Float32  color)
```

FORTRAN Syntax

```
SUBROUTINE COLOR(color)
INTEGER*4 color
SUBROUTINE COLORF(color)
REAL color
```

Description

The **color** subroutine selects a color from the color map, and sets that color as the default in the current drawing mode. For example, if the drawing mode is NORMALDRAW (the default), the color index is written to the standard bitplanes during all drawing routines.

In NORMALDRAW mode, the *color* parameter allows you to access up to 12 bitplanes in onemap mode and up to 8 bitplanes in multimap mode. However, since the 8-bit High-Performance 3-D Color Graphics Processor has only one 8-bit main frame buffer, the 12-bit onemap mode is not available on this machine.

In OVERDRAW and UNDERDRAW modes, 0, 2, or 4 bitplanes are accessible.

In alternate drawing modes such as OVERDRAW the **color** subroutine serves the same function as in NORMALDRAW, except that different bitplanes are used, and a separate, smaller map is indexed.

The **colorf** subroutine is identical to the **color** subroutine, 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 indexes with fractional precision

yield more precise shading results using the **colorf** subroutine than with the **color** subroutine. The results of these subroutines are indistinguishable when drawing with FLAT shading.

The **color** and **colorf** subroutines serve no purpose in RGB mode because the RGB components of the color are written directly to the bitplanes.

Parameter

color Specifies a color index (0 to 4095 in onemap mode, 0 to 255 in multimap mode).

Example

The example C language program **boxcirc.c** uses the **color** subroutine to set the color for subsequent drawing routines.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Choosing a set of bitplanes for drawing with the **drawmode** subroutine.

Returning the current color with the **getcolor** subroutine.

Getting a copy of the RGB values for a color map entry with the **getmcolor** subroutine.

Changing a color map entry with the **mapcolor** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Setting a display mode that bypasses the color map with the **RGBmode** subroutine.

Gaining write access to a subset of available bitplanes with the **RGBwritemask** subroutine.

Gaining write access to the available bitplanes with the **writemask** subroutine.

Graphics Library Overview, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes.

---

compactify Subroutine

Purpose

Unfragments and compacts the memory storage of an object.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void compactify(Int32  object)
```

FORTRAN Syntax

```
SUBROUTINE COMPAC(object)
INTEGER*4 object
```

Description

The **compactify** subroutine unfragments and compacts the memory storage of an object.

Using the object editing subroutines to modify an open object definition can fragment the memory storage for the definition. The **compactify** subroutine eliminates wasted space, but calling it is rarely necessary because the **closeobj** subroutine automatically compacts an object definition that is too fragmented.

Because the **compactify** subroutine requires a significant amount of time to execute, calls to it should be avoided unless storage space is critical.

Note: This editing subroutine itself cannot be added to a display list.

Parameter

object Specifies the identifier for the object to be compacted.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the minimum object size in memory with the **chunksize** subroutine.

Closing an object with the **closeobj** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

concave Subroutine

Purpose

Forces the system to draw accurate concave polygons.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void concave(Int32  bool)
```

FORTRAN Syntax

```
SUBROUTINE CONCAV(bool)
LOGICAL bool
```

Description

The **concave** subroutine forces the system to draw accurate concave polygons. When the *bool* parameter is True (1), the system draws accurate concave polygons. When the value of the *bool* parameter is False (0) (the default), the results of drawing a concave polygon are unpredictable.

The system draws polygons significantly faster if checking for concavity is turned off. Therefore, unless you specifically want to draw concave polygons, you should generally operate with the *bool* parameter set to False.

Parameter

bool Specifies a value for the state of concavity. The settings for the *bool* parameter are:

True System draws accurate concave and convex polygons.

False System draws only accurate convex polygons. This is the default.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

On the POWERgraphics GXT1000, the value of the **concave** flag is ignored when drawing into the overlay planes.

If the **concave** flag has been set to TRUE, then the GXT1000 limits the number of vertices that can appear between a **bgnline** and **endline** pair to 255 vertices. Otherwise, an unlimited number of vertices can be specified.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a vertex-based polygon with the **bgnpolygon** subroutine.

Specifying the next point in a polygon with the **pdr** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Drawing a filled polygon with the **polf** subroutine.

Drawing a polygon with the **poly** subroutine.

Graphics Library Overview, Setting Pipeline Options, Drawing Rectangles, Circles, Arcs, and Polygons.

---

cpack Subroutine

Purpose

Specifies RGBA color with a single packed 32-bit integer.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void cpack(unsigned Int32  pack)
```

FORTRAN Syntax

```
SUBROUTINE CPACK(pack)
INTEGER*4 pack
```

Description

The **cpack** subroutine changes the current RGBA (red, green, blue, alpha) values. It is valid only in RGB mode. Red is the least significant byte in the packed integer, then green, blue, and alpha. Components must range from 0 through 255. For example,

```
cpack(0xFF004080);
```

sets red to `0x80`, green to `0x40`, blue to `0x0`, and alpha to `0xFF`. On systems without alpha bitplanes, set the alpha bit values to zero.

Note that the **cpack** subroutine can be used while a lighting model is active.

Parameter

pack Specifies a packed integer containing the RGBA (red, green, blue, alpha) values to assign as the current color.

Example

To clear the screen to black, then set the color to white, the example C language program **localatten.c** calls the **cpack** subroutine with a value of `0` (black) before a **clear** subroutine, then calls the **cpack** subroutine with a value of `0xFFFFFF` (white).

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current color in RGB mode with the **c** subroutine.

Setting the current color in color map mode with the **color** subroutine.

Returning the current color in color map mode with the **getcolor** subroutine.

Returning the current color in RGB mode with the **gRGBcolor** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Graphics Library Overview, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes.

---

crv Subroutine

Purpose

Draws a cubic spline curve.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

FORTRAN Syntax

```
SUBROUTINE CRV(points)
REAL points(3,4)
```

Description

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

Parameter

points	Array containing the four points that define the curve. These must be 3-D points with *x, y,* and *z* coordinates for each point.

Example

The example C language program **curve1.c** uses the **crv** subroutine, after changing the curve basis and precision for each curve, to draw three curves.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a series of curve segments with the **crvn** subroutine.

Setting the current cubic spline curve basis matrix with the **curvebasis** subroutine.

Setting the number of line segments that draw a curve segment with the **curveprecision** subroutine.

Defining a cubic spline basis matrix with the **defbasis** subroutine.

Drawing a rational curve with the **rcrv** subroutine.

Drawing a series of rational curve segments with the **rcrvn** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

crvn Subroutine

Purpose

Draws a series of curve segments.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void crvn(Int32  n, Coord  geom [ ][3])
```

FORTRAN Syntax

```
SUBROUTINE CRVN(n, geom)
INTEGER*4 n
REAL geom(3, n)
```

Description

The **crvn** subroutine 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.

Parameters

geom Specifies a matrix of 3-D points.
n Number of points in the matrix referenced by *geom*.

Example

The example C language program **curve2.c** uses the **crvn** subroutine to draw a curve with six control points.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a cubic spline curve with the **crv** subroutine.

Setting the current cubic spline curve basis matrix with the **curvebasis** subroutine.

Setting the number of line segments that compose a cubic spline curve with the **curveprecision** subroutine.

Defining a cubic spline basis matrix with the **defbasis** subroutine.

Drawing a rational curve with the **rcrv** subroutine.

Drawing a series of rational curve segments with the **rcrvn** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

curorigin Subroutine

Purpose

Sets the origin of a cursor.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void curorigin
(Int16  index, Int16  xorigin, Int16  yorigin)
```

FORTRAN Syntax

```
SUBROUTINE CURORI(index, xorigin, yorigin)
INTEGER*4 index, xorigin, yorigin
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **curorigin** subroutine sets the origin of a cursor glyph. The origin is the point on the cursor glyph that aligns with the current cursor valuators. The origin is specified in pixels from the lower left-hand corner of the cursor glyph. Before calling the **curorigin** subroutine, the cursor glyph must have been defined with the **defcursor** subroutine. The **curorigin** subroutine does not take effect until the **setcursor** subroutine is called.

The default origin for a user-defined cursor glyph is (0,0), that is, the lower left-hand corner.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

index	Specifies an index into the cursor table.
xorigin	Specifies the *x* distance of the origin relative to the lower left corner of the cursor.
yorigin	Specifies the *y* distance of the origin relative to the lower left corner of the cursor.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining the type and size of a cursor with the **curstype** subroutine.

Defining a cursor with the **defcursor** subroutine.

Setting the drawing mode to CURSORDRAW with the **drawmode** subroutine.

Setting the cursor characteristics with the **setcursor** subroutine.

Graphics Library Overview, Creating a Cursor, and Creating and Managing Windows.

curson or cursoff Subroutine

Purpose
Controls cursor visibility by window.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void curson( )
void cursoff( )
```

FORTRAN Syntax
```
SUBROUTINE CURSON
SUBROUTINE CURSOF
```

Description
The **curson** and **cursoff** subroutines control the visibility of the cursor in the current window. These subroutines control only the visibility of the cursor and do not disable or enable the cursor or mouse button click events inside the current window. The **curson** subroutine is the default.

Use the **getcursor** subroutine to find out if the cursor is visible.

Note: These subroutines cannot be used to add to a display list.

Example
The example C language program **text.c** uses **cursoff** subroutine to turn off the cursor while drawing. The program uses the **curson** subroutine to turn on the cursor after drawing.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Setting the origin of a cursor with the **curorigin** subroutine.

Defining the type and size of a cursor with the **curstype** subroutine.

Defining a cursor with the **defcursor** subroutine.

Returning the cursor characteristics with the **getcursor** subroutine.

Setting the cursor characteristics with the **setcursor** subroutine.

Graphics Library Overview, Creating a Cursor, and Creating and Managing Windows.

---

curstype Subroutine

Purpose
Defines the type and size of the cursor.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void curstype(Int32  type)
```

FORTRAN Syntax
```
SUBROUTINE CURSTY(type)
INTEGER*4 type
```

Description
The **curstype** subroutine defines the type and size of the cursor. The system supports five cursor types:

Type	**Size**	**Description**
C16X1	16x16 bit	No more than one color (default)
C16X2	16x16 bit	No more than three colors
C32X1	32x32 bits	No more than one color
C32X2	32x32 bits	No more than three colors
CCROSS	Full window	Cross-hair

After calling the **curstype** subroutine, call the **defcursor** subroutine to assign a numeric value to the cursor and specify the cursor bitmap.

The cross-hair cursor is formed with a horizontal line and a vertical line (each one pixel wide) that extend completely across the window. Its origin (15,15) is at the intersection of the two lines. It is a one-color cursor that uses color number one.

Parameter
type Specifies one of five values that describe the cursor. (The values are the same for both C and FORTRAN.)

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the origin of a cursor with the **curorigin** subroutine.

Controlling cursor visibility by window with the **curson** or **cursoff** subroutine.

Defining a cursor with the **defcursor** subroutine.

Setting the drawing mode to CURSORDRAW with the **drawmode** subroutine.

Changing a color map entry with the **mapcolor** subroutine.

Setting the cursor characteristics with the **setcursor** subroutine.

Graphics Library Overview, Creating a Cursor, and Creating and Managing Windows.

---

curvebasis Subroutine

Purpose

Sets the current cubic spline curve basis matrix.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void curvebasis(Int32  basis_id)
```

FORTRAN Syntax

```
SUBROUTINE CURVEB(basis_id)
INTEGER*4 basis_id
```

Description

The **curvebasis** subroutine sets a basis matrix as defined by the **defbasis** subroutine 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.

Parameter

basis_id	Specifies the basis identifier of the basis matrix to use when drawing a curve.

Example

The example C language program **curve1.c** uses the **curvebasis** subroutine to select each of three previously defined basis matrices, then draws a curve using each basis matrix.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a cubic spline curve with the **crv** subroutine.

Drawing a series of curve segments with the **crvn** subroutine.

Setting the number of line segments that compose a cubic spline curve with the **curveprecision** subroutine.

Defining a cubic spline basis matrix with the **defbasis** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

curveit Subroutine

Purpose

Draws a curve segment by iterating the forward difference matrix.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void curveit(Int16  count)
```

FORTRAN Syntax

```
SUBROUTINE CURVEI(count)
INTEGER*4 count
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **curveit** subroutine repeats the forward difference algorithm with the current matrix (the one on top of the matrix stack) for the number of times assigned by the *count* parameter. Each iteration draws one of the line segments that approximate the curve. The **curveit** subroutine accesses low-level hardware capabilities for curve drawing.

Parameter

count Specifies the number of times to repeat the current matrix.

Example

The example C language program **curve3.c** uses the **curveit** subroutine to draw a Bezier curve segment after building the correct transformation matrix.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a cubic spline curve with the **crv** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

curveprecision Subroutine

Purpose

Sets the number of line segments that draw a curve segment.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void curveprecision(Int16  nsegments)
```

FORTRAN Syntax

```
SUBROUTINE CURVEP(nsegments)
INTEGER*4 nsegments
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **curveprecision** subroutine sets the number of line segments used to draw a curve segment. Whenever the **crv**, **crvn**, **rcrv**, or **rcrvn** subroutines execute, a number of straight line segments, represented by the *nsegments* parameter, approximates each curve segment. Increasing the value of *nsegments* makes a smoother curve, but also increases the drawing time.

Parameter

nsegments Specifies the number of line segments to use in drawing the curve segment.

Example

The example C language program **curve1.c** uses the **curveprecision** subroutine to draw three curves so that each curve is approximated by 20 line segments.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a cubic spline curve with the **crv** subroutine.

Drawing a series of curve segments with the **crvn** subroutine.

Setting the current cubic spline curve basis matrix with the **curvebasis** subroutine.

Drawing a rational curve with the **rcrv** subroutine.

Drawing a series of rational curve segments with the **rcrvn** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

cyclemap Subroutine

Purpose

Cycles through color maps at a specified rate.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void cyclemap(Int16  duration, Int16  map, Int16  nextmap)
```

FORTRAN Syntax

```
SUBROUTINE CYCLEM(duration, map, nextmap)
INTEGER*4 duration, map, nextmap
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **cyclemap** subroutine initiates cycling between two small color maps. The system, when using the current color map *map*, waits the number of vertical retraces specified by the *duration* parameter before switching to the next map to be used (*nextmap*).

You can use the **cyclemap** subroutine used to chain together several color maps into a loop, thus initiating a cycling or blinking that continues indefinitely. The system loops automatically through the chain until cycling is disabled.

You can eliminate a cyclemap entry by calling the **cyclemap** subroutine with the *duration* parameter set to 0.

This subroutine must be used in multimap mode.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

duration	Specifies duration for the current map in vertical retraces.
map	Specifies the number of the current map.
nextmap	Specifies the number of the next map to use.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER GXT1000 adapter does not support the **cyclemap** subroutine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Changing a color map entry at a selectable rate with the **blink** subroutine.

Reconfiguring the system with the **gconfig** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Working in Color Map and RGB Modes, Creating Animated Scenes.

---

czclear Subroutine

Purpose

Clears the color bitplanes and the z-buffer simultaneously.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void czclear(Int32  cval, Int32  zval)
```

FORTRAN Syntax

```
SUBROUTINE CZCLEA(cval, zval)
INTEGER*4 cval, zval
```

Description

The **czclear** subroutine simultaneously clears the color bitplanes to the value of the *cval* parameter and the z-buffer to the value of the *zval* parameter. The **czclear** subroutine clears all active color bits (8 or 12 in color map mode, 24 in RGB mode), and all 24 z-buffer bits. Pattern 0 (zero) is always used, regardless of the current pattern specification. The system ignores the current writemask.

Only the screen area inside the current screenmask is cleared. The screenmask cannot be made larger than the window. By default, the screenmask is exactly the same size as the window. Use the **scrmsk** subroutine to change the size of the screenmask.

> **Note:** The **viewport** subroutine resets the screenmask to be precisely the same size as the viewport.

In RGB mode, the *cval* parameter requires a packed integer of the same format used by the **cpack** subroutine, 0x*aaggbbrr*, where *rr* is the red value, *gg* is the green value, *bb* is the blue value, and *aa* is the alpha value. In color map mode, the *cval* parameter requires an index into the current color map, so that only the bottom 8 or 12 bits are significant.

Whenever you need to clear both the z-buffer and the color bitplanes to constant values at the same time, use the **czclear** subroutine. The **czclear** subroutine executes as fast as, or faster than, the **clear** and **zclear** subroutines called sequentially.

The **czclear** subroutine can be called whether z-buffer mode is on or off. The current color does not change.

> **Note:** The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See Hardware Considerations in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.)

Parameters

cval Specifies the color to which to clear the color bitplanes.
zval Specifies the depth to which to clear the z-buffer. The *zval* parameter has the following values:

Values for the zval parameter

Min	Max
-0x800000	0x7FFFFF

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Clearing to the screenmask with the **clear** subroutine.

Setting the current color as a packed 32-bit integer using the **cpack** subroutine.

Enabling and disabling the z-buffer for storing depth information with the **zbuffer** subroutine.

Clearing the z-buffer with the **zclear** subroutine.

Specifying the function used for depth comparison with the **zfunction** subroutine.

Understanding the Hardware Used by GL, Configuring the Frame Buffer, and Working in Color Map and RGB Modes.

Clearing, Resetting, and Initiliazing GL, Writemasks and Logical Operations.

---

defbasis Subroutine

Purpose

Defines a cubic spline basis matrix.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void defbasis(Int32  id, Matrix  mat)
```

FORTRAN Syntax

```
SUBROUTINE DEFBAS(id, mat)
INTEGER*4 id
REAL mat(4,4)
```

Description

The **defbasis** subroutine assigns a basis matrix identifier for use by subroutines that generate curves and patches. Use the basis matrix identifier in subsequent calls to the **curvebasis** and **patchbasis** subroutines.

Note: This subroutine cannot be used to add to a display list.

Parameters

id	Specifies the basis identifier to assign to the matrix.
mat	Specifies the matrix to which to assign the basis identifier.

Example

The example C language program **curve1.c** uses the **defbasis** subroutine to define three basis matrices for drawing curves.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a cubic spline curve with the **crv** subroutine.

Drawing a series of curve segments with the **crvn** subroutine.

Setting the current cubic spline curve basis matrix with the **curvebasis** subroutine.

Setting the number of line segments that draw a curve segment with the **curveprecision** subroutine.

Drawing surface patches with the **patch** subroutine.

Setting the current spline surface basis matrices with the **patchbasis** subroutine.

Setting the number of curves used to represent a patch with the **patchcurves** subroutine.

Setting the precision at which curves are drawn with the **patchprecision** subroutine.

Drawing a rational curve with the **rcrv** subroutine.

Drawing a series of rational curve segments with the **rcrvn** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

defcursor Subroutine

Purpose
Defines a cursor glyph.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void defcursor(Int32  index, Uint16 * cursor)
```

FORTRAN Syntax
```
SUBROUTINE DEFCUR(index, cursor)
INTEGER*4 index
INTEGER*2 cursor(1);
```

Description
The **defcursor** subroutine defines a cursor glyph with the specified name and bitmap. Call the **curstype** subroutine to set the type and size of cursor before calling the **defcursor** subroutine.

The bitmap can be 16x16 or 32x32 and one or two layers deep. If the cursor has been defined by the **curstype** subroutine as type C16X2 or C32X2, the bitmap array is two layers deep.

By default, the bitmap cursor origin is at (0,0), its lower-left corner. Use the **curorigin** subroutine to reset the cursor origin (the position controlled by valuators attached to the cursor and the position used for the picking region).

To replace the bitmap assigned to a cursor constant, call the **defcursor** subroutine again, keeping the *index* parameter the same but changing the bitmap *cursor* parameter.

By default, a cross-hair cursor origin is at (15,15), the intersection of its two lines.

By default, an arrow is defined as cursor 0 (zero) and cannot be redefined.

> **Note:** This subroutine cannot be used to add to a display list.

The ordering of the values in the *cursor* array is left-to-right, then bottom-to-top, low-order bit plane, then high-order bit plane. This format is the same as that used by the **defrasterfont** subroutine. In other words, the first 16-bit short word specifies the lower-left bits of the image. If the cursor is 32 bits wide, the next short word specifies the lower-right bits. For a 16-bit wide cursor, the second short word specifies the second row from the bottom, the third word specifies the third row, and so on. If the cursor is a three-color cursor, place the bitmap for the high bit after the data for the low bit. The four different cursor configurations make use of different array sizes, as follows:

Cursor Type	Array Size
C16X1	16

Cursor Type	Array Size
C16X2	32
C32X1	64
C32X2	128

Parameters

index Specifies a number to identify a cursor to other cursor routines.
cursor Specifies the bitmap for the cursor. For the cross-hair cursor type, this parameter is disregarded.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the origin of a cursor with the **curorigin** subroutine.

Controlling cursor visibility by window with the **curson** or **cursoff** subroutine.

Defining the type and size of a cursor with the **curstype** subroutine.

Returning the cursor characteristics with the **getcursor** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Setting the cursor characteristics with the **setcursor** subroutine.

Graphics Library Overview, Creating and Managing Windows.

Creating a Cursor.

---

deflinestyle Subroutine

Purpose

Defines a linestyle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void deflinestyle(Int32  index, Linestyle  linestyle)
```

FORTRAN Syntax

```
SUBROUTINE DEFLIN(index, linestyle)
INTEGER*4 index, linestyle
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **deflinestyle** subroutine defines a linestyle, which is a write-enabled bit pattern that is applied when lines are drawn. The least significant bit of the linestyle is applied first.

You can define up to 2(16) (65536) linestyles. By default, index 0 contains the pattern `0xFFFF`, which draws solid lines. Index 0 cannot be redefined. A linestyle can be redefined by reusing an index.

Notes:

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)
2. This subroutine cannot be used to add to a display list.

Parameters

index	Specifies the index into a table of linestyles.
linestyle	Specifies a 16-bit pattern stored in the linestyle table at *index*.

Example

The example C language program **curved.c** uses the **deflinestyle** subroutine to define a dashed line.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

On some display adapters, notably the High-Performance 3-D Graphics Processor, there is a performance penalty for drawing non-solid lines.

On the POWER GXT1000 adapter, the line stipple is reset for every line segment between a **bgnline** and **endline** subroutine pair and a **bgnclosedline** and **endclosedline** subroutine pair.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a cursor with the **defcursor** subroutine.

Defining a pattern with the **defpattern** subroutine.

Defining bitmaps for a raster font with the **defrasterfont** subroutine.

Returning the current linestyle with the **getlstyle** subroutine.

Setting the repeat factor for the current linestyle with the **lsrepeat** subroutine.

Selecting a linestyle with the **setlinestyle** subroutine.

Graphics Library Overview, Drawing Wire Frame Curves and Surface Patches, Drawing NURBS Curves and Surfaces, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

defpattern Subroutine

Purpose

Defines patterns used in area fills.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void defpattern
(Int32  index, Int16  size, Int16 * mask)
```

FORTRAN Syntax

```
SUBROUTINE DEFPAT(index, size, mask)
INTEGER*4 index, size
INTEGER*2 mask((size*size)/16)
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **defpattern** subroutine defines patterns for filling areas such as polygons, rectangles, and curves. By default, pattern 0 is a 16x16-bit solid pattern. Index 0 cannot be changed. The system stores the pattern in a pattern table at *index*.

The pattern is described from left to right and bottom to top, just as characters are described in a raster font. The **defpattern** subroutine allows you to define an arbitrary pattern and assign it an index identifier. You can later reference this pattern with the **setpattern** subroutine via this identifier. Patterns, cursors, and fonts are available to all windows when using multiple windows.

Notes:

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)
2. This subroutine cannot be used to add to a display list.

Parameters

index	Specifies the constant to use as an identifier for the pattern described by the *mask* parameter. This constant is used as an index into a table of patterns.
mask	Specifies the array of short integers that form the actual bit pattern.
size	Specifies the size of the pattern: for example, 16 specifies a 16x16-bit pattern.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

On the High-Performance 3-D Graphics Processor, there is a performance penalty for non-solid patterns. The High-Performance 3-D Graphics Processor supports only 16x16-bit, 32x32-bit, or 64x64-bit patterns.

The POWER Gt4 and POWER Gt4x adapters support arbitrary-sized patterns, from 1x1 to 64x64 bits.

The Supergraphics Processor Subsystem and the POWER GXT1000 adapter support The Supergraphics Processor Subsystem supports only 16x16-bit and 32x32-bit patterns.The getgdesc(GD_MAX_PATTERN_SIZE) call can be used to determine the largest supported tile pattern.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a cursor with the **defcursor** subroutine.

Defining bitmaps for a raster font with the **defrasterfont** subroutine.

Returning the index of the current fill pattern with the **getpattern** subroutine.

Selecting a pattern for filling polygons and rectangles with the **setpattern** subroutine.

Graphics Library Overview, Drawing Wire Frame Curves and Surface Patches, Drawing NURBS Curves and Surfaces, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

defpup Subroutine

Purpose

Defines a pop-up menu.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

Int32 defpup(Char8 * *string* **[, Int32** *arguments* **...])**

FORTRAN Syntax

None: available only in C.

Description

The **defpup** subroutine defines a pop-up menu and returns a menu identifier.

Note: This subroutine cannot be used to add to a display list.

Parameters

string Specifies the pointer to the text to add as a menu item. There are seven menu item type flags for optional pairing with each menu item:

- **%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** flag. You must specify the invoked routine in the *arguments* 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 the **%F** flag.
- **%f** Invokes a routine when this particular menu item is selected. You must specify the invoked routine in the *arguments* 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 the **%f** flag. If you have also used the **%F** flag within this menu, then the result of the **%f** flag is passed as a parameter of the **%F** flag.
- **%l** Adds a line under the current entry. This is useful in providing visual clues for grouping 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 *arguments* parameter.
- **%n** Like the **%f** flag, this flag invokes a routine when the user selects this menu item. However, the **%n** flag differs from the **%f** flag in that it ignores the routine (if any) specified by the **%F** flag. 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 the **%f** flag.
- **%x***n* Assigns a numeric value to this menu item. This value overrides the default position-based value assigned to this menu item (third item=3). You must enter the numeric value as the part of the text string specified by the *n* parameter. Do not use the *arguments* parameter to specify the numeric value.

arguments Specifies an optional set of arguments. Each argument expects the command or submenu to be assigned to this menu item. The *arguments* parameter can be used as many times as necessary.

Return Value

The menu identifier of the menu just defined.

Example

The example C language program **curved.c** uses the **defpup** subroutine to define a pop-up menu.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

File

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.

Related Information

Adding an item to an existing pop-up menu with the **addtopup** subroutine.

Displaying a pop-up menu with the **dopup** subroutine.

Deallocating a pop-up menu and its data structures with the **freepup** subroutine.

Allocating and initializing a structure for a new pop-up menu with the **newpup** subroutine.

Enabling or disabling a given pop-up entry with the **setpup** subroutine.

Graphics Library Overview and Creating and Managing Pop-Up Menus.

---

defrasterfont Subroutine

Purpose

Defines bitmaps for a raster font.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void defrasterfont
(Int32  index, Int16  height,
Int16  numchars, Fontchar  chars [ ],
Int16  numraster, Int16  raster [ ])
```

FORTRAN Syntax

SUBROUTINE DEFRAS(*index***,** *height***,** *numchars***,** *chars***,** *numraster***,** *raster***)**

INTEGER *4 *index***,** *height***,** *numchars***,** *numraster*

INTEGER*2 *raster***(***numraster***),** *chars***(4*** *numchars***)**

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **defrasterfont** subroutine defines a raster font.

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, linestyles, cursors, and fonts are available to all windows when using multiple windows. The assumed code set table encoding is locale dependent. In most locales, this is an ISO8859 encoding.

Notes:

1. This subroutine cannot be used to add to a display list.
2. The hardest part of creating a new raster font is generating a bitmap for each character. You may want to write a graphically oriented tool for creating the bitmaps expected by the *raster* parameter.

Parameters

index Specifies a constant 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.

height Specifies the maximum height (in pixels) for a character.

numchars Specifies the number of characters in this font.

chars Specifies an array of character description structures of type **Fontchar**. The **Fontchar** structure is defined in the **/usr/include/gl/gl.h** file as:

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

The character description structures are as follows:

offset Specifies the element of the *raster* array at which the bitmap for this character starts.

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

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

xoff Specifies the bitmap columns between the start of the character's bitmap and the start of the character.

yoff Specifies the number of rows between the character's baseline and the bottom of the bitmap. For characters with descenders (for example, g) this value is a negative number. For characters that rest entirely on the baseline, this value is zero.

width Specifies 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.)

numraster Specifies the length of the *raster* parameter array.

raster Specifies a one-dimensional array that contains all the bitmaps 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.

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 records in the *chars* array.

Example

The example C language programs **curved.c** and **font3.c** use the **defrasterfont** subroutine to define a new raster font.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

This subroutine is not available for National Language Support.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a string of raster characters on the screen with the **charstr** subroutine.

Updating the current character position with the **cmov** subroutine.

Selecting a raster font with the **font** subroutine.

Returning the current character position with the **getcpos** subroutine.

Returning the baseline extent of the longest character descender with the **getdescender** subroutine.

Returning the current raster font number with the **getfont** subroutine.

Returning the maximum character height in the current raster font with the **getheight** subroutine.

Using an AIXwindows font to define a raster font with the **loadXfont** subroutine.

Returning the width of the specified text string with the **strwidth** subroutine.

National Language Support Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Creating Text Characters.

---

delobj Subroutine

Purpose

Deletes an object.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void delobj(Int32  object)
```

FORTRAN Syntax

```
SUBROUTINE DELOBJ(object)
INTEGER*4 object
```

Description

The **delobj** subroutine deletes an entire object, freeing the entire display list and all associated tags. The object identifier becomes undefined and unused.

Note: This subroutine cannot be used to add to a display list.

Parameter

object Specifies the identifier of the object to delete.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Deleting tags from a display list with the **deltag** subroutine.

Opening an object for editing with the **editobj** subroutine.

Creating an object with the **makeobj** subroutine.

Deleting a routine from an object with the **objdelete** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

deltag Subroutine

Purpose

Deletes tags from objects.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void deltag(Int32  tag)
```

FORTRAN Syntax

```
SUBROUTINE DELTAG(tag)
INTEGER*4 tag
```

Description

The **deltag** subroutine removes the tag from the object currently open for editing. The STARTTAG and ENDTAG special tags cannot be deleted.

Note: This editing subroutine itself cannot be used to add to a display list.

Parameter

tag Specifies the tag to delete.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Opening an object for editing with the **editobj** subroutine.

Marking a location in a display list with the **maketag** subroutine.

Creating Objects (Display Lists).

Graphics Library Overview and Creating Objects (Display Lists).

---

depthcue Subroutine

Purpose

Turns depth-cueing on and off.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void depthcue(Int32  mode)
```

FORTRAN Syntax

```
SUBROUTINE DEPTHC(mode)
LOGICAL mode;
```

Description

The **depthcue** subroutine turns depth-cueing on or off. If the value of the *mode* parameter is TRUE, depth-cueing is enabled, and all lines, points, characters, and polygons drawn by the system are depth-cued. This means that the color of the lines, points, characters, or polygons are determined by the *z* values and the range of color indexes specified by the **lshaderange** or **lRGBrange** subroutine determines the color of the lines, points, characters, or polygons.

The *z* values, whose range is set by the **lsetdepth** subroutine, are mapped linearly into the range of color indexes. In this mode, lines that vary greatly in *z* value span the range of colors specified by the **lshaderange** subroutine.

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

Parameter

mode Specifies a value indicating OFF or ON state of depth-cueing. Values for the *mode* parameter are as follows:
1 = True (ON)
0 = False (OFF)

Example

The example C language program **depthcue.c** uses the **depthcue** subroutine to turn on depth-cueing.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Indicating whether depth-cue mode is on or off with the **getdcm** subroutine.

Setting the range of RGB colors used for depth-cueing with the **lRGBrange** subroutine.

Setting the viewport depth range with the **lsetdepth** subroutine.

Setting the range of color indexes used for depth-cueing with the **lshaderange** subroutine.

Graphics Library Overview, Performing Depth-Cueing, and Working in Color Map and RGB Modes.

---

dopup Subroutine

Purpose

Displays the specified pop-up menu.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 dopup(Int32  popup)
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION DOPUP(popup)
INTEGER*4 popup
```

Description

The **dopup** subroutine displays the specified pop-up menu until the user makes a selection. If the calling program has the input focus, the specified menu is displayed and the system 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, the **dopup** subroutine returns a value of -1 (negative one).

The **dopup** subroutine manipulates events in the event queue. If the **dopup** subroutine is called as a result of a left mouse or middle mouse button press event, the menu remains posted until the right mouse button is pressed to make a selection. The **dopup** subroutine removes a pair of right mouse press and release events from the event queue (if the right mouse was queued), and also removes the left mouse or middle mouse button release event.

If the **dopup** subroutine is called as a result of a right mouse button press event, the menu remains posted until the right mouse button is released. The menu entry selected is the entry that was highlighted when the right mouse button was released. The **dopup** subroutine removes the right mouse button release event from the event queue.

If no selection is made (that is, if the user releases the right mouse button off the pop-up menu) no menu entry selection is made, and the **dopup** subroutine returns a value of -1. The right mouse button release event is removed from the queue.

In more complicated event sequences, all mouse button-up events up to and including the first right mouse button-up event are removed. If the user accidently presses other mouse buttons before making a pop-up menu selection, *orphaned* button down events (button down events without matching button up events) may remain in the event queue.

When the menu is defined, the **defpup** or **addtopup** subroutine specifies the list of menu entries and their corresponding actions.

> **Note:** This subroutine cannot be used to add to a display list.

Parameter

popup Specifies which pop-up menu to display.

Example

The example C language program **curved.c** uses the **dopup** subroutine to display a pop-up menu.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Adding an item to an existing pop-up menu with the **addtopup** subroutine.

Defining a pop-up menu with the **defpup** subroutine.

Deallocating a pop-up menu and its data structures with the **freepup** subroutine.

Allocating and initializing a structure for a new pop-up menu with the **newpup** subroutine.

Enabling or disabling a given pop-up entry with the **setpup** subroutine.

Graphics Library Overview and Creating and Managing Pop-Up Menus.

---

doublebuffer Subroutine

Purpose

Sets the display mode to double buffer mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void doublebuffer( )
```

FORTRAN Syntax

```
SUBROUTINE DOUBLEBUFFER
```

> **Note:** In using FORTRAN syntax, it is necessary to spell out DOUBLEBUFFER because the word *double* is a reserved word.

Description

The **doublebuffer** subroutine reorganizes the frame buffer bitplanes into a pair of frame buffers, the front buffer and the back buffer. In double buffer mode, only the front buffer is displayed. Drawing routines normally update only the back bitplanes; the **frontbuffer** and **backbuffer** subroutines can override the default.

The actual repartitioning of the frame buffer into double buffer mode does not occur until the **gconfig** subroutine is called. The **gconfig** subroutine sets **frontbuffer** = False and **backbuffer** = True in double buffer mode.

To determine the number of bitplanes available in each buffer, use the **getgdesc** subroutine.

Note: This subroutine cannot be used to add to a display list.

Example

The example C language program **doublebuff.c** sets the display mode for double buffering with the **doublebuffer** subroutine to allow smooth motion when moving and redrawing a cube.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling updating in the back buffer with the **backbuffer** subroutine.

Reconfiguring the system with the **gconfig** subroutine.

Finding out which buffers are enabled for writing with the **getbuffer** subroutine.

Returning the current display mode with the **getdisplaymode** subroutine.

Returning the number of bitplanes available in each buffer with the **getgdesc** subroutine.

Enabling updating in the front buffer with the **frontbuffer** subroutine.

Writing to and displaying all bitplanes with the **singlebuffer** subroutine.

Exchanging the font and back buffers with the **swapbuffers** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Configuring the Frame Buffer, Creating Animated Scenes.

---

draw Subroutine

Purpose

Draws a line.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

void draw2
(Coord  x, Coord  y)

void draw2i
(Icoord  x, Icoord  y)

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

FORTRAN Syntax

```
SUBROUTINE DRAW(x, y, z)
REAL x, y, z
SUBROUTINE DRAWI(x, y, z)
INTEGER*4 x, y, z
SUBROUTINE DRAWS(x, y, z)
INTEGER*2 x, y, z
SUBROUTINE DRAW2(x, y)
REAL x, y
SUBROUTINE DRAW2I(x, y)
INTEGER*4 x, y
SUBROUTINE DRAW2S(x, y)
INTEGER*2 x, y
```

> **Note:** For FORTRAN users, the INTEGER*2 versions of this subroutine, **DRAWS** and **DRAW2S**, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking one of the short versions of this subroutine with an integer constant will result in unexpected behavior.

Description

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

The **draw** subroutine updates the current graphics position to the specified point.

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

The six different forms for the **draw** subroutine are as follows:

	2-D	3-D

Int16	draw2s	draws
Int32	draw2i	drawi
float	draw2	draw

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **draw** expects real coordinates, **drawi** expects integer coordinates, and **draws** expects short integer coordinates. In addition, the **draw2** routines assume a 2-D point instead of a 3-D point.

Parameters

x Specifies the *x* coordinate of the point to which to draw a line segment.
y Specifies the *y* coordinate of the point to which to draw a line segment.
z Specifies the *z* coordinate of the point to which to draw a line segment (not used by 2-D subroutines).

Example

The example C language program **depthcue.c** uses the **drawi** subroutine to draw the edges of a cube.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Moving the current graphics position to a specified point with the **move** subroutine.

Drawing a point with the **pnt** subroutine.

Drawing a relative line with the **rdr** subroutine.

Moving the current graphics position to a point relative to the current point with the **rmv** subroutine.

Graphics Library Overview, Drawing with Move-Draw Style Subroutines, Performing Depth-Cueing in GL, Setting Drawing Attributes, and Working in Color Map and RGB Modes.

drawmode Subroutine

Purpose

Specifies the target frame buffer for the drawing subroutines.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void drawmode(Int32 *mode***)**

FORTRAN Syntax

SUBROUTINE DRAWMO(*mode***)**
INTEGER*4 *mode*

Description

The **drawmode** subroutine reconfigures the system and redirects the target of a number of GL drawing and attribute subroutines. The affected routines depend on the mode that the user chooses. Calls to the **color**, **getcolor**, **writemask**, **getwritemask**, **mapcolor**, and **getmcolor** subroutines affect only the current drawing mode. In cursor mode, only the **getmcolor** and **mapcolor** subroutines perform a function.

Note: This subroutine cannot be used to add to a display list.

Parameter

mode Specifies the drawing mode identifier. The mode constants for both C and FORTRAN and their functions are displayed in the following table:

Mode Constants		
C	**FORTRAN**	**Description**
UNDERDRAW	UNDRDR	Redirects drawing into the background (underlay) bitplanes.
NORMALDRAW	NORMDR	Redirects all drawing into the mainframe buffer bitplanes.
OVERDRAW	OVRDRW	Redirects drawing into the foreground (overlay) bitplanes.
PUPDRAW	PUPDRW	Redirects **mapcolor** and **getmcolor** subroutines to affect only the pop-up menus.
CURSORDRAW	CURSDR	Redirects **mapcolor** and **getmcolor** subroutines to affect only the cursor.

The following provides additional descriptions for the preceding modes:

Mode **Description**

UNDERDRAW The line drawing and polygon drawing routines (both begin-end style and move-draw style) draw into the underlay planes rather than the main frame buffer. The pixmap transfer subroutines (the **rectread**, **rectwrite**, and **rectcopy** subroutines) also draw into the underlays. All of the current attributes are used during drawing, except the color and the writemask. Each drawing mode has a separate current color and current writemask, which are saved and restored when that drawing mode is exited and entered.

Drawing into the underlay planes can only be done in colorindex mode. The system automatically goes into colorindex mode when the UNDERDRAW mode is entered. Lighting and NURBS subroutines do not work correctly in UNDERDRAW mode. Because of the small number of bitplanes, only flat shading is possible; Gouraud shading does not work. The z-buffer is not updated when in UNDERDRAW mode.

The system must have been configured to contain underlay planes before the UNDERDRAW mode can be entered. Underlay planes may be configured by calling the **underlay** subroutine followed by the **gconfig** subroutine.

NORMALDRAW All drawing occurs in the main frame buffer. NORMALDRAW is the default drawing mode.

Mode	Description
OVERDRAW	The line drawing and polygon drawing routines (both begin-end style and move-draw style) draw into the overlay planes rather than the main frame buffer. The pixmap transfer subroutines (the **rectread**, **rectwrite**, and **rectcopy** subroutines) also draw into the overlays. All of the current attributes are used during drawing, except the color and the writemask. Each draw mode has a separate current color and current writemask, which are saved and restored when that draw mode is exited and entered. Drawing into the overlay planes can only be done in colorindex mode. The system automatically goes into colorindex mode when the OVERDRAW mode is entered. Lighting and NURBS subroutines do not work correctly in OVERDRAW mode. Because of the small number of bitplanes, only flat shading is possible; Gouraud shading does not work. The z-buffer is not updated when in OVERDRAW mode. The system must have been configured to contain overlay planes before the OVERDRAW mode can be entered. Overlay planes may be configured by calling the **overlay** subroutine followed by the **gconfig** subroutine. **Note:** To z-buffer drawing into the overlay planes, enable z-buffering. If you do not want to z-buffer this type of drawing, disable z-buffering by calling the **zbuffer** subroutine with a value of False while drawing into the overlay planes. Unclear visual images may be the unintentional result when a drawing in the overlay becomes z-buffered.
PUPDRAW	Only the **mapcolor** subroutine and **getmcolor** subroutine are affected. Drawing subroutines cannot be used to draw into the pop-up menus. In particular, lines, polygons, and pixmaps cannot be drawn into the pop-up menus. Only the pop-up subroutines may be used to access the pop-up menus.
CURSORDRAW	Only the **mapcolor** and **getmcolor** subroutine are affected. Drawing subroutines cannot be used to draw into the cursor. Only the cursor subroutines access the cursor.

Example

The example C language program **prompt.c** uses the **drawmode** subroutine with the PUPDRAW mode identifier to set the drawing mode for operations on the pop-up menus.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Only the Color Graphics Processor has underlay planes. Neither the POWERgraphics Gt4 family of graphics adapters, nor the POWERgraphics GXT1000 posses underlay planes. The call drawmode(UNDERDRAW) is ignored on the GXT1000 and Gt4 family adapters.

The call drawmode(OVERDRAW) is ignored for windows that have been created with the **winX** subroutine, unless the **glcompat** GLC_CREATE_OVERLAY flag has been set.

When performing BLITs to and from the overlay planes on the GXT1000, note that the transparent pixel value on the GXT1000 is 255, not 0. That is, transparent pixels in a pixmap should have the value of 255. Ordinary drawing into the overlay planes still uses color 0 as the transparent color.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling drawing into the back buffer with the **backbuffer** subroutine.

Setting color map mode as the current mode with the **cmode** subroutine.

Setting the color index in the current mode with the **color** subroutine.

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Enabling or disabling drawing into the front buffer with the **frontbuffe**r subroutine.

Changing a color map entry to an RGB value with the **mapcolor** subroutine.

Setting the number of bitplanes used for overlay colors with the **overlay** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Setting a display mode that bypasses the color map with the **RGBmode** subroutine.

Granting write access to a subset of available bitplanes with the **RGBwritemask** subroutine.

Setting the display mode to single buffer mode with the **singlebuffer** subroutine.

Setting the number of bitplanes used for underlay colors with the **underlay** subroutine.

Granting write permission to a subset of available bitplanes with the **writemask** subroutine.

Turning z-buffer mode on and off with the **zbuffer** subroutine.

Enabling drawing to the z-buffer with the **zdraw** subroutine.

Creating Animated Scenes, Removing Hidden Surfaces, and Working in Color Map and RGB Modes.

Configuring the Frame Buffer and Writemasks and Logical Operation.

---

editobj Subroutine

Purpose

Opens a display list for editing.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void editobj(Int32  object)
```

FORTRAN Syntax

```
SUBROUTINE EDITOB(object)
INTEGER*4 object
```

Description

The **editobj** subroutine opens a display list for editing. The system initializes an editing pointer to the end of the newly opened object. It appends all new routines at that pointer location until there is a call to the **closeobj** subroutine or to one that repositions the editing pointer, such as the **objdelete**, **objinsert**, or **objreplace** subroutines.

Usually it is not necessary to be concerned about memory allocation. Objects grow and shrink automatically as routines are added and deleted.

A call for an undefined object identifier causes the system to display an error message.

Note: This editing subroutine itself cannot be added to a display list.

Parameter

object Specifies identifier for object definition to edit.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the amount of memory allocated for an object with the **chunksize** subroutine.

Closing an object with the **closeobj** subroutine.

Compacting the memory storage of an object with the **compactify** subroutine.

Creating an object with the **makeobj** subroutine.

Inserting a routine into an object with the **objinsert** subroutine.

Deleting a routine from an object with the **objdelete** subroutine.

Replacing an existing display list routine with a new one with the **objreplace** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

endfullscrn Subroutine

Purpose

Ends full-screen mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void endfullscrn( )
```

FORTRAN Syntax

```
SUBROUTINE ENDFUL
```

Description

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

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling drawing outside current window boundaries with the **fullscrn** subroutine.

Windows and Input Control Overview.

---

endpick Subroutine

Purpose

Turns off picking mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 endpick(Int16  buffer[ ])
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION ENDPIC(buffer)
INTEGER*2 buffer(1)
```

Description

The **endpick** subroutine turns off picking mode and returns the number of hits.

When the system is in picking mode, and a subroutine draws in the picking region, the contents of the name stack are stored in a buffer, along with the number of names in the stack.

If a drawing primitive overlaps or intrudes upon the picking volume, a hit has occurred. The hit is recorded only if the name stack has been touched since the last hit. Any of the subroutines **loadname**, **pushname**, or **popname** touch the name stack. The first hit after picking begins is always recorded.

A hit is recorded by placing the depth of the name stack into the next vacant slot in the buffer, followed by the entire contents of the name stack. The bottom of the name stack is transferred to the buffer first, followed by the second from the bottom entry of the name stack, and so forth. In other words, from bottom to top is mapped to from left to right.

Note: This subroutine cannot be used to add to a display list.

Parameter

buffer Specifies a buffer in which to write the number of hits.

Return Value

The number of times the name stack was written to the buffer. If the returned function value is negative, then the buffer was too small to contain all the readings from the name stack. The absolute value is the number of stacks actually recorded.

Example

The example C language program **pick1.c** calls the **endpick** subroutine to turn off picking mode.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning off selecting mode with the **endselect** subroutine.

Putting the system in selecting mode with the **gselect** subroutine.

Initializing the name stack with the **initnames** subroutine.

Loading the name on top of the name stack with the **loadname** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Popping a name off the name stack with the **popname** subroutine.

Pushing a new name onto the name stack with the **pushname** subroutine.

Graphics Library Overview, Working with Coordinate Systems, and Picking and Selecting Overview.

---

endselect Subroutine

Purpose

Turns off selecting mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 endselect(Int16  buffer[ ])
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION ENDSEL(buffer)
INTEGER*2 buffer(1)
```

Description

The **endselect** subroutine turns off selecting mode and returns the number of hits.

When the system is in selecting mode, and a subroutine draws in the selecting region, the contents of the name stack are stored in a buffer, along with the number of names in the stack.

If a drawing primitive overlaps or intrudes upon the selecting volume, a hit has occurred. The hit is recorded only if the name stack has been touched since the last hit. Any of the subroutines **loadname**, **pushname**, or **popname** touch the name stack. The first hit after selecting begins is always recorded.

A hit is recorded by placing the depth of the name stack into the next vacant slot in the buffer, followed by the entire contents of the name stack. The bottom of the name stack is transferred to the buffer first, followed by the second from the bottom entry of the name stack, and so forth. In other words, from bottom to top is mapped to from left to right.

Note: This subroutine cannot be used to add to a display list.

Parameter

buffer Specifies a buffer in which to write the number of hits.

Return Value

The number of times the name stack was recorded into the buffer. If the returned function value is negative, then the buffer was too small to contain all the readings from the name stack. The absolute value is the number of stacks actually recorded.

Example

The example C language program **select1.c** uses the **endselect** subroutine to turn off selecting mode.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning off picking mode with the **endpick** subroutine.

Putting the system in selecting mode with the **gselect** subroutine.

Initializing the name stack with the **initnames** subroutine.

Loading the name on top of the name stack with the **loadname** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Popping a name off the name stack with the **popname** subroutine.

Pushing a new name onto the name stack with the **pushname** subroutine.

Graphics Library Overview and Picking and Selecting Overview.

---

finish Subroutine

Purpose

Blocks until all buffers and first-in-first-out (FIFOs) queues are empty.

Library

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void finish
```

FORTRAN Syntax

```
SUBROUTINE FINISH
```

Description

The **finish** subroutine blocks until all buffers and FIFOs that buffer up GL drawing commands are empty. The **finish** subroutine does not return to the user application until all drawings sent to a window are displayed on the screen.

The amount of time that elapses before the **finish** subroutine returns can be highly variable. This timeframe depends on the depth of the FIFO, the nature of its contents (for example, lines or polygons), and the performance of the graphics adapter. Excessive use of the **finish** subroutine may adversely impact overall system performance.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000.

GL implementations are not consistent in the use of buffers and/or FIFOs. The following list illustratrates the variety of ways FIFOs or buffers can be used in a GL implementation:

- A separate FIFO for every window
- One FIFO per process
- A single FIFO for the entire system.

Certain events, such as switching drawing from one window to another or using some **get** functions, can cause buffers and FIFOs to be drained.

The POWER Gt4 and POWER Gt4x adapters implement one logical FIFO per window. The **finish** subroutine forces the draining of only the FIFO associated with the current window.

The following adapters and processors have one graphics command FIFO for the entire system:

- 3-D Color Graphics Processor
- POWERstation 730
- POWERgraphics GTO.

This FIFO is automatically drained whenever a switch to a different window is initiated. That is, drawing in one window, and then drawing in another, implicitly drains the FIFO.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/X11/Xlib.h	Contains C language constant and variable type definitions for version X11 of Enhanced X-Windows.

Related Information

Using Enhanced X-Windows Calls with GL Subroutines in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Enhanced X-Windows and GL Interoperability in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*

---

font Subroutine

Purpose

Selects a raster font for drawing text strings.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void font(Int32  fontnum)
```

FORTRAN Syntax

```
SUBROUTINE FONT(fontnum)
INTEGER*4 fontnum
```

Description

The **font** subroutine selects the raster font that the **charstr** subroutine uses when it draws a text string. This font remains in effect until you call the **font** subroutine again. Font `0` (zero) is the default.

Parameter

fontnum Specifies a font identifier, an index into the font table built by the **defrasterfont** subroutine. If you specify a font number that is not defined, the system selects font `0` (zero).

Example

The example C language programs **curved.c** and **font3.c** use the **font** subroutine to select a raster font defined with the **defrasterfont** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a string of raster characters on the screen with the **charstr** subroutine.

Defining bitmaps for a raster font with the **defrasterfont** subroutine.

Returning the baseline extent of the longest character descender with the **getdescender** subroutine.

Returning the current raster font number with the **getfont** subroutine.

Returning the maximum character height in the current raster font with the **getheight** subroutine.

Returning the width of the specified text string with the **strwidth** subroutine.

Graphics Library Overview and Creating Text Characters.

freepup Subroutine

Purpose
Deallocates a menu.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void freepup(Int32  popup)
```

FORTRAN Syntax
```
SUBROUTINE FREEPU(popup)
INTEGER*4 popup
```

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

Note: This subroutine cannot be used to add to a display list.

Parameter
popup Specifies the pop-up menu to deallocate.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Adding an item to an existing pop-up menu with the **addtopup** subroutine.

Defining a pop-up menu with the **defpup** subroutine.

Displaying a pop-up menu with the **dopup** subroutine.

Allocating and initializing a structure for a new pop-up menu with the **newpup** subroutine.

Enabling or disabling a given pop-up entry with the **setpup** subroutine.

Graphics Library Overview and Creating and Managing Pop-Up Menus.

frontbuffer Subroutine

Purpose
Enables or disables drawing into the front buffer.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void frontbuffer(Int32  bool )
```

FORTRAN Syntax
```
SUBROUTINE FRONTB(bool)
LOGICAL bool
```

Description
The **frontbuffer** subroutine enables drawing into the front frame buffer. In common usage, drawing is done to the back buffer, after which a call to the **swapbuffers** subroutine is made to exchange buffers. The **frontbuffer** subroutine can be used to override this default.

This routine is useful only in double buffer mode and is ignored in single buffer mode.

Parameter
bool Specifies the value for the state of the front frame buffer. The settings for the *bool* parameter are:
True = drawing into the front buffer is enabled.
False = drawing into the front buffer is disabled.

The **gconfig** subroutine sets the front buffer to FALSE.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Enabling drawing into the back buffer with the **backbuffer** subroutine.

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Finding out which buffers are enabled for writing with the **getbuffer** subroutine.

Exchanging the front and back buffers with the **swapbuffers** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Configuring the Frame Buffer, Creating Animated Scenes.

---

frontface Subroutine

Purpose
Controls frontfacing polygon removal.

Library
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void frontface (Int32 mode)
```

FORTRAN Syntax
```
SUBROUTINE FRONTF(mode)
INTEGER*4 mode
```

Description
The **frontface** subroutine allows or suppresses the display of frontfacing filled polygons. With the **backface** subroutine, it is useful for drawing polygons that have different colors on each side. The **frontface** subroutine can also be useful for performing a primitive kind of hidden surface removal.

A frontfacing polygon is defined as a polygon whose vertices are in counterclockwise order in screen coordinates. When frontfacing polygon removal is on, the system displays only polygons whose vertices are in clockwise order.

If both frontfacing and backfacing polygon removal is enabled, no filled polygons are displayed. (An exception is made if the graphics pipeline cannot determine if a polygon is front- or backfacing because it is degenerate or otherwise exceptional.)

Notes:

1. 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 the **frontface** subroutine to do the opposite of what is intended.
2. If a polygon is extremely small or has degenerate vertices, the graphics pipeline cannot determine if the polygon is frontfacing. In such cases, the polygon is rendered by default.

Parameters
mode Sets action of subroutine:
True = Enables frontfacing polygon removal.
False = Disables frontfacing polygon removal.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The 3-D Color Graphics Processor does not support this subroutine. It has no effect and is ignored.

On the POWERgraphics GTO adapter On the POWERgraphics GTO and POWER GXT1000 adapters, either frontfacing polygon removal or backfacing polygon removal can be specified, but not both simultaneously. Enabling frontfacing culling disables backfacing culling.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling or disabling backfacing polygon removal with the **backface** subroutine.

Graphics Library Overview and Removing Hidden Surfaces.

---

fudge Subroutine

Purpose

Specifies pixel values that are added to a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void fudge(Int32  xfudge, Int32  yfudge)
```

FORTRAN Syntax

```
SUBROUTINE FUDGE(xfudge, yfudge)
INTEGER*4 xfudge, yfudge
```

Description

The **fudge** subroutine specifies pixel values that are added to the dimensions of a window when it is sized. Typically, this subroutine is used to create interior window borders. Call the **fudge** subroutine before calling the **winopen** subroutine.

The **fudge** subroutine is useful in conjunction with the **stepunit** and **keepaspect** subroutines. With the **stepunit** subroutine, the window size for integers *m* and *n* is:

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

With the **keepaspect** subroutine the window size is (*width*, *height*), where:

*(width - xfudge) * yaspect = (height - yfudge) * xaspect*

Note: This subroutine cannot be used to add to a display list.

Parameters

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Obtaining the size of the window with the **getsize** subroutine.

Removing the border from a window with the **noborder** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

fullscrn Subroutine

Purpose

Enables drawing outside current window boundaries.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void fullscrn( )
```

FORTRAN Syntax

```
SUBROUTINE FULLSC
```

Description

The **fullscrn** subroutine allows drawing outside of the current window boundaries. It does so by disabling window clipping.

Note: When full-screen drawing is enabled, the user can draw all over other windows and other clients; therefore, this subroutine should be used with caution.

Normally, the **fullscrn** subroutine is used to draw user interface graphics into the overlay planes by calling `drawmode(OVERDRAW)`.

This subroutine calls `viewport(0,XMAXSCREEN,0,YMAXSCREEN)`. To draw outside the window boundaries, but in an area smaller than the full screen, use the **viewport** subroutine and/or the **scrmask** subroutine.

In addition to enabling drawing into the overlays, the fullscreen mode enables drawing into the main frame buffer—but this is strongly discouraged. It is recommended that you use the fullscreen mode only for rendering into the overlays—not for drawing into the principal frame buffer. The reason for this is that when fullscreen mode is exited, no GL REDRAW or X Exposure events are generated.

When fullscreen mode is exited, the overlay planes are not cleared. Therefore, if the overlay planes have been rendered into, they should be cleared as a matter of courtesy before the call to the **endfullscrn** subroutine.

If a window belonging to a process has overlays that have been explicitly disabled with the **overlay** and **gconfig** subroutines, the **fullscrn** subroutine call canot reenable overlays for that window. Therefore, when rendering into overlay planes while in fullscreen mode, the results of the rendering may not be visible if the drawing occurred over a window that has disabled overlays.

In the current implementation, all mouse motion and mouse button press events are redirected to the process that called the **fullscrn** subroutine. The redirection continues until the **endfullscrn** subroutine call is made. This redirection of input overrides the window manager focus policy.

If a second process enters fullscreen mode when another process is already in this mode, pointer motion and mouse button press events will not be grabbed for the second process.

If keyboard events are needed, use the **XGrabKey** and **XGrabKeyboard** subroutines.

Notes:

1. Using the **fullscrn** subroutine in an application makes that application not compliant with the *Inter-Client Communication Conventions Manual* (ICCCM). Compliance to ICCCM, although encouraged, is optional. For more information on ICCCM, refer to the AIXwindows documentation.
2. The **fullscrn** subroutine cannot be used to add to a display list.
3. When the GLC_FB_FULLSCREEN mode is enabled, and **fullscrn** is used, the colors in the main frame buffer will be temporarily become completely black. The colors are restored when **endfullscrn** is called.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Initializing the graphics system without changing the color map with the **gbegin** subroutine.

Ending fullscreen mode with the **endfullscrn** subroutine.

Reconfiguring the system with the **gconfig** subroutine.

Setting the number of user-defined bitplanes used for overlay drawing with the **overlay** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

gammaramp Subroutine

Purpose

Defines a color map ramp for gamma correction.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void gammaramp
(Int16  red[256], Int16  green[256], Int16  blue[256])
```

FORTRAN Syntax

```
SUBROUTINE GAMMAR(red, green, blue)
INTEGER*2 red(256), green(256), blue(256)
```

Description

The **gammaramp** subroutine supplies a level of indirection for all color map and RGB values. It can provide gamma correction, equalize monitors with different color characteristics, or modify the color warmth of the monitor. The default setting has red[*i*]=green[*i*]=blue[*i*]=*i*.

When the system draws an object in RGB mode, it writes the actual red, green, and blue values to the bitplanes. However, the values displayed on the screen are the indirect values: *red*, *green*, *blue* (where *red*, *green*, *blue* are the arrays last specified by the **gammaramp** subroutine).

In color map mode, objects written in color *i* are displayed as *red*, *green*, *blue*.

Notes:

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)
2. This subroutine cannot be used to add to a display list.

Parameters

red Specifies an array of 256 elements, each containing a setting for the red electron gun.

green Specifies an array of 256 elements, each containing a setting for the green electron gun.
blue Specifies an array of 256 elements, each containing a setting for the blue electron gun.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 supports entire-screen gammaramps. Setting the gammaramp on the GXT1000 will affect all windows on the screen, including X11 windows.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting color map mode as the current mode with the **cmode** subroutine.

Setting the current color in color map mode with the **color** subroutine.

Changing a color map entry to an RGB value with the **mapcolor** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Working in Color Map and RGB Modes.

---

gbegin Subroutine

Purpose

Initializes the graphics system without changing the color map.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void gbegin( )
```

FORTRAN Syntax

```
SUBROUTINE GBEGIN
```

Description

The **gbegin** subroutine initializes the graphics environment to its default values for global state attributes and creates a window that covers the screen. The **gbegin** subroutine queues the REDRAW window manager device, but does not change the color map or interfere with other programs that use the current color map.

This subroutine is not recommended for new application development. The **winopen** subroutine, together with related window management subroutines, provides a more flexible interface for graphics display initialization.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling drawing outside the current window boundaries with the **fullscrn** subroutine.

Initializing the graphics system with the **ginit** subroutine.

Resetting all global state attributes to their initial values with the **greset** subroutine.

Creating a new window with the **winopen** subroutine.

Understanding the Hardware Used by GL and Setting Drawing Attributes.

Clearing, Resetting, and Initializing GL and Windows and Input Control Overview.

---

gconfig Subroutine

Purpose

Reconfigures the system.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void gconfig( )
```

FORTRAN Syntax

```
SUBROUTINE GCONFI
```

Description

The **gconfig** subroutine reconfigures the system by setting the requested modes. This subroutine must be called after any call to the **overlay**, **underlay**, **doublebuffer**, **multimap**, **onemap**, **RGBmode**, **cmode**, or **singlebuffer** subroutines.

After a call to the **gconfig** subroutine, the current writemask and color are reset to their default values. The contents of the color map do not change.

Notes:

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″ Understanding the Adapter″.)
2. This subroutine cannot be used to add to a display list.

Example

The example C language program **overlay.c** uses the **qconfig** subroutine to configure the system after calling the **overlay** subroutine and the **doublebuffer** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 supports RGB and color index modes by creating separate separate and distinct X11 windows—one window is permanently in RGB mode, and the other is permanently in color index mode. The **gconfig** subroutine on this adapter merely redirects drawing into one of these two windows. This requires a round-trip to the X11 server causing **gconfig** to run more slowly than on earlier adapters.

The **gconfig** subroutine cannot be used to change the rendering mode (RGB mode versus colorindex mode) of windows created with the **winX** subroutine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting color map mode as the current mode with the **cmode** subroutine.

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Organizing the color map as one large map with the **onemap** subroutine.

Setting the number of bitplanes used for overlay with the **overlay** subroutine.

Setting a display mode that bypasses the color map with the **RGBmode** subroutine.

Setting the display mode to single buffer mode with the **singlebuffer** subroutine.

Setting the number of bitplanes used for underlay with the **underlay** subroutine.

Understanding the Adapter and Working in Color Map and RGB Modes.

Clearing, Resetting, and Initializing GL and Configuring the Frame Buffer.

genobj Subroutine

Purpose
Returns a unique integer for use as an object identifier.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 genobj( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION GENOBJ
```

Description
The **genobj** subroutine generates unique 31-bit integer numbers for use as object identifiers.

When using a combination of user-defined and **genobj**-defined numbers to generate object numbers, ensure that each combination is unique because the **genobj** subroutine will not generate an object name that is currently in use.

The **isobj** subroutine can affirm the uniqueness of an object number.

Note: This editing subroutine itself cannot be added to a display list.

Return Value
A unique 31-bit integer for use as an object identifier.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Drawing an instance of an object with the **callobj** subroutine.

Opening an object for editing with the **editobj** subroutine.

Returning a unique integer for use as a tag with the **gentag** subroutine.

Establishing the uniqueness of an object number with the **isobj** subroutine.

Creating an object with the **makeobj** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

gentag Subroutine

Purpose
Returns a unique integer for use as a tag.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 gentag( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION GENTAG
```

Description
The **gentag** subroutine generates a unique 31-bit integer number for use as a tag. Tags must be unique within an object. Although the **gentag** subroutine generates unique tags, if a tag is defined later with the same value, the first tag is lost.

The **istag** subroutine can affirm the uniqueness of a tag number.

> **Note:** This editing subroutine itself cannot be added to a display list.

Return Value
A unique 31-bit integer for use as a tag.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Deleting tags from a display list with the **deltag** subroutine.

Returning a unique integer for use as an object identifier with the **genobj** subroutine.

Establishing the uniqueness of a tag with the **istag** subroutine.

Marking a location in a display list with the **maketag** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

getbackface Subroutine

Purpose

Returns the state of backfacing filled polygon removal.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getbackface( )
```

FORTRAN Syntax

```
INTEGER*4  GETBAC
```

Description

The **getbackface** subroutine returns the state of backfacing filled polygon removal. If backface removal is on, the system draws only those polygons that face the viewer.

Note: This subroutine cannot be used to add to a display list.

Return Values

0 Removal enabled.
1 Removal disabled.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allowing or suppressing the display of backfacing polygons with the **backface** subroutine.

Graphics Library Overview, Querying the System, and Removing Hidden Surfaces.

getbuffer Subroutine

Purpose
Indicates which buffers are enabled for writing.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 getbuffer( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION GETBUF
```

Description
The **getbuffer** subroutine indicates which buffers are enabled for writing in double buffer mode.

Note: This subroutine cannot be used to add to a display list.

Return Values

Symbolic Name		
C	**FORTRAN**	**Buffer Enabled**
NOBUFFER	NOBUFF	None
BCKBUFFER	BCKBUF	Back buffer (default)
FRNTBUFFER	FRNTBU	Front buffer
BOTHBUFFERS	BOTHBU	Both buffers
DRAWZBUFFER	DRAWZB	Z-buffer drawing

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindowsFeature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Enabling updating in the back buffer with the **backbuffer** subroutine.

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Enabling updating in the front buffer with the **frontbuffer** subroutine.

Exchanging the front and back buffers with the **swapbuffers** subroutine.

Enabling drawing in the zbuffer with the **zdraw** subroutine.

Understanding the Hardware Used by GL, Configuring the Frame Buffer, and Querying the System in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Creating Animated Scenes.

---

getbutton Subroutine

Purpose

Returns the state (up or down) of a button.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getbutton(Device  number)
```

FORTRAN Syntax

```
LOGICAL FUNCTION GETBUT(number)
INTEGER*4 number
```

> **Note:** For FORTRAN users, this function accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **getbutton** subroutine returns the state of the button specified in the *number* parameter. A complete list of buttons can be found in the file **/usr/include/gl/device.h**.

> **Note:** This subroutine cannot be used to add to a display list.

Parameter

number Specifies the number of the button to test.

Return Values

The return values and their corresponding states are as follows:

Value	State
0	Up
1	Down
-1	Invalid device number

Example

The example C language program **scrn_rotate.c** uses the **getbutton** subroutine to check the state of various buttons.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current state of a valuator with the **getvaluator** subroutine.

Graphics Library Overview, Controlling Queues and Devices, Using the Keyboard, and Querying the System.

---

getcmmode Subroutine

Purpose

Returns the organization of the color map.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getcmmode( )
```

FORTRAN Syntax

```
LOGICAL FUNCTION GETCMM
```

Description

The **getcmmode** subroutine returns the organization of the current color map.

Note: This subroutine cannot be used to add to a display list.

Return Values

C	FORTRAN	Description
TRUE	CMAPMU	multimap mode
FALSE	CMAPON	onemap mode

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current display mode with the **getdisplaymode** subroutine

Returning the number of the current color map with the **getmap** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Organizing the color map as one large map with the **onemap** subroutine.

Querying the System.

Working in Color Map and RGB Modes.

---

getcolor Subroutine

Purpose

Returns the current color.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getcolor( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETCOL
```

Description

The **getcolor** subroutine returns the current index into the color map for the current drawing mode. In NORMALDRAW, an index from 0 to 4095 is returned. In OVERDRAW and UNDERDRAW, an index from 0 to 15 is returned. The under/over planes have a color map that is separate from the main color map.

Note: This subroutine cannot be used to add to a display list.

Return Value

An index into the color map for the current color.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the color index in the current mode with the **color** subroutine.

Specifying the target framebuffer of the drawing subroutines with the **drawmode** subroutine.

Returning a color map entry with the **getmcolor** subroutine.

Returning the current RGB value with the **gRGBcolor** subroutine.

Graphics Library Overview, Creating Animated Scenes, Querying the System, Setting Drawing Attributes, Understanding the Hardware Used by GL, Drawing Rectangles, Circles, Arcs, and Polygons, and Working in Color Map and RGB Modes.

---

getcpos Subroutine

Purpose

Returns the current character position.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getcpos(Screencoord * ix, Screencoord * iy)
```

FORTRAN Syntax

```
SUBROUTINE GETCPO(ix, iy)
INTEGER*2 ix, iy
```

Description

The **getcpos** subroutine gets the current character position, in screen coordinates relative to the lower left corner of the window, and writes it into the parameters.

Note: This subroutine cannot be used to add to a display list.

Parameters

ix Specifies a pointer to the location in which to write the *x* coordinate of the current character position.
iy Specifies a pointer to the location in which to write the *y* coordinate of the current character position.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a string of raster characters on the screen with the **charstr** subroutine.

Moving the current character position with the **cmov** subroutine.

Returning the current graphics position with the **getgpos** subroutine.

Graphics Library Overview, Creating Text Characters, and Querying the System.

---

getcursor Subroutine

Purpose

Returns the cursor characteristics.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getcursor
(Int16 * index,
Colorindex * color, Colorindex * writemask,
Int32 * bool)
```

FORTRAN Syntax

```
SUBROUTINE GETCUR(index, color, writemask, bool)
INTEGER*2 index, color, writemask
LOGICAL bool
```

Description

The **getcursor** subroutine finds the index of the current cursor and returns it in the *index* parameter. The cursor index is an index into a table of cursor bitmaps set by the **defcursor** subroutine.

The default is the cursor at index 0 (zero) in the cursor bitmaps. This cursor is displayed in red and is automatically updated on each vertical retrace.

Note: This subroutine cannot be used to add to a display list.

Parameters

index	Specifies an index that was previously associated with a bitmap by the **defcursor** subroutine.
color	Retained for compatibility, but disregarded.
writemask	Retained for compatibility, but disregarded.
bool	Specifies a pointer to the location into which the system returns a boolean indicating if the cursor is visible in the current window.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a cursor with the **defcursor** subroutine.

Setting the cursor characteristics with the **setcursor** subroutine.

Graphics Library Overview, Creating a Cursor, and Creating and Managing Windows.

---

getdcm Subroutine

Purpose

Indicates whether depth-cue mode is on or off.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getdcm( )
```

FORTRAN Syntax

```
LOGICAL FUNCTION GETDCM
```

Description

The **getdcm** subroutine returns True if the system is in depth-cue mode and False if it is not.

Note: This subroutine cannot be used to add to a display list.

Return Values

False	System not in depth-cue mode.

True System in depth-cue mode.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning depth-cueing on and off with the **depthcue** subroutine.

Graphics Library Overview, Performing Depth-Cueing, and Working in Color Map and RGB Modes.

---

getdescender Subroutine

Purpose

Returns the baseline extent of the longest character descender.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getdescender( );
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETDES
```

Description

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

Note: This subroutine cannot be used to add to a display list.

Return Value

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current raster font number with the **getfont** subroutine.

Returning the maximum character height in the current raster font with the **getheight** subroutine.

Returning the width of the specified text string with the **strwidth** subroutine.

Graphics Library Overview, Creating Text Characters, and Querying the System.

---

getdev Subroutine

Purpose

Reads a list of valuators.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getdev
(Int32  number,
Device * devices,
Int16 * values)
```

FORTRAN Syntax

```
SUBROUTINE GETDEV(number, devices, values)
INTEGER*4 number
INTEGER*2 devices(number), values(number)
```

Description

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

Note: This subroutine cannot be used to add to a display list.

Parameters

number	Specifies the number of devices pointed to by the *devices* parameter (no more than 128).
devices	Specifies an array containing the identifiers (device number constants, such as MOUSEX, BPADX, and LEFTMOUSE) of the devices to read. The array pointed to by the *devices* parameter can contain up to 128 devices.
values	Specifies the array into which the system is to write the values read from the devices listed in the *devices* array. Each member in the *values* array corresponds to a member of the *devices* array and returns the state of each device in the corresponding location.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current state of a valuator with the **getvaluator** subroutine.

Graphics Library Overview, Controlling Queues and Devices, Using the Keyboard, and Querying the System.

---

getdisplaymode Subroutine

Purpose

Returns the current display mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getdisplaymode( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETDIS
```

Description

The **getdisplaymode** subroutine returns the current configuration of the frame buffer bitplanes and color map.

Note: This subroutine cannot be used to add to a display list.

Return Values

Symbolic Name		
C	**FORTRAN**	**Display Mode**
DMRGB	DMRGB	RGB single buffer mode
DMSINGLE	DMSING	Color map single buffer mode
DMDOUBLE	DMDOUB	Color map double buffer mode
DMRGBDOUBLE	DMRGBD	RGB double buffermode

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting color mode as the current mode with the **cmode** subroutine.

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Returning the organization of the current color map with the **getcmmode** subroutine.

Returning the current drawing mode with the **getdrawmode** subroutine.

Setting a display mode that bypasses the color map with the **RGBmode** subroutine.

Setting the display to single buffer mode with the **singlebuffer** subroutine.

Understanding the Hardware Used by GL, Creating Animated Scenes, Querying the System.

Working in Color Map and RGB Modes and Configuring the Frame Buffer for GL.

---

getdrawmode Subroutine

Purpose

Returns the current drawing mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getdrawmode( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETDRA
```

Description

The **getdrawmode** subroutine returns the current drawing mode specified by the **drawmode** subroutine.

Note: This subroutine cannot be used to add to a display list.

Return Values

Symbolic Name		
C	**FORTRAN**	**Draw mode**
UNDERDRAW	UNDRDR	Underlay (background) bitplanes
NORMALDRAW	NORMDR	Main frame buffer bitplanes
OVERDRAW	OVRDRW	Overlay (foreground) bitplanes
PUPDRAW	PUPDRW	Pop-up menus
CURSORDRAW	CURSDR	Cursor mode

Example

The example C language program **prompt.c** uses the **getdrawmode** subroutine to get the current drawing mode so that it can be restored later.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Choosing a set of bitplanes for drawing with the **drawmode** subroutine.

Returning the current display mode with the **getdisplaymode** subroutine.

Popping the viewport stack with the **popviewport** subroutine.

Creating a Cursor, Removing Hidden Surfaces, and Working in Color Map and RGB Modes.

Configuring the Frame Buffer, Writemasks and Logical Operation, and Querying the System.

---

getfont Subroutine

Purpose

Returns the index of the current raster font.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 getfont( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION GETFON
```

Description
The **getfont** subroutine returns the index of the current raster font. The returned value is an index into the raster font table.

Note: This subroutine cannot be used to add to a display list.

Return Value
The index of the current raster font.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Writing a text string with the **charstr** subroutine.

Defining bitmaps for a raster font with the **defrasterfont** subroutine.

Selecting a raster font with the **font** subroutine.

Graphics Library Overview, Creating Text Characters, and Querying the System.

---

getfontencoding Subroutine

Purpose
Indicates the font encoding.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getfontencoding (char * enc)
```

FORTRAN Syntax

```
SUBROUTINE GETFONTENCODING (enc)
CHARACTER *(*)  enc
```

Description

The **getfontencoding** subroutine returns an ASCII string that indicates the registry and encoding of the currently bound font.

The returned string is always NULL for user-defined fonts (defined with the **defrasterfont** subroutine). The returned string may or may not be NULL for fonts loaded with the **loadXfont** subroutine. Typically, newer fonts indicate their encoding, older fonts do not.

The encoding is determined with the Enhanced X-Windows Logical Font Description (XLFD) mechanism. The actual string returned is a concatenation of the contents of the CharSetRegistry and CharSetEncoding fields of logical font description.

Examples of font set registries and encodings are ISO8859-5 or JISX.1983-0. In the first example, ISO8859 is the registry (in this case, the International Standard ISO 8859, Information Processing - 8-bit Single Byte Coded Graphics Character Sets), and 5 is the encoding (in this case, ISO 8859/5, a Russian/Cyrillic encoding).

More information about font encodings can be found in ″National Language Support Overview″.

Parameters

enc Indicates a pointer to a memory location. The user passes a pointer to a memory area that point to *at least* 24 bytes that can be overwritten. Upon return, their memory area contains a ASCII string identifying the registry and encoding of the currently bound font.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Determining the font format with the **getfonttype** subroutine.

Defining bitmaps for a raster font with the **defrasterfont** subroutine.

Selecting a raster font with the **font** subroutine.

Loading an Enhanced X-Windows font into the font table with the **loadXfont** subroutine.

National Language Support Overview in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Graphics Library Overview in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

getfonttype Subroutine

Purpose

Indicates whether the current font is a double-byte character set (DBCS) font.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
int getfonttype()
```

FORTRAN Syntax

```
INTEGER *4 FUNCTION GETFONT
```

Description

The **getfonttype** subroutine indicates whether the currently bound font is a single-byte or double-byte font.

The behavior of the **charstr** subroutine depends on the font type of the currently loaded font. Use the **getfonttype** subroutine to determine whether to pass single-byte or double-byte character strings to the **charstr** subroutine.

> **Note:** Most European fonts, including all ISO8859 code sets, are single-byte fonts. Most Asian font sets, including the JIS (Japanese Kanji) fonts, are double-byte fonts.

Return Values

FT_SBCS Indicates that the current font is a single-byte font.
FT_DBCS Indicates that the current font is a double-byte font.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Determining the font name with the **getfontencoding** subroutine.

Defining bitmaps for a raster font with the **defrasterfont** subroutine.

Selecting a raster font with the **font** subroutine.

National Language Support Overview for Programming in *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Creating Text Characters.

---

getgdesc Subroutine

Purpose

Returns information about the currently installed graphics hardware .

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getgdesc(Int32  inquiry);
```

FORTRAN Syntax

```
INTEGER *4 FUNCTION GETGDE(inquiry)
INTEGER*4 inquiry
```

Description

The **getgdesc** subroutine allows you to inquire about characteristics of the graphics system. The characteristics are all hardware characteristics; that is, the values returned do not change as the adapter is software reconfigured.

It is not necessary to open a GL window or call the **ginit** subroutine before using the **getgdesc** subroutine.

Note: This subroutine cannot be used to add to a display list.

Parameter

inquiry Represents the characteristics about which you want to inquire. Inquiries are submitted using tokens.

Return Values

The value of the requested characteristic, or -1 if the request is not valid or cannot be determined. For a listing of return values for adapters, see the ″Adapter Description Table for GL″, Appendix B.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Getting the library version number with the **gversion** subroutine.

Understanding the Hardware Used by GL and Clearing, Setting, and Initializing GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Tokens for the getgdesc Subroutine

The following data is returned when the indicated token is passed to the subroutine:

C	FORTRAN	Description
GD_BITS_ALPHABUFFER	GDBITA	The number of alpha blending bitplanes available on adapter. A value of zero is returned if there are no alpha bitplanes.
GD_BITS_CURSOR	GDBITC	The number of cursor bitplanes available on adapter.
GD_BITS_LINESTYLE	GDBITL	The number of independent bits in a linestyle.
GD_BITS_NORM_DBL_CMODE	GDDBLC	The number of bitplanes in each of the pair of main display buffers when the main display buffer is configured for double-buffered color map mode.
GD_BITS_NORM_DBL_RGB	GDDBLR	The number of bitplanes in each of the pair of main display buffers when the main display buffer is configured for double-buffered RGB mode.
GD_BITS_NORM_SNG_CMODE	GDSNGC	The number of bitplanes in the main display buffer when it is configured for single-buffered color map mode.
GD_BITS_NORM_SNG_RGB	GDSNGR	The number of bitplanes in the main display buffer when it is configured for single-buffered RGB mode.
GD_BITS_OVERLAY	GDBITO	The maximum number of overlay bitplanes for which the overlay buffer can be configured. Note that some adapters possess auxiliary bitplanes that can be configured as overlay or underlay bitplanes, but not both.
GD_BITS_PLANE_MASK	GDBITP	The maximum number of plane mask bitplanes available on the adapter.
GD_BITS_UNDERLAY	GDBITU	The maximum number of underlay bitplanes for which the underlay buffer can be configured. Note that some adapters possess auxiliary bitplanes that can be configured as overlay or underlay bitplanes, but not both.
GD_BITS_ZBUFFER	GDBITZ	The number of bitplanes in the z-buffer. A value of zero is returned if a z-buffer is not installed.

GD_BLEND	GDBLEN	True if alphablending is supported, False if not. **Note:** Alphablending does not necessarily require alpha bitplanes.
GD_BUTBOX	GDBUTB	True if a button box is attached to the system, False if not.
GD_DIALS	GDDIAL	True if dials are attached to the system, False if not.
GD_DITHER	GDDITH	A bit flag is returned indicating which buffers support dithering. The bit flag is the bitwise OR of the following flags: **GD_BF_UNDER** - Dithering in the underlay planes is supported. **GD_BF_OVER** - Dithering in the overlay planes is supported. **GD_BF_MAIN** - Dithering in the main frame buffer is supported. **GD_BF_ALPHA** - Dithering in the alpha buffer is supported.
GD_LARGE_MAP_SIZE	GDLMAP	The size of the single ″large″ color map.
GD_LIGHTING_TWOSIDE	GDLIG2	True if two-sided lighting is supported, False if not.
GD_LINESMOOTH_CMODE	GDLINC	True if smooth lines (antialiased lines) are supported in color index mode.
GD_LINESMOOTH_RGB	GDLINR	True if smooth lines (antialiased lines) are supported in RGB mode.
GD_MAX_ATTR_STACKDEPTH	GDMAST	The maximum depth of the attribute stack.
GD_MAX_LSREPEAT	GDMLSR	The maximum linestyle repeat factor that can be set.
GD_MAX_MATRIX_STACKDEPTH	GDMMST	The maximum depth of the modeling/viewing matrix stack.
GD_MAX_NURBS_ORDER	GDMNOR	The maximum supported order of a NURBS surface.
GD_MAX_PATTERN_SIZE	GDMPSZ	The maximum dimension of a pattern bitmask.
GD_MAX_TRIM_ORDER	GDMTOR	The maximum supported order of a NURBS trimming curve.
GD_MAX_VERTS	GDMVER	The maximum number of vertices that can be specified between a **bgn...()** and **end...()** subroutine pair.
GD_MAX_VIEWPORT_STACKDEPTH	GDMVST	The maximum depth of the viewport stack.
GD_NUM_SMALL_MAPS	GDNSM	The number of small color maps.
GD_OVERUNDER_SHARED	GDOUSH	True if overlay and underlay bitplanes are shared, False if not.
GD_POINTSMOOTH_CMODE	GDPSCM	True if smooth points (antialiased, pixel-sized points) are supported in color index mode.
GD_POINTSMOOTH_RGB	GDPSRG	True if smooth points (antialiased, pixel-sized points) are supported in RGB mode.
GD_POLYMODE	GDPOLY	True if polymode is functional, FALSE if not.

GD_RGBMODE	GDRGBM	A bit flag is returned indicating which buffers support an RGB mode (color map mode is always supported in all bitplanes). The bit flag is the bitwise *or* of the following flags: **GD_BF_UNDER** - RGB mode in the underlay planes is supported. **GD_BF_OVER** - RGB mode in the overlay planes is supported. **GD_BF_MAIN** - RGB mode the main frame buffer is supported.
GD_SHADEBUFFERS	GDSHAB	A bit flag is returned indicating which buffers support smooth (Gouraud) shading. The bit flag is the bitwise *or* of the following flags: **GD_BF_UNDER** - Smooth shading in the underlay planes is supported. **GD_BF_OVER** - Smooth shading in the overlay planes is supported. **GD_BF_MAIN** - Smooth shading in the main frame buffer is supported. **GD_BF_ALPHA** - Smooth shading in the alpha buffer is supported.
GD_SHADEMODEL	GDSHAM	A bit flag is returned indicating which shading models are supported. The bit flag is the bitwise OR of the following flags: **GD_SHADEMODEL_FLAT** - Only flat shading is supported. **GD_SHADEMODEL_GOURAUD** - Gouraud shading (linear interpolation of colors) is supported. **GD_SHADEMODEL_PSEUDO_PHONG** - Pseudo Phong shading (adaptive tessellation of lit polygons) is supported. **GD_SHADEMODEL_PHONG** - Phong shading (hardware interpolation of normal vectors) is supported. **GD_SHADEMODEL_QUADRATIC** - Quadratic interpolation of colors and textures maps is supported.
GD_SMALL_MAP_SIZE	GDSMAP	The size of the multiple ″small″ color maps.
GD_SUBPIXEL_LINE	GDSPLI	A bit flag is returned indicating which buffers support subpixel-positioned lines. The bit flag is the bitwise *or* of the following flags: **GD_BF_UNDER** - Subpixel-positioned lines in the underlay planes are supported. **GD_BF_OVER** - Subpixel-positioned lines in the overlay planes are supported. **GD_BF_MAIN** - Subpixel-positioned lines in the main frame buffer are supported. **GD_BF_ALPHA** - Subpixel-positioned lines in the alpha buffer are supported.

GD_SUBPIXEL_PNT	GDSPPN	A bit flag is returned indicating which buffers support subpixel-positioned points. The bit flag is the bitwise *or* of the following flags: **GD_BF_UNDER** - Subpixel-positioned points in the underlay planes are supported. **GD_BF_OVER** - Subpixel-positioned points in the overlay planes are supported. **GD_BF_MAIN** - Subpixel-positioned points in the main frame buffer are supported. **GD_BF_ALPHA** - Subpixel-positioned points in the alpha buffer are supported.
GD_SUBPIXEL_POLY	GDSPPL	A bit flag is returned indicating which buffers support subpixel-positioned polygons. The bit flag is the bitwise *or* of the following flags: **GD_BF_UNDER** - Subpixel-positioned polygons in the underlay planes are supported. **GD_BF_OVER** - Subpixel-positioned polygons in the overlay planes are supported. **GD_BF_MAIN** - Subpixel-positioned polygons in the main frame buffer are supported. **GD_BF_ALPHA** - Subpixel-positioned polygons in the alpha buffer are supported.
GD_TEXTPORT	GDTXTP	True if the textport routines are functional, False if not.
GD_VERT_RETRACE_FREQ	GDVREF	The vertical retrace frequency.
GD_WSYS	GDWSYS	The type of windowing system running on the processor. Returns one of the following:**GD_WSYS_NONE** - No window system is running.**GD_WSYS_4S** - 4Sight Window System (TM) is currently running.**GD_WSYS_AIX113** - AIXwindows, derived from X Windows System release 11.3, is currently running.**GD_WSYS_AIX114** - AIXwindows, derived from Enhanced X-Windows System release 11.4, is currently running.
GD_XMMAX	GDXMAX	Horizontal size of the screen in millimeters.
GD_YMMAX	GDYMAX	Vertical size of the screen in millimeters.
GD_XPMAX	GDXPMA	Horizontal size of the screen in pixels.
GD_YPMAX	GDYPMA	Vertical size of the screen in pixels.
GD_ZDRAW_GEOM	GDZGEO	True if geometrical figures can be drawn into the z-buffer, False if not.
GD_ZDRAW_PIXELS	GDZPIX	True if pixel block transfers to and from the z-buffer are possible, False if not.
GD_ZMIN	GDZMIN	The smallest value that can be written into the z-buffer.
GD_ZMAX	GDZMAX	The largest value that can be written into the z-buffer.

Related Information

The **getgdesc** subroutine.

getgpos Subroutine

Purpose
Returns the current graphics position.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

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

FORTRAN Syntax
```
SUBROUTINE GETGPO(fx, fy, fz, fw)
REAL fx, fy, fz, fw
```

Description
The **getgpos** subroutine returns the current graphics position after transformation by the current matrix.

Note: This subroutine cannot be used to add to a display list.

Parameters
fx Specifies the pointer to the location into which to write the *x* coordinate of the current graphics position.
fy Specifies the pointer to the location into which to write the *y* coordinate of the current graphics position.
fz Specifies the pointer to the location into which to write the *z* coordinate of the current graphics position.
fw Specifies the pointer to the location into which to write the *w* coordinate of the current graphics position. The *w* value is used when defining a 3-D point in homogeneous coordinates.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information
Getting the current text character position with the **getcpos** subroutine.

Graphics Library Overview, Drawing with Move-Draw Style Subroutines, Querying the System, and Working in Color Map and RGB Modes.

getheight Subroutine

Purpose
Returns the maximum character height in the current raster font.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 getheight( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION GETHEI
```

Description
The **getheight** subroutine 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).

Note: This subroutine cannot be used to add to a display list.

Return Value
The maximum height (in pixels) of a character in the current font.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Returning the baseline extent of the longest character descender with the **getdescender** subroutine.

Returning the current raster font number with the **getfont** subroutine.

Returning the width of the specified text string with the **strwidth** subroutine.

Graphics Library Overview, Creating Text Characters, and Querying the System.

getlsrepeat Subroutine

Purpose
Returns the linestyle repeat count.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 getlsrepeat( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION GETLSR
```

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

Note: This subroutine cannot be used to add to a display list.

Return Value
The repeat factor for the current linestyle.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Setting the repeat factor for the current linestyle with the **lsrepeat** subroutine.

Drawing NURBS Curves and Surfaces, Drawing Wire Frame Curves and Surface Patches, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Querying the System, Setting Drawing Attributes, Understanding the Hardware Used by GL, Drawing Rectangles, Circles, Arcs, and Polygons.

getlstyle Subroutine

Purpose
Returns the current linestyle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getlstyle( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETLST
```

Description

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

Note: This subroutine cannot be used to add to a display list.

Return Value

An index into the linestyle table for the current linestyle.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a linestyle with the **deflinestyle** subroutine.

Selecting a linestyle with the **setlinestyle** subroutine.

Drawing NURBS Curves and Surfaces, Drawing Wire Frame Curves and Surface Patches, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Querying the System, Setting Drawing Attributes, Understanding the Hardware Used by GL, Drawing Rectangles, Circles, Arcs, and Polygons.

---

getlwidth Subroutine

Purpose

Returns the current linewidth.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getlwidth( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETLWI
```

Description

The **getlwidth** subroutine returns the current linewidth in pixels.

Note: This subroutine cannot be used to add to a display list.

Return Value

The current linewidth in pixels.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying a linewidth with the **linewidth** subroutine.

Drawing Wire Frame Curves and Surface Patches, Drawing NURBS Curves and Surfaces, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Working in Color Map and RGB Modes, Querying the System, Setting Drawing Attributes, Understanding the Hardware Used by GL, Drawing Rectangles, Circles, Arcs, and Polygons.

---

getmap Subroutine

Purpose

Returns the number of the current color map.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getmap( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETMAP
```

Description

The **getmap** subroutine returns the identification number of the current color map. In multimap mode, there are 16 small, independent color maps; therefore, the **getmap** subroutine returns a value in the range 0 to 15. In onemap mode, the **getmap** subroutine returns 0 (zero).

Note: This subroutine cannot be used to add to a display list.

Return Value

The number of the current color map, a value from 0 to 15.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the organization of the current color map with the **getcmmode** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Organizing the color map as one large map with the **onemap** subroutine.

Selecting one of 16 small color maps with the **setmap** subroutine.

Querying the System.

Working in Color Map and RGB Modes.

---

getmatrix Subroutine

Purpose

Returns the current transformation matrix.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getmatrix(Matrix  matrix)
```

FORTRAN Syntax

```
SUBROUTINE GETMAT(matrix)
REAL matrix(4,4)
```

Description

The **getmatrix** subroutine copies the matrix from the top of the stack to a user-specified array. This routine does not alter the matrix stack. When the system is in projection matrix mode, the matrix stack is not accessible. In projection mode, the *matrix* array receives a copy of the projection matrix instead.

Note: This subroutine cannot be used to add to a display list.

Parameter

matrix Specifies an array into which to copy a matrix.

Example

The example C language program **scrn_rotate.c** uses the **getmatrix** subroutine to get the current transformation matrix after manipulating it.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Loading a transformation matrix with the **loadmatrix** subroutine.

Setting the current matrix mode with the **mmode** subroutine.

Premultiplying the current transformation matrix with the **multmatrix** subroutine.

Popping the transformation matrix stack with the **popmatrix** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Graphics Library Overview, Querying the System, and Working with Coordinate Systems.

---

getmcolor Subroutine

Purpose

Gets a copy of the RGB values for a color map entry.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getmcolor
(Colorindex  index,
Int16 * red, Int16 * green, Int16 * blue)
```

FORTRAN Syntax

```
SUBROUTINE GETMCO(index, red, green, blue)
INTEGER*4 index
INTEGER*2 red, green, blue
```

Description

The **getmcolor** subroutine gets the red, green, and blue components of a color map entry and copies them to the specified locations. This subroutine returns only the values associated with a slot in the current color table. It does not return, nor does it set, the current drawing color. For the current drawing color, use the **getcolor** subroutine in color map mode, and the **gRGBcolor** subroutine in RGB mode.

Note: This subroutine cannot be used to add to a display list.

Parameters

index	Specifies the index into the color map.
red	Specifies a pointer to the location into which to copy the red value of the color specified by *index*.
green	Specifies a pointer to the location into which to copy the green value of the color specified by *index*.
blue	Specifies a pointer to the location into which to copy the blue value of the color specified by *index*.

Example

The example C language program **overlay.c** uses the **getmcolor** subroutine to save the values in the color map before changing them with the **mapcolor** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Choosing a set of bitplanes for drawing with the **drawmode** subroutine.

Returning the current color with the **getcolor** subroutine.

Returning the number of the current color map with the **getmap** subroutine.

Returning the current RGB color with the **gRGBcolor** subroutine.

Changing a color map entry to an RGB value with the **mapcolor** subroutine.

Querying the System.

Working in Color Map and RGB Modes.

---

getmcolors Subroutine

Purpose

Returns a range of color map RGB values.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void getmcolors(Int16 *start_idx*, **Int16** *end_idx*, **Int16 *** *r*, **Int16 *** *g*, **Int16 *** *b*)

FORTRAN Syntax

```
SUBROUTINE GETMCOLORS (start_idx, end_idx, r, g, b)
INTEGER*4 start_idx, end_idx,
INTEGER*2 r(1), g(1), b(1)
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when started from a FORTRAN program, although it accepts short integers when started from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **getmcolors** subroutine returns a range of color map table entries. The range that is returned begins with the *start_idx* parameter and ends with the *end_idx* parameter, inclusive. The length of the array must be equal to *end_idx* - *start_idx* + 1. For instance, to return only one color map entry, set *start_idx* and *end_idx* to the same number. To return two entries, set *end_idx* to *start_idx* + 1.

The **getmcolors** subroutine is functionally equivalent to the C code shown in the following fragment, although it executes considerably faster.

```
{
  int i;
  for ( i=0; i < (end_idx - start_idx + 1); i++ )
     getmcolors (start_idx+i,&r[i],&g[i],&b[i]);
}
```

The **getmcolors** subroutine can be used to read the underlay, overlay, cursor, pop-up, or main frame buffer color map. Which map is read depends on the current drawing mode (as set by the **drawmode** subroutine).

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

start_idx	Specifies the starting index in the color map to be returned.
end_idx	Specifies the ending index in the color map to be returned.
r	Specifies an array containing the intensity of the red component.

g Specifies an array containing the intensity of the green component.
b Specifies an array containing the intensity of the blue component.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Choosing a set of bitplanes for drawing with the **drawmode** subroutine.

Returning the current RGB color with the **gRGBcolor** subroutine.

Loading a range of color map entries with the **mapcolors** subroutine

Querying the System.

Working in Color Map and RGB Modes.

---

getmmode Subroutine

Purpose

Returns the current matrix mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getmmode( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETMMO
```

Description

The **getmmode** subroutine returns the current matrix mode.

Note: This subroutine cannot be used to add to a display list.

Return Values

There are three possible return values for this function:

C	FORTRAN	Mode
MSINGLE	MSINGL	Single matrix
MPROJECTION	MPROJE	Projection matrix
MVIEWING	MVIEWI	Viewing matrix

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Making a material, light, or lighting model definition active with the **lmbind** subroutine.

Defining a new material, light, or lighting model with the **lmdef** subroutine.

Setting the current matrix mode with the **mmode** subroutine.

Graphics Library Overview, Creating Lighting Effects, Querying the System, and Setting Pipeline Options.

---

getnurbsproperty Subroutine

Purpose

Returns the current value of a trimmed NURBS surfaces display property

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getnurbsproperty(Int32  property, Float32 * value)
```

FORTRAN Syntax

```
SUBROUTINE GETNUR(property, value)
INTEGER*4 property
REAL value
```

Description

The **getnurbsproperty** subroutine returns the current value of a trimmed Non-Uniform Rational B-Spline (NURBS) surfaces display property. The display of NURBS surfaces can be controlled in different ways, as explained in the description of the *value* parameter.

Note: This subroutine cannot be used to add to a display list.

Parameters

property
Specifies the name of the property to be queried.

value Specifies a pointer to the location into which the system is to write the value of the named property. The display properties that can be affected and their possible values are as follows:

C	FORTRAN	Description
N_ERRORCHECKING	NERROR	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	NPIXEL	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.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Marking the beginning and end of a NURBS surface definition with the **bgnsurface** and **endsurface** subroutines.

Marking the beginning and end of a NURBS surface trimming loop with the **bgntrim** and **endtrim** subroutines.

Controlling the shape of a NURBS trimming curve with the **nurbscurve** subroutine.

Controlling the shape of a NURBS surface with the **nurbssurface** subroutine.

Describing a piecewise linear trimming curve for NURBS surfaces with the **pwlcurve** subroutine.

Setting a property for the display of trimmed NURBS with the **setnurbsproperty** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Querying the System.

getopenobj Subroutine

Purpose
Returns the current open object.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 getopenobj( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION GETOPE
```

Description
The **getopenobj** subroutine returns the identifier of the object that is currently open for editing. If no object is open, the subroutine returns -1.

Note: This editing subroutine itself cannot be added to a display list.

Return Value
The number of the object currently open for editing.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Closing an object with the **closeobj** subroutine.

Opening an object for editing with the **editobj** subroutine.

Creating an object with the **makeobj** subroutine.

Graphics Library Overview, Creating Objects (Display Lists), and Querying the System.

getorigin Subroutine

Purpose
Returns the position of a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getorigin(Int32 * x, Int32 * y)
```

FORTRAN Syntax

```
SUBROUTINE GETORI(x, y)
INTEGER*4 x, y
```

Description

The **getorigin** subroutine returns the position (in pixels) of the lower-left corner of the current window. A window must be open for this subroutine to work.

Call the **winopen** subroutine to open a window, or the **winset** subroutine to choose the current window.

Note: This subroutine cannot be used to add to a display list.

Parameters

x Specifies a pointer to the location in which to return the *x* position (in pixels) of the lower left corner of the window.

y Specifies a pointer to the location in which to return the *y* position (in pixels) of the lower left corner of the window.

Example

The example C language program **paint.c** uses the **getorigin** subroutine to determine the origin of the window.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Obtaining the size of the window with the **getsize** subroutine.

Constraining the window position and size with the **prefposition** subroutine.

Moving the current window by its lower left corner with the **winmove** subroutine.

Creating a window with the **winopen** subroutine.

Changing the current location and size of a window with the **winposition** subroutine.

Setting the current window with the **winset** subroutine.

Creating and Managing Windows.

---

getpattern Subroutine

Purpose

Returns the index of the current fill pattern.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
long getpattern( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETPAT
```

Description

The **getpattern** subroutine returns the index of the current fill pattern. The returned value is an index into the pattern table.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a pattern with the **defpattern** subroutine.

Selecting a fill pattern for polygons, rectangles, and curves with the **setpattern** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, Drawing Wire Frame Curves and Surface Patches, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Drawing Rectangles, Circles, Arcs, and Polygons.

getplanes Subroutine

Purpose
Returns the number of available bitplanes.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 getplanes( )
```

FORTRAN Syntax
```
INTEGER*4 GETPLA
```

Description
The **getplanes** subroutine returns the number of active bitplanes that are currently being used for drawing. The number returned depends on how the frame buffer has been configured. In particular, the returned value depends on the most recent setting of the **drawmode** subroutine, whether the system is in single or double buffer mode, whether it is in color map or RGB mode, and finally, on the capabilities of the installed adapter.

Note: This subroutine cannot be used to add to a display list.

Return Value
The number of bitplanes available under the current drawmode.

Example
The example C language program **scrn_rotate.c** uses the **getplanes** subroutine to get the number of bitplanes available.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Setting color map mode as the current mode with the **cmode** subroutine.

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Choosing a set of bitplanes for drawing with the **drawmode** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Organizing the color map as one large map with the **onemap** subroutine.

Setting a display mode that bypasses the color map with the **RGBmode** subroutine.

Setting the display mode to single buffer mode with the **singlebuffer** subroutine.

Configuring the Frame Buffer, Writemasks and Logical Operations, and Querying the System.

---

getscrmask Subroutine

Purpose

Returns the current screenmask.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getscrmask
(Screencoord * left, Screencoord * right,
Screencoord * bottom, Screencoord * top)
```

FORTRAN Syntax

```
SUBROUTINE GETSCR(left, right, bottom, top)
INTEGER*2 left, right, bottom, top
```

Description

The **getscrmask** subroutine returns the dimensions of the current screenmask (the top of the screenmask stack) and copies these dimensions to the location variables specified as parameters. The *left*, *right*, *bottom*, *top* parameters are the addresses of four memory locations assigned the left, right bottom, and top coordinates of the screenmask.

Note: This subroutine cannot be used to add to a display list.

Parameters

left	Specifies the memory location into which the system copies the *x* coordinate (in pixels) of the left side of the screenmask.
right	Specifies the memory location into which the system copies the *x* coordinate (in pixels) of the right side of the screenmask.
bottom	Specifies the memory location into which the system copies the *y* coordinate (in pixels) of the bottom side of the screenmask.
top	Specifies the memory location into which the system copies the *y* coordinate (in pixels) of the top side of the screenmask.

Example

The example C language program **prompt.c** uses the **getscrmask** subroutine to get the current screenmask so that it can be restored later.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Pushing the viewport onto the viewport stack with the **pushviewport** subroutine.

Popping the viewport stack with the **popviewport** subroutine.

Defining a rectangular 2-D clipping mask with the **scrmask** subroutine.

Graphics Library Overview, Querying the System, and Using Viewports and Screenmasks.

---

getsize Subroutine

Purpose

Returns the size of a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void getsize(Int32 * x, Int32 * y)
```

FORTRAN Syntax

```
SUBROUTINE GETSIZ(x, y)
INTEGER*4 x, y
```

Description

The **getsize** subroutine returns the size of the current window. A window must be open for this subroutine to work.

Call the **winopen** subroutine to open a window, or the **winset** subroutine to choose the current window.

Note: This subroutine cannot be used to add to a display list.

Parameters

x Specifies a pointer to the location into which to copy the width (in pixels) of the window.
y Specifies a pointer to the location into which to copy the height (in pixels) of the window.

Example

The example C language program **paint.c** uses the **getsize** subroutine to determine the size of a window.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Obtaining the position of a window with the **getorigin** subroutine.

Specifying the maximum size of a window with the **maxsize** subroutine.

Specifying the minimum size of a window with the **minsize** subroutine.

Constraining the size of a window with the **prefsize** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Creating a window with the **winopen** subroutine.

Setting the current window with the **winset** subroutine.

Creating and Managing Windows.

---

getsm Subroutine

Purpose

Returns the shading model the system uses to draw filled polygons.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getsm( )
```

FORTRAN Syntax

```
INTEGER*4 GETSM
```

Description

The **getsm** subroutine returns the shading model that the system uses to draw filled polygons. The returned value of this function indicates which shading model is now active.

Note: This subroutine cannot be used to add to a display list.

Return Values

There are two possible return values:

C	FORTRAN	Description
FLAT	FLAT	The system draws a filled polygon with a constant color across the entire surface of the polygon.
GOURAUD	GOURAU	The system draws a filled polygon with a color that varies as a linear interpolation of the colors at the polygon's vertices.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Selecting the shading model used to draw polygons with the **shademodel** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

getvaluator Subroutine

Purpose

Returns the current state of a valuator.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

Int32 getvaluator(Device *device***)**

FORTRAN Syntax

INTEGER*4 FUNCTION GETVAL(*device***)**
INTEGER*4 *device*

Description

The **getvaluator** subroutine returns the current value (an integer) of the valuator specified in the *device* parameter.

Note: This subroutine cannot be used to add to a display list.

Parameter

device Identifier of the valuator (such as MOUSEX or BPADX) to be read.

Return Value

The value stored at the device named by the *device* parameter.

Example

The example C language program **select1.c** uses the **getvaluator** subroutine to obtain the mouse coordinates whenever the left mouse button is pressed.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Reading a list of valuators with the **getdev** subroutine.

Assigning initial, minimum, and maximum values to a valuator with the **setvaluator** subroutine.

Graphics Library Overview, Controlling Queues and Devices, Using the Keyboard, and Querying the System.

---

getviewport Subroutine

Purpose

Returns the dimensions of the current viewport.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void getviewport(Screencoord * *left***, Screencoord *** *right***,**

Screencoord * *bottom***, Screencoord *** *top***)**

FORTRAN Syntax

```
SUBROUTINE GETVIE(left, right, bottom, top)
INTEGER*2 left, right, bottom, top
```

Description

The **getviewport** subroutine returns the dimensions of the current viewport (the top of the viewport stack) and copies these dimensions to the location variables specified as parameters. The *left*, *right*, *bottom*, and *top* parameters are the addresses of four memory locations assigned the left, right bottom, and top coordinates of the viewport.

Note: This subroutine cannot be used to add to a display list.

Parameters

left	Specifies the memory location in which to return the *x* coordinate (in pixels) of the left side of the viewport.
right	Specifies the memory location in which to return the *x* coordinate (in pixels) of the right side of the viewport.
bottom	Specifies the memory location in which to return the *y* coordinate (in pixels) of the bottom side of the viewport.
top	Specifies the memory location in which to return the *y* coordinate (in pixels) of the top side of the viewport.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Popping the viewport stack with the **popviewport** subroutine.

Pushing the viewport onto the viewport stack with the **pushviewport** subroutine.

Setting the viewport to the dimensions of the current window with the **reshapeviewport** subroutine.

Setting the area of the window used for all drawing with the **viewport** subroutine.

getXdpy or getXwid Subroutine

Purpose

Returns the Enhanced X-Windows connection for the given GL session, or the window ID of the current GL window.

Library

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Display *getXdpy()
Window getXwid()
```

FORTRAN Syntax

```
INTEGER *4 FUNCTION GETXDPY()
INTEGER *4 FUNCTION GETXWID()
```

Description

The **getXdpy** subroutine returns a pointer to a data structure that describes the Enhanced X-Windows connection being used by the GL library to communicate with the X server.

The **getXwid** subroutine returns the Enhanced X-Windows window ID of the current GL window.

The Enhanced X-Windows display connection, with the Enhanced X-Windows window ID, enables applications programs to use Enhanced X-Windows routines to modify a GL window. With this ability, application programs can do several things, including the following:

- Manipulate properties
- Work with resources
- Process events
- Develop widgets.

Not all possible combinations of GL and Enhanced X-Windows calls result in defined behavior, however. In particular, avoid the following usages:

- Do not use Enhanced X-Windows drawing routines to draw into a GL window. The screen results are undefined. Use Enhanced X-Windows drawing routines to draw into Enhanced X-Windows only.
- Use only the GL **loadXfont** subroutine to load X fonts for use with the GL **charstr** subroutine. Use the **XLoadFont** or **XLoadQueryFont** subroutine only if you plan to use X text-rendering routines to draw into an Enhanced X-Windows window.
- Do not change, modify, or delete GL internal properties associated with GL windows. Doing so causes unpredictable behavior.
- Do not grab or ungrab pointers, keyboards, or the server while in GL fullscreen mode. Doing so causes unpredictable behavior.
- Events associated with GL windows can be obtained through the Enhanced X-Windows event queue. However, if you choose to obtain events in this manner, do not also use the GL event queue. Doing so

causes unpredictable results. An application can use both the GL and Enhanced X-Windows event queues within the same executable if the GL queue is used only for GL events and the Enhanced X-Windows event queue is used only for Enhanced X-Windows events.

The previous brief restrictions are explained more fully in Using Enhanced X-Windows Calls with GL Subroutines in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

All GL windows opened by a client normally share one Enhanced X-Windows connection. GL rendering can be performed only locally; that is, a GL application must execute on the same physical machine as that which contains the display adapter. GL does not support a client-server model of network graphics programming.

Return Values

getXdpy Returns a pointer to the Enhanced X-Windows Display structure.
getXwid Returns the Enhanced X-Windows window ID.

Example

Example programs showing the usage of the **getXdpy** and **getXwid** subroutines can be found in the **/usr/lpp/GL/examples** directory.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 supports RGB and color index modes by creating separate separate X11 windows—one window is permanently in RGB mode, and the other is permanently in color index mode. Both of these windows are wrapped in a single parent window. The **getXwid** subroutine returns the X11 window handle of that parent window. Standard X11 interfaces can be used to determine the child windows of the parent.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.
/usr/include/X11/Xlib.h Contains C language constant and variable type definitions for version X11 of Enhanced X-Windows.

Related Information

The **loadXfont** subroutine, **charstr** subroutine, **XReparentWindow** subroutine.

Using Enhanced X-Windows Calls with GL Subroutines in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

getwritemask Subroutine

Purpose

Returns the current writemask.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 getwritemask( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION GETWRI
```

Description

The **getwritemask** subroutine returns the color map mode writemask. Independently settable writemasks exist for the overlay, underlay, and main frame buffers.

The returned value of this function is an integer with up to 12 significant bits, one for each available bitplane. When a bit is set to zero in the writemask, the corresponding bitplane is read only.

This subroutine is intended for user in color map mode only. To get the RGB mode writemask, use the **gRGBmask** subroutine.

Note: This subroutine cannot be used to add to a display list.

Return Value

The writemask for the current drawing mode.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the target frame buffer for the drawing subroutines with the **drawmode** subroutine.

Returning the current RGB writemask with the **gRGBmask** subroutine.

Granting write access to a subset of available bitplanes with the **RGBwritemask** subroutine.

Granting write permission to a subset of available bitplanes with the **writemask** subroutine.

Configuring the Frame Buffer and Querying the System.

Writemasks and Logical Operations and Working in Color Map and RGB Modes.

getzbuffer Subroutine

Purpose
Determines whether z-buffering is on or off.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 getzbuffer( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION GETZBU
```

Description
The **getzbuffer** subroutine returns the status of the z-buffer. The z-buffer option to the High-Performance 3-D Color Graphics Processor must be installed before the z-buffer can be turned on.

Note: This subroutine cannot be used to add to a display list

Return Values

False(0)	Z-buffering off (the default value).
True(1)	Z-buffering on.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Clearing the z-buffer with the **zclear** subroutine.

Initializing z-buffer mode with the **zbuffer** subroutine.

Graphics Library Overview, Configuring the Frame Buffer, Querying the System, and Removing Hidden Surfaces.

gexit Subroutine

Purpose
Terminates a graphics program.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void gexit( )
```

FORTRAN Syntax

Description
The **gexit** subroutine is the final graphics routine in a program. This subroutine waits for the graphics pipeline to empty, then frees all GL data structures.

After the **gexit** subroutine, a process can no longer call any routines that require the graphics to be initialized.

> **Note:** This subroutine cannot be used to add to a display list.

The **gexit** subroutine unmaps and eliminates all windows, closes the display connection, detaches all GL shared memory segments, cleans up and closes all GL graphics hardware resources, and eliminates GL processes and kernel context.

The **gexit** subroutine leaves the calling process in a state similar to that in which it would have been if no GL routine had ever been callled.

Example
The example C language program **scrn_rotate.c** uses the **gexit** subroutine to end graphics processing.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Resetting all global state attributes to their initial values with the **greset** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Clearing, Resetting, and Initializing GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

ginit Subroutine

Purpose
Initializes the graphics system.

Library
Graphics Library (**libgl.a**)

C Syntax
```
void ginit( )
```

FORTRAN Syntax
```
SUBROUTINE GINIT
```

Description
The **ginit** subroutine initializes the graphics environment to its default values for the global state attributes and creates a window that covers the screen. The **ginit** subroutine queues the REDRAW window manager device.

Call the **ginit** subroutine once, before any other GL subroutine.

The recommendation is to use the **winopen** subroutine for initialization functions to take advantage of the window manager and to avoid unexpected events in the event queue.

Note: This subroutine cannot be used to add to a display list.

Example
The example C language program **prompt.c** uses the **ginit** subroutine to do a basic graphics setup.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Initializing the graphics system without changing the color map with the **gbegin** subroutine.

Terminating a graphics program with the **gexit** subroutine.

Resetting all global state attributes to their initial values with the **greset** subroutine.

Creating a new window with the **winopen** subroutine.

Setting Drawing Attributes and Controlling Queues and Devices.

Understanding the Hardware Used by GL, Clearing, Resetting, and Initializing GL, and Windows and Input Control Overview.

---

glcompat Subroutine

Purpose

Controls backwards compatibility modes.

C Syntax

void glcompat (Int32 *mode*, **Int32** *value*)

FORTRAN Syntax

SUBROUTINE GLCOMPAT (*mode, value*)
INTEGER*4 *mode, value*

Parameters

mode Token representing the compatibility mode to be changed.
value The value that the compatibility mode should have.

Functional Description

This subroutine governs various compatibility modes for the GL subsystem. Applications should use this subroutine judiciously to minimize porting headaches while maximizing useful function. The following **glcompat** modes can also be set with environment variables with the same corresponding names.

Mode GLC_CREATE_OVERLAY
Default FALSE
Function On the POWERgraphics GXT1000 adapter, this mode controls whether the **winX** subroutine will automatically create an overlay window. When this mode is set to TRUE, and the XID passed to **winX** is not an overlay window, then the **winX** routine will create an overlay window and make it a child of the indicted X window. Although the overlay window will be a distinct X11 window, internally, it will be used to emulate overlay plane functions. It can be accessed via the **drawmode** subroutine and can be drawn into with the usual GL functions.
When this mode is set to FALSE, the **winX** routine will simply accept the indicated X Window, and create and attach a GL context to that window. The **drawmode** subroutine should not be used with windows created by **winX** when this mode is FALSE.
This mode has no effect on graphics adapters other than the POWERgraphics GXT1000.

Mode GLC_ISSUE_OVERDRAW
Default FALSE
Function On the POWERgraphics GXT1000 adapter, this mode controls how exposure events for the overlay planes are handled. When the mode is TRUE, the REDRAW_OVERLAY event will be issued whenever the overlay planes are uncovered and need redrawing. The REDRAW event will not be issued if the main frame buffer window has not been damaged.
When the mode is FALSE, damage to the overlay planes will be reported with the REDRAW event. Note that if the main frame buffer has also been damaged, it may happen that two REDRAW events are reported for the window.
Set this mode to FALSE if your application does not handle REDRAW_OVERLAY events, but needs to find out when the overlays need to be redrawn.
This mode has no effect on graphics adapters other than the POWERgraphics GXT1000.

Mode GLC_FB_FULLSCREEN
Default FALSE

Function On the POWERgraphics GXT1000 adapter, this mode enables the **fullscrn** routine to affect the main frame buffer. By default, the **fullscrn** subroutine applies only to the overlay bit-planes.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

greset Subroutine

Purpose

Resets all global state attributes to their initial values.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void greset( )
```

FORTRAN Syntax

```
SUBROUTINE GRESET
```

Description

The **greset** subroutine resets all global state attributes to their initial values. This subroutine can be called at any time to reset the attributes.

The following table lists the global state attributes:

Global State Attributes	
Attribute	**Initial Value**
backface mode	off
blinking	off
buffer mode	single
color	undefined
color map mode	one map
concave	off
cursor	0 (arrow)
depth range	-0x800000,+0x7FFFFF (On Supergraphics Processor Subsystem: 0x0,0x1FFFFF)
depthcue mode	off

display mode	color map
drawmode	NORMALDRAW
font	0
linestyle	0 (solid)
linewidth	1 pixel
logical operation	LO_SRC
lsrepeat	1
pattern	0 (solid)
picking size	10x10 pixels
RGB color	undefined
RGB shaderange	undefined
RGB writemask	undefined
shademodel	GOURAUD
shaderange	0,7,-0x800000,+0x7FFFFF (On Supergraphics Processor Subsystem: 0x0,0x1FFFFF)
viewport	entire window
writemask	all planes enabled
z-buffer mode	off

Note: Font 0 (zero) is a Helvetica-like font.

The **greset** subroutine puts a 2-D orthographic projection transformation on the matrix stack with left, right, bottom, and top set to the boundaries of the screen. The subroutine also turns on the cursor and ties it to MOUSEX and MOUSEY.

The **greset** subroutine removes all button, valuator, and keyboard entries from the event queue and discards them. Each button is set to FALSE and untied from valuators. Each valuator (and in particular MOUSEX) is set to XMAXSCREEN/2 with range 0 to XMAXSCREEN. MOUSEY is set to YMAXSCREEN/2 and has range 0 to YMAXSCREEN.

The **greset** subroutine also defines certain entries in the color map, as follows:

Color Map Entries			
	RGB Value		
Name	**Red**	**Green**	**Blue**
BLACK	0	0	0
RED	255	0	0
GREEN	0	255	0
YELLOW	255	255	0
BLUE	0	0	255
MAGENTA	255	0	255
CYAN	0	255	255
WHITE	255	255	255

Note: This subroutine cannot be used to add to a display list.

Example

The example C language program **vlsi.c** uses the **greset** subroutine to reset all global attributes on any keyboard event.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Reconfiguring the graphics system with the **gconfig** subroutine.

Initializing the graphics system with the **ginit** subroutine.

Creating a new window with the **winopen** subroutine.

Controlling Queues and Devices, Creating a Cursor, Understanding the Hardware Used by GL, Setting Drawing Attributes, Windows and Input Control Overview, Working with Coordinate Systems.

Configuring the Frame Buffer, Clearing, Resetting, and Initializing GL.

---

gRGBcolor Subroutine

Purpose

Gets the current RGB color values.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void gRGBcolor

(Int16 *red***, Int16** *green***, Int16** *blue***)**

FORTRAN Syntax

SUBROUTINE GRGBCO(*red***,** *green***,** *blue***)**

INTEGER*2 *red***,** *green***,** *blue*

Description

The **gRGBcolor** subroutine gets the current RGB color values and copies them into the parameters.

1. This subroutine cannot be used to add to a display list.

2. This subroutine is available in RGB mode. It will not function in color map mode.

Parameters

red	Specifies the pointer to the location into which to copy the current red value.
green	Specifies the pointer to the location into which to copy the current green value.
blue	Specifies the pointer to the location into which to copy the current blue value.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.

/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current color with the **getcolor** subroutine.

Getting a copy of the RGB values for a color map entry with the **getmcolor** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Setting a display mode that bypasses the color map with the **RGBmode** subroutine.

Understanding the Adapter, Working in Color Map and RGB Modes, Setting Drawing Attributes, and Querying the System.

---

gRGBmask Subroutine

Purpose

Returns the current RGB writemask.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void gRGBmask
(Int16 * redmask, Int16 * greenmask, Int16 * bluemask)
```

FORTRAN Syntax

```
SUBROUTINE GRGBMA(redmask, greenmask, bluemask)
INTEGER*2 redmask, greenmask, bluemask
```

Description

The **gRGBmask** subroutine gets the current RGB writemask as three 8-bit masks and copies them into the parameters. The subroutine places masks in the low order 8-bits of the locations *redmask*, *greenmask*, and *bluemask*. The system must be in RGB mode when this routine executes.

This subroutine is intended for use in RGB mode only. To get the writemask when in color map mode, use the **getwritemask** subroutine.

Note: This subroutine cannot be used to add to a display list.

Parameters

redmask	Specifies the pointer to the location into which the system is to copy the current red writemask value.
greenmask	Specifies the pointer to the location into which the system is to copy the current green writemask value.
bluemask	Specifies the pointer to the location into which the system is to copy the current blue writemask value.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current writemask with the **getwritemask** subroutine.

Granting write access to a subset of available bitplanes with the **RGBwritemask** subroutine.

Specifying the RGBA writemask with a single packed integer with the **wmpack** subroutine.

Granting write permission to available bitplanes with the **writemask** subroutine.

Configuring the Frame Buffer, Querying the System, and Working in Color Map and RGB Modes.

Writemasks and Logical Operations and Working in Color Map and RGB Modes.

---

gselect Subroutine

Purpose

Puts the system in selecting mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 gselect(Int16  buffer[ ], Int32  numnames)
```

FORTRAN Syntax

```
SUBROUTINE GSELEC(buffer, numnames)
INTEGER*4 numnames
INTEGER*2 buffer(numnames)
```

Description

The **gselect** subroutine puts the system in selecting mode. In this mode, the system notes when a drawing routine intersects the selecting volume and writes the contents of the name stack to the specified buffer.

If you push a name onto the name 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.

The **gselect** and **pick** subroutines differ only in the manner in which the pick or select volume is specified. The **pick** subroutine uses a volume (default 10x10 pixels) centered on the current cursor location, while the **gselect** subroutine uses the unit cube (1x1x1) in modeling coordinates, thus employing the current viewing matrix in determining the selecting volume.

Nothing is drawn to the screen when the system is in selecting mode. Instead, drawing commands are piped to the select mechanism and used to determine the pick or select region hits.

With one exception, all drawing routines cause hits, including clear, points, lines, polygons, arcs, circles, curves, patches, and NURBS. The **charstr** subroutine does not cause a hit, although **cmov** and **cmov2** do cause hits.

To end selecting mode, call the **endselect** subroutine.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

buffer	Specifies the buffer into which the system is to save the contents of the name stack.
numnames	Specifies the maximum number of names to be saved.

Example

The example C language program **select1.c** uses the **gselect** subroutine to enter selecting mode.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning off picking mode with the **endpick** subroutine.

Turning off selecting mode with the **endselect** subroutine.

Initializing the name stack with the **initnames** subroutine.

Loading the name on top of the name stack with the **loadname** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Setting the dimensions of the picking region with the **picksize** subroutine.

Popping a name off the name stack with the **popname** subroutine.

Pushing a new name onto the name stack with the **pushname** subroutine.

Graphics Library Overview, Picking and Selecting Overview, and Working with Coordinate Systems.

---

gsync Subroutine

Purpose

Waits for a vertical retrace period.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void gsync( )
```

FORTRAN Syntax

```
SUBROUTINE GSYNC
```

Description

The **gsync** subroutine waits for the next vertical retrace. Because this subroutine does not return until vertical retrace begins, the calling process is effectively blocked until that time.

This subroutine is useful for pacing the drawing when in single buffer mode. If the amount of drawing to be done is small, this subroutine can be used to achieve a limited amount of smooth animation in single buffer mode. For high-quality, smooth-animation, double-buffer mode should be used with the **swapbuffers** subroutine.

Note: This subroutine cannot be used to add to a display list.

Example

The **worms.c** example C language program uses the **gsync** subroutine to help smooth the display while it is changing the frame buffer.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 does not support the **gsync** subroutine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Setting the display mode to single buffer mode with the **singlebuffer** subroutine.

Exchanging the front and back buffers in double buffer mode with the **swapbuffers** subroutine.

Understanding the Hardware Used by GL and Working in Color Map and RGB Modes.

Creating Animated Scenes.

---

gversion Subroutine

Purpose

Returns graphics hardware and library version information.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 gversion(Char8 * v)
```

FORTRAN Syntax

```
INTEGER*4 GVERSI(v,  length)
CHARACTER*(*) v
INTEGER*4 length
```

Description

The **gversion** subroutine fills the buffer, identified by the *v* parameter, with a null-terminated string that specifies the graphics hardware type and the version number of GL.

It is not necessary to open a GL window or to call the **ginit** subroutine before using the **getdesc** subroutine.

The **gversion** subroutine returns a value of -1 if GL is not supported on this particular hardware configuration.

Note: This subroutine cannot be used to add to a display list.

Attention: Calling the **gversion** subroutine before calling the **winopen** subroutine on the POWERgraphics GTO and the POWERstation 730, results in a core dump when the initial **winopen** subroutine call is made.

Return Value

In the following values table, *m* and *n* indicate the major and minor release numbers of the installed software libraries:

Installed Adapter	String Returned
3-D Color Graphics Processor	GL:CGP-*m.n*
Supergraphics Processor	GL:GTO-*m.n*
POWER GTO adapter	GL:GTO-*m.n*
POWER Gt4 adapter	GL:GT4-*m.n*
POWER Gt4x adapter	GL:GT4-*m.n*

Parameters

v Specifies a buffer into which you can copy a string. Reserve at least a 12-character buffer.
length Specifies the length of the string.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Creating a new window with the **winopen** subroutine.

Understanding the Hardware Used by GL and Clearing, Resetting, and Initializing GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

iconsize Subroutine

Purpose

Specifies the icon size of a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void iconsize(Int32  x, Int32  y)
```

FORTRAN Syntax

```
SUBROUTINE ICONSI(x, y)
integer*4 x, y
```

Description

The **iconsize** subroutine specifies the size of a window icon as *x* pixels by *y* pixels. If a window has an icon size, the window manager reshapes the window to be that size and sends a REDRAWICONIC token to the graphics queue when the user stores that window. Windows without an icon size are handled by the window manager with the appropriate default behavior.

To assign a new window an icon size, call the **iconsize** subroutine before opening the window. To give an existing window an icon size, use the **iconsize** subroutine with the **winconstraints** subroutine.

Any application using the **iconsize** subroutine should also call the **qdevice** subroutine to queue the tokens WINFREEZE and WINTHAW after opening the window.

In the current implementation, the **iconsize** subroutine does not set the size of the icon. The actual size and shape of the icon are controlled by the window manager application.

Note: This subroutine cannot be used to add to a display list.

Parameters

x Specifies the width of the window icon in pixels.
y Specifies the height of the window icon in pixels.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Obtaining the size of the window with the **getsize** subroutine.

Specifying the title of a window icon with the **icontitle** subroutine.

Specifying the maximum size of a window with the **maxsize** subroutine.

Specifying the minimum size of a window with the **minsize** subroutine.

Constraining the size of a window with the **prefsize** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

icontitle Subroutine

Purpose
Specifies the icon title for the current window.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void icontitle(Char8 * name)
```

FORTRAN Syntax
```
SUBROUTINE ICONTI( name,  length)
CHARACTER *(*) name
INTEGER*4 length
```

Description
The **icontitle** subroutine specifies the string displayed on an icon if the window manager draws that window's icon.

Note: This subroutine cannot be used to add to a display list.

Parameters
name	Specifies a pointer to the string containing the icon title.
length	Specifies the length of the string containing the icon title.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Specifying the size of a window icon with the **iconsize** subroutine.

Adding a title bar to the current window with the **wintitle** subroutine.

Creating and Managing Windows.

initnames Subroutine

Purpose
Initializes the name stack.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void initnames( )
```

FORTRAN Syntax
```
SUBROUTINE INITNA
```

Description
The **initnames** subroutines initializes the name stack for use during picking or selecting.

Example
The example C language program **pick1.c** calls the **initnames** subroutine to clear all of the names from the name stack.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Putting the system in selecting mode with the **gselect** subroutine.

Loading a name onto the top of the name stack with the **loadname** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Popping a name off the name stack with the **popname** subroutine.

Pushing a new name onto the name stack with the **pushname** subroutine.

Graphics Library Overview and Picking and Selecting Overview.

isobj Subroutine

Purpose
Indicates whether a given object number actually identifies an object.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 isobj(Int32  object)
```

FORTRAN Syntax
```
LOGICAL FUNCTION ISOBJ(object)
INTEGER*4 object
```

Description
The **isobj** subroutine indicates whether a given object number actually identifies an object. The returned value is either True, if the object number is already in use, or False, if it is not.

Note: This editing subroutine itself cannot be added to a display list.

Parameter
object Specifies the object identifier to test.

Return Values
True Object number is already in use.
False Object number is not in use.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information
Returning a unique integer for use as an object identifier with the **genobj** subroutine.

Establishing the uniqueness of a tag number with the **istag** subroutine.

Creating an object with the **makeobj** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

isqueued Subroutine

Purpose
Indicates whether the specified device is enabled for queuing.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 isqueued(Int16  device)
```

FORTRAN Syntax
```
LOGICAL FUNCTION ISQUEU(device)
INTEGER*4 device
```

> **Note:** For FORTRAN users, this function accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description
The **isqueued** subroutine indicates whether the specified device is enabled for queuing.

> **Note:** This subroutine cannot be used to add to a display list.

Parameter
device Specifies the identifier for the device you want to test (for example, MOUSEX or BPADX).

Return Values
True	Enabled for queuing.
False	Not enabled for queuing.

Example
The example C language program **prompt.c** uses the **isqueued** subroutine to determine whether the keyboard is enabled.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.

/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling an input device for event queuing with the **qdevice** subroutine.

Reading the first entry in the event queue with the **qread** subroutine.

Disabling an input device for event queuing with the **unqdevice** subroutine.

Graphics Library Overview, Controlling Queues and Devices, Using the Keyboard, and Querying the System.

---

istag Subroutine

Purpose

Indicates whether a given tag is used within the current open object.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 istag(Int32  tag)
```

FORTRAN Syntax

```
LOGICAL FUNCTION ISTAG(tag)
INTEGER*4 tag
```

Description

The **istag** subroutine indicates whether a given tag number actually identifies an existing tag. The returned value is either True (1), if the tag is already in use, or False (0), if it is not. If there is no current open object, the result is undefined.

Note: This editing subroutine, itself, cannot be used to add to a display list.

Parameter

tag Specifies the tag to test.

Return Values

True	Tag number is already in use.
False	Tag number is not in use.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning a unique integer for use as a tag number with the **gentag** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

keepaspect Subroutine

Purpose

Specifies the aspect ratio of a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void keepaspect(Int32  x, Int32  y)
```

FORTRAN Syntax

```
SUBROUTINE KEEPAS(x, y)
INTEGER*4 x, y
```

Description

The **keepaspect** subroutine specifies the aspect ratio of a window. It is called at the beginning of a graphics program, but only takes effect when the **winopen** subroutine is called. The resulting window maintains the aspect ratio specified in the **keepaspect** subroutine, even if the window changes size.

For example, `keepaspect(1,1)` always results in a square window. The **keepaspect** subroutine can also be called in conjunction with the **winconstraints** subroutine to modify the enforced aspect ratio after the window is created.

With the **keepaspect** subroutine, the programmer can prevent the user from resizing a window to an aspect ratio that is different from the specified ratio.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

x Specifies the horizontal proportion of the aspect ratio.
y Specifies the vertical proportion of the aspect ratio.

Example

The example C language program **colored.c** uses the **keepaspect** subroutine to restrict a window to a specific aspect ratio.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Constraining the location and size of a window with the **prefposition** subroutine.

Constraining the size of a window with the **prefsize** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

lampoff or lampon Subroutine

Purpose

Turns the keyboard display lights off or on.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lampoff(Int8  lamps);

void lampon(Int8  lamps);
```

FORTRAN Syntax

```
SUBROUTINE LAMPOF(lamps)
CHARACTER*1 lamps
SUBROUTINE LAMPON(lamps)
CHARACTER*1 lamps
```

Description

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

Note: This subroutine cannot be used to add to a display list.

Parameter

lamps Indicates the mask that specifies which lamps to manipulate. If a bit is set, then the corresponding keyboard lamp is either on or off.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning on or off the keyboard click with the **clkon** or **clkoff** subroutine.

Ringing the keyboard bell with the **ringbell** subroutine.

Setting the duration of the keyboard bell sound with the **setbell** subroutine.

Graphics Library Overview, Using the Keyboard.

---

lgetdepth Subroutine

Purpose

Gets the distance of the near and far clipping planes.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lgetdepth(Int32 * near, Int32 * far)
```

FORTRAN Syntax

```
SUBROUTINE LGETDE(near, far)
INTEGER*4 near, far
```

Description

The **lgetdepth** subroutine gets the distance of the near and far clipping planes and writes them into the *near* and *far* parameters. Set these distances using the **lsetdepth** subroutine.

Note: This subroutine cannot be used to add to a display list.

Parameters

near Specifies a pointer to the location into which to write the distance of the near clipping plane.
far Specifies a pointer to the location into which to write the distance of the far clipping plane.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the viewport depth range with the **lsetdepth** subroutine.

Graphics Library Overview, Querying the System, and Working with Coordinate Systems.

---

linesmooth Subroutine

Purpose

Turns line antialiasing on and off.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void linesmooth(Int32  mode )
```

FORTRAN Syntax

```
SUBROUTINE LINESM(mode)
INTEGER*4 mode
```

Description

The **linesmooth** subroutine allows the drawing of antialiased lines in color map and RGB modes.

In RGB mode, the current color is blended with the contents of the frame buffer in a read-modify-write operation. That is, the new pixel color is a calculated blend of the old pixel color and the current color, based on the percentage of overlap of the line with the pixel.

In color map mode, the low-order 4 bits of the current color index are replaced with a value representing the pixel coverage. Therefore, a color ramp must be loaded, with the low-order 4 bits of the color ramp blending between the foreground and background colors.

This type of color-index antialiasing hardware design allows the user to implement one of several antialiasing algorithms. One algorithm draws monocolored lines on a multicolored background; the other draws multicolored lines on a monochrome background. Refer to the section on Smoothing Jagged Lines with Antialiasing in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* for examples and further detail.

Only solid (nonpatterned), unit-width lines are supported. That is, for antialiased lines to be drawn correctly, set `linewidth=1` using the **linewidth** subroutine, `linestyle=0xffff` using the **deflinestyle** subroutine, and `lsrepeat=1` using the **lsrepeat** subroutine.

1. In order for antialiased lines and points to be displayed as visually smooth, gamma correction *must* be performed. A gamma correction factor in the range of 2.4 to 2.7 is suggested. If gamma correction is not performed, lines are not displayed as smooth, but exhibit a *roping* or *braiding* effect, as if the line were composed of separate, intertwining strands. For more information on gamma-correcting antialiased lines, see Smoothing Jagged Lines with Antialiasing in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.
2. Only solid, single, pixel-wide lines can be antialiased, and they cannot be used in conjunction with shading processor functions such as z-buffer, light, shade, and depth cue.

Parameter

mode Sets line-drawing mode as follows:

Mode Constants		
C	**FORTRAN**	**Description**
SML_ON	SMLON	Enables smooth lines.
SML_OFF	SMLOFF	Disables smooth lines.

Example

The example C language program **alias.c** uses the **linesmooth** subroutine to draw antialiased lines. Other examples that use the **linesmooth** subroutine include the **alias_fore.c** and **alias_back.c** example programs.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

1. The 3-D Color Graphics Processor supports antialiased lines only in color map mode. Only solid (nonpatterned) unit width lines are supported. On this adapter, the value `0xf` represents maximum pixel coverage, and `0x0` represents minimum pixel coverage. That is, the following color ramp should be loaded:

3-D Display Graphics Processor Color Ramp	
Low-Order 4 Bits	**Ramp**
0000	(1/16) * foreground color + (15/16) * background color
0001	(2/16) * foreground color + (14/16) * background color
0010	(3/16) * foreground color + (13/16) * background color
. . . .	. . . . .
1111	(16/16) * foreground color + (0/16) * background color

For the 3-D Color Graphics Processor, the appearance of the intersections of color-index-mode antialiased lines and points can be significantly improved by setting the value of the **zsource** subroutine to ZSRC_COLOR and the value of the **zfunction** subroutine to ZF_GREATER. Refer to the subsection on improving intersections within Smoothing Jagged Lines with Antialiasing in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* for more details.

2. The Supergraphics Processor supports antialiased lines in RGB mode only. Only solid (nonpatterned) unit width lines are supported.
3. The POWER Gt4 and POWER Gt4x adapters support antialiasing in both RGB and color index modes. For these adapters, RGB-mode antialiased lines are drawn significantly slower than normal lines. For these adapters, the appearance of the intersections of color-index-mode antialiased lines can be improved significantly by setting the value of the **logicops** subroutine to LO_MAX.
4. The 8-bit POWERstation 730 and the 8-bit POWERgraphics GTO adapter do not support the drawing of antialiased lines. The 24-bit option must be installed for antialiased line support.
5. When anti-aliased lines and points are drawn in RGB mode, a coverage factor (alpha value) for pixels on and near the anti-aliased objects is computed. This coverage factor is used to blend the color of the anti-aliased point or line with the contents of the frame buffer. The way in which the blending occurs is controlled by the **blendfunctio**n subroutine. When rendering anti-aliased objects in RGB mode, set the blendfunction to an appropriate value.
6. The POWERgraphics GXT1000 supports anti-aliased line rendering in both color-index and RGB rendering modes.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying antialiasing of points with the **pntsmooth** subroutine.

Controlling the placement of point, line, and polygon vertices with the **subpixel** subroutine.

Smoothing Jagged Lines with Antialiasing, Understanding the Hardware Used by GL, Working in Color Map and RGB Modes, Setting Drawing Attributes.

---

linewidth Subroutine

Purpose

Specifies the linewidth.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void linewidth(Int16 *number***)**

FORTRAN Syntax

SUBROUTINE LINEWI(*number***)**
INTEGER*4 *number*

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **linewidth** subroutine specifies the width of a line. The default width is one pixel. Wide lines are centered as nearly as possible on the infinitely thin mathematical line.

> **Note:** This subroutine cannot be used to add to a display list.

Parameter

number Width of the line in pixels.

Example

The example C language program **prompt.c** uses the **linewidth** subroutine to draw a two-pixel thick border around a prompt string.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 supports line widths from 1 to 255.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a linestyle with the **deflinestyle** subroutine.

Returning the current linewidth with the **getlwidth** subroutine.

Selecting a linestyle with the **setlinestyle** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Drawing NURBS Curves and Surfaces, Drawing Wire Frame Curves and Surface Patches, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Drawing Rectangles, Circles, Arcs, and Polygons.

Imbind Subroutine

Purpose

Makes a material, light, or lighting model definition active.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lmbind(Int16  target, Int32  index)
```

FORTRAN Syntax

```
SUBROUTINE LMBIND(target, index)
INTEGER*4 target, index
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **lmbind** subroutine takes a previously defined material, light, or lighting model and makes it active. Up to eight lights can be active (turned on) at the same time, but only one lighting model or material can be turned on at any time. Therefore, binding a material or lighting model automatically unbinds (deactivates) the previous lighting model or material.

The definition of a light, lighting model, or material (created previously with the **lmdef** subroutine) is not destroyed when one of these is unbound. The definition remains and can be rebound at a later time.

Notes:

1. Some of the properties of the currently bound material can be changed on the fly with the **lmcolor** subroutine, which provides a highly efficient path for temporarily changing material properties. Using the **lmcolor** subroutine is much more efficient than employing a combination of the **lmdef** and **lmbind** subroutines.
2. Lighting cannot be turned on and does not work if the matrix mode is not set to MVIEWING. The matrix mode can be changed with the **mmode** subroutine.
3. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See Hardware Considerations in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.)

Parameters

index Specifies the index into the table of previously defined materials, lights, or lighting models. Created with the **lmdef** subroutine.

target Specifies the target of the bind. There are 10 valid constants tokens that can be used for this parameter:

Constants		
C	**FORTRAN**	**Description**
MATERIAL	MATERI	Indicates that the material definition passed by the *index* parameter should become the currently active material. Lighting can be turned off by binding index 0 to a material.

LIGHT0, LIGHT1, LIGHT2, LIGHT3, LIGHT4, LIGHT5, LIGHT6, LIGHT7	LIGHT0, LIGHT1, LIGHT2, LIGHT3, LIGHT4, LIGHT5, LIGHT6, LIGHT7	Indicate that the light definition passed by the *index* parameter should be bound to the respective light. Each light is distinct from the others and can be on or off. A light can be turned off by binding index 0 to it.
LMODEL	LMODEL	Indicates that the lighting model definition passed by the *index* parameter is the current lighting model. There can be only one current lighting model.

Example

The example C language program **cylinder2.c** uses the **lmbind** subroutine to use the materials, lights, and lighting model defined with the **lmdef** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Materials with different material properties on each side are now supported on the Gt4 class of graphics adapters and the GXT1000. Support for two-sided lighting is accomplished with the following token:

Target BACKMATERIAL
Function Indicates that the material definition passed by the index parameter should be used as the active material for back-facing polygons. Two-sided lighting must be enabled in order for this material definition to be used.

If two-sided lighting is enabled, and BACKMATERIAL is bound to a non-zero index, then the BACKMATERIAL properties are used for back-facing polygons, and MATERIAL properties are used for front-facing polygons. If two-sided lighting is enabled, and a zero index is bound to BACKMATERIAL, then the MATERIAL properties are used for both sides. If two-sided lighting is disabled, the material bound to BACKMATERIAL is not used.

Two-sided lighting applies only to polygons and triangles; it does not apply to points, lines and characters, where only the MATERIAL properties are used.

Target LIGHT0
Function The light position is transformed by the current matrix at the time the the light is bound. That is, the actual position that the light will have in world coordinates is determined by its defined position and the current matrix when it is bound. Subsequent changes to the matrix do not affect the light position. The position value stored in the light definition (as defined with the **lmdef** subroutine) is not affected or changed.
Description Lighting is disabled when index 0 (zero) is bound to either the MATERIAL or to the LMODEL (or both). Both the material and the lighting model must be bound to a non-zero index for lighting to be enabled. Even if all lights are off, lighting is still enabled, although only the emmisive material properties contribute to the lit color.

Note: If a light does not appear to be in the correct location, be sure that the current matrix had an appropriate value when the light was bound. The light position is transformed by the current matrix when the light is bound.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying RGBA colors with a single packed 32-bit integer using the **cpack** subroutine.

Changing the target of the color commands with the **lmcolor** subroutine.

Making a new material, light source, or lighting model active with the **lmdef** subroutine.

Setting the current matrix mode with the **mmode** subroutine.

Specifying a normal vector with the **n3f** subroutine.

Updating the current normal vector with the **normal** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Graphics Library Overview, Creating Lighting Effects, and Setting Pipeline Options.

---

lmcolor Subroutine

Purpose

Changes the target of the color commands while lighting is active.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lmcolor(Int32  mode)
```

FORTRAN Syntax

```
SUBROUTINE LMCOLO(mode)
INTEGER*4 mode
```

Description

The **lmcolor** subroutine changes the target of the RGB color subroutines while lighting is active.

Properties of the currently bound material can be changed by calls to the **lmdef** subroutine, but that subroutine is relatively slow to execute. The **lmcolor** subroutine 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. Changes made by the **lmcolor** subroutine are lost when a new material is bound.

The standard RGB color subroutines (**RGBcolor**, **c**, **cpack**) are used to change material properties efficiently. The **lmcolor** subroutine specifies which material property is to be affected by these subroutines.

Because the **lmcolor** subroutine is effective only when lighting is active, the standard color subroutines are used to change the current color when lighting is off.

Note: The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)

Parameter

mode Specifies the name of the mode to be used in conjunction with RGB color subroutines. Possible modes are:

Mode Constants		
C	**FORTRAN**	**Description**
LMC_COLOR	LMCCOL	Color subroutines 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 is lighted. **LMC_COLOR** is the default mode.
LMC_EMISSION	LMCEMI	Color subroutines set the current EMISSION color property of the current material.
LMC_AMBIENT	LMCAMB	Color subroutines set the current AMBIENT color property of the current material.
LMC_DIFFUSE	LMCDIF	Color subroutines set the current DIFFUSE color property of the current material. Alpha, the fourth color component specified by RGB color subroutines, sets the ALPHA property of the current material.
LMC_SPECULAR	LMCSPE	Color subroutines set the current SPECULAR color property of the current material.
LMC_AD	LMCAD	Color subroutines set the current DIFFUSE and AMBIENT color properties of the current material. Alpha, the fourth color component specified by RGB color subroutines, sets the ALPHA property of the current material.
LMC_NULL	LMCNUL	RGB subroutines are ignored.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying RGBA colors as vectors using the **c** subroutine.

Specifying RGBA colors with a single packed 32-bit integer using the **cpack** subroutine.

Making a new material, light, or lighting model definition active with the **lmbind** subroutine.

Defining a new material, light, or lighting model with the **lmdef** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Graphics Library Overview, Creating Lighting Effects, Setting Pipeline Options, and Working in Color Map and RGB Modes.

---

lmdef Subroutine

Purpose

Defines a new material, light, or lighting model.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lmdef
(Int16  deftype, Int32  index,
Int16  numpoints, Float32  properties[ ])
```

FORTRAN Syntax

```
SUBROUTINE LMDEF(deftype, index, numpoints, properties)
INTEGER*4 deftype, numpoints, index
REAL properties(numpoints)
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when started from a FORTRAN program, although it accepts short integers when started from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **lmdef** subroutine defines a new material, light, or lighting model.

The type of definition (material, light, or lighting model) is specified by the *deftype* parameter. The definition is read from the *properties* array and stored in the appropriate definition table at the index specified by the *index* parameter.

You can make incremental changes to a material, light, or lighting model definition. Each call to the **lmdef** subroutine changes only the properties specified in the *properties* array.

Any property of any definition can be changed regardless of whether that definition is currently bound. Changes made to a definition that is currently bound by the **lmbind** subroutine are effective immediately.

Index 0 (zero) of the material, light, and lighting model definition tables contains predefined definitions. These predefined definitions have set all properties to their default values and cannot be changed. Their values are as follows:

Value	Function
DEFMATERIAL	Turns off lighting. Most efficient way to disable calculations. Equivalent to lighting model 0.
DEFLIGHT	Turns off lighting. Binding light 0 to a light turns off that light.
DEFLMODEL	Turns off lighting.

To turn off lighting, bind material 0 as the current material. You can also turn off lighting by binding lighting model 0 as the current lighting model, but this method is less efficient than binding material 0. To turn off a light, but not all lighting calculations, bind light definition 0 to the light you want to turn off.

There is a unique properties table for each category of definition created by this routine (materials, light sources, or lighting models). Indexes within each of these categories are independent. Valid entries for this parameter range from 1 to 65535. In each category, index 0 is reserved.

For maximum efficiency, use the default values for all properties. Lighting model performance is best if relatively few properties are changed from the default. A definition can be reset to all default values by calling the **lmdef** subroutine with the symbolic constant LMNULL as the first command token in the *properties* array.

The default material values are as follows:

EMISSION
0.0, 0.0, 0.0

AMBIENT
0.2, 0.2, 0.2

DIFFUSE
0.8, 0.8, 0.8

SPECULAR
0.0, 0.0, 0.0

SHININESS
0.0

ALPHA
1.0

COLORINDEXES
0.0, 127.5, 255.0

The default light values are as follows:

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 values are as follows:

AMBIENT	0.2, 0.2, 0.2
LOCALVIEWER	0.0
ATTENUATION	1.0, 0.0

Notes:

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)
2. This subroutine cannot be used to add to a display list.

Parameters

deftype
Category in which you want to create a new definition. There are three categories:

Constant		
C	**FORTRAN**	**Description**
DEFMATERIAL	DEFMAT	Indicates that this routine defines the properties of a material.
DEFLIGHT	DEFLIG	Indicates that this routine defines the properties of a light source.
DEFLMODEL	DEFLMO	Indicates that this routine defines the properties of a lighting model.

index Specifies the index into the table of stored definitions.

numpoints
Specifies the number of floating-point values contained within the *properties* array.

properties
Specifies an array that contains the definition to store at the *index* parameter. A definition is a grouping of properties and values ended by the symbolic constant LMNULL. There is a total of 13 defined symbolic constants (command tokens) that identify the properties of a definition. The valid symbolic constants for the *properties* parameter are shown in the following table.

Constant		
C	**FORTRAN**	**Description**
EMISSION	EMISSI	Assigns an emission color to a material. Following this symbolic constant, enter the red, green, and blue color component values for the desired emission color. Valid color component values range from 0.0 to 1.0 inclusive.

AMBIENT	AMBIEN	Can be a property of a material, a light, or a lighting model. Following this symbolic constant, enter the red, green, and blue color values for the desired ambient color. Valid color component values range from 0.0 to 1.0 inclusive. For each definition type, AMBIENT assigns the following: DEFMATERIAL - The ambient reflectance of the material. DEFLIGHT - The ambient light associated with the light source. DEFLMODEL - The ambient light present in the scene. The properties of a lighting model apply to an entire scene.
DIFFUSE	DIFFUS	Assigns the diffuse reflectance of a material. Following this symbolic constant, enter the red, green, and blue color component values for the desired diffuse reflectance color. Valid color component values range from 0.0 to 1.0 inclusive.
SPECULAR	SPECUL	Assigns the specular reflectance of a material. Following this symbolic constant, enter the red, green, and blue color component values for the desired specular reflectance color. Valid color component values range from 0.0 to 1.0 inclusive.
SHININESS	SHININ	Assigns the material specular scattering exponent of the material. Following this symbolic constant, enter a number between 0.0 and 128.0 inclusive. The higher the value, the smoother the surface appearance and the more focused the specular highlight. **Note:** The 3-D Color Graphics Processor rounds the SHININESS symbolic constant to the nearest whole integer. The POWER Gt4 and POWER Gt4x adapters accept fractional SHININESS values.
ALPHA	ALPHA	Assigns the transparency of the material. Following the ALPHA symbolic constant, enter a value between 0.0 and 1.0, inclusive. (Systems without alpha bitplanes cannot use this property.)
LCOLOR	LCOLOR	Assigns the color of a light source. Following this symbolic constant, enter the red, green, and blue color component values for the desired color of the light. Valid color component values range from 0.0 to 1.0 inclusive.

POSITION	POSITI	Assigns the position of a light source. Following this symbolic constant, enter the *x, y, z,* and *w* coordinates of the light source. If *w* is zero, the light source is infinitely distant and the *x, y,* and *z* values specify the direction of the light. Locating all light sources at infinity *w* = 0 improves performance.
SPOTDIRECTION	SPOTDI	Assigns the direction (axis) in which a spotlight source emits. Following this symbolic constant, enter the *x*, *y*, and *z* components of the direction vector. You can specify any values for *x*, *y*, and *z*; the system normalizes the direction before employing it in the lighting equation. The default direction points down the negative *z*-axis, direction (0.0, 0.0, -1.0). The direction is transformed by the current modeling/viewing matrix when the light is bound. The direction is ignored unless the light is a spotlight. **Note:** This token is not supported on the 3-D Color Graphics Processor.
LOCALVIEWER	LOCALV	Assigns the local viewer status for a lighting model. If you want the viewer (eye position) to be local, enter 1.0 after this constant, and the lighting calculations assume that the viewer is located at (0,0,0). If you do not want the viewer to be local, then enter 0.0 after this constant, and the lighting calculations assume that the viewer is at positive infinity along the *z* axis. There is a performance penalty when you request a local viewer.
ATTENUATION	ATTENU	Assigns the light attenuation factor for the lighting model (scene). Following this symbolic constant, enter two attenuation factor values to specify: k-off - Attenuation offset factor. k-rate - Attenuation rate factor.
COLORINDEXES	COLORI	Specifies the material properties used when lighting in color map mode. This property is ignored while the current frame buffer is in RGB mode, as are most other material properties when the current frame buffer 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 indexes.

Example

The example C language program **cylinder2.c** uses the **lmdef** subroutine to define the properties of two materials and two lights, and to define a lighting model.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

On the POWERstation 730 and POWERgraphics GTO, the specular reflection exponent (as set with the SHININESS symbolic constant of the **lmdef** subroutine *properties* parameter) is truncated to the nearest power of 2 before computing lighting. The only valid values for the specular exponent are 1, 2, 4, 8, 16, 32, 64, and 128.

The POWERstation 730 and POWERgraphics GTO do not support lighting in color index mode.

Materials with different material properties on each side are supported on the POWER Gt4 class of graphics adapters. Support for two-sided lighting is accomplished with the following token:

Type	DEFLMODEL
Property	TWOSIDE
Function	The TWOSIDE token followed by 0.0 disables two-sided lighting; followed by 1.0 enables two-sided lighting.

When two-sided lighting is disabled, the direction of the normal vector solely affects how a material is lit. When the normal vector points towards the viewer, there will typically be a positive contribution to the ambient, diffuse and specular terms of the lighting equation (based, of course, on the material properties, the light positions and colors) When the normal points away from the viewer, there will only be a contribution from the ambient term, and the surface will not appear shiny and reflective (This is because all terms in the lighting equation involving a dot product with the normal vector are clamped to zero, and are not allowed to go negative. If the normal points away from the user, such dot product terms will be clamped to zero, no mater what the light positions might be.)

When two-sided lighting is enabled, the manner in which lighting is performed depends on whether the polygon being lit is front-facing or back-facing. If the polygon is front-facing (the polygon vertices were specified in counter-clockwise order in screen coordinates), then lighting is performed as usual. If the polygon is backfacing, then the direction of the normal vector is reversed before it is used in the lighting equation (that is, it is multiplied by -1 before being used.). Thus, if two-sided lighting is enabled, and if the normal vectors were specified consistently with the handedness of the polygon, then both sides of the polygon will have diffuse and specular contributions,

If a material has been bound to the BACKMATERIAL token, then this material will be used when the back-side lighting is being computed. Otherwise, both front and back-facing polygons will be lit with the MATERIAL properties.

Two-sided lighting can only be enabled for, and affects only polygons and triangle strips. Points, lines and characters are always lit as if two-sided lighting were disabled.

The default lighting model is:
AMBIENT 0.2, 0.2, 0.2
ATTENUATION 1.0, 0.0
LOCALVIEWER 0.0
TWOSIDE 0.0

Note: To obtain the desired lighting effect with two-sided lighting, one must be careful that the direction of the normal vector is consistent with the handedness of the polygon. If the direction of the normal vector is accidentally reversed, then both sides of the polygon will appear dark.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current color in RGB mode with the **c** subroutine.

Setting the current color as a single packed 32-bit integer with the **cpack** subroutine.

Making a new material, light, or lighting model definition active with the **lmbind** subroutine.

Changing the target of the color commands with the **lmcolor** subroutine.

Specifying a normal vector with the **n3f** subroutine.

Updating the current normal vector with the **normal** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Graphics Library Overview, Creating Lighting Effects, Setting Pipeline Options, and Working in Color Map and RGB Modes.

---

loadmatrix Subroutine

Purpose

Loads a transformation matrix.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void loadmatrix(Matrix  matrix)
```

FORTRAN Syntax

```
SUBROUTINE LOADMA(matrix)
REAL matrix(4,4)
```

Description

The **loadmatrix** subroutine loads a 4x4 floating point matrix onto the transformation stack, replacing the current top matrix. If the system is in projection matrix mode (`mmode(MPROJ);`), the projection matrix is replaced.

Be sure to exit projection matrix mode before performing any drawing because drawing is not enabled while in this mode.

> **Note:** The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See Hardware Considerations in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.)

Parameter

matrix Specifies the matrix to be loaded onto the matrix stack.

Example

After changing the matrix mode to **MVIEWING** with the **mmode** subroutine, the example C language program **localatten.c** uses the **loadmatrix** subroutine to load the modelling/viewing matrix with the identity matrix.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current matrix mode with the **mmode** subroutine.

Getting a copy of the current transformation matrix with the **getmatrix** subroutine.

Premultiplying the current transformation matrix with the **multmatrix** subroutine.

Popping the transformation matrix stack with the **popmatrix** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Graphics Library Overview, Working with Coordinate Systems.

---

loadname Subroutine

Purpose

Loads a name onto the name stack.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void loadname(Int16  name)
```

FORTRAN Syntax

```
SUBROUTINE LOADNA(name)
INTEGER*4 name
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **loadname** subroutine replaces the top name in the name stack with a new 16-bit integer.

If a hit has occurred since the last time the name stack was touched, the system stores the contents of the name stack in a buffer. This enables the user to identify the part of an image that appears near the cursor.

The name stack is used only in picking or selecting mode.

Parameter

name Specifies the name to be loaded onto the name stack.

Example

The example C language program **pick1.c** calls the **loadname** subroutine to place a name on the top of the name stack.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Putting the system in selecting mode with the **gselect** subroutine.

Initializing the name stack with the **initnames** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Popping a name off the name stack with the **popname** subroutine.

Pushing a new name onto the name stack with the **pushname** subroutine.

Graphics Library Overview and Picking and Selecting Overview.

loadXfont Subroutine

Purpose
Loads an Enhanced X-Windows font into the font table.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void loadXfont(Int32  id_num, Char8 * name)
```

FORTRAN Syntax
```
SUBROUTINE LOADXF (id_num, name, length)
INTEGER *4 id_num
CHARACTER *(*) name (*)
INTEGER *4 length
```

Description
The **loadXfont** subroutine adds the named font to the list of defined fonts for this process. The *id_num* parameter identifies the font and may be used with the **font** subroutine to set the current font.

The font name is a null-terminated string that is a valid font name. The list of available fonts can be found with the **XListFonts** subroutine. The example program on loading X fonts shows an example of the use of the **XListFonts** subroutine.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters
id_num	Specifies the number to be assigned to an Enhanced X-Windows font name. This parameter is a 32-bit integer.
name	Specifies a null-terminated string identifying a valid AIXwindows font name.
length	In FORTRAN, specifies the length of the string in the *name* parameter. This parameter is a 32-bit integer.

Example
The example C language program **xfonts.c** uses the **loadXfont** subroutine to add a font to the fonts defined for a process.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Getting a list of available fonts with the **XListFonts** subroutine.

Getting a list of available fonts with information with the **XListFontsWithInfo** subroutine.

Graphics Library Overview, Creating Text Characters, Understanding the Hardware Used by GL, and Clearing, Resetting, and Initializing GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

logicop Subroutine

Purpose

Specifies the logical, arithmetic, or comparison operation for pixel updates.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void logicop(Int32  opcode)
```

FORTRAN Syntax

```
SUBROUTINE LOGICO(opcode)
INTEGER*4 opcode
```

Description

The **logicop** subroutine specifies the bitwise logical, arithmetic, or comparison operation for pixel updates. The operation is applied between the incoming (source) and existing (destination) values to generate the final pixel value. In color map mode, all writemask-enabled index bits (up to 12) are changed. In RGB mode, all enabled component bits (up to 32) are changed.

The **logicop** subroutine defaults to the value of LO_SRC, meaning that the incoming (source) value replaces the existing (destination) value.

The arithmetic and comparison operations treat each channel (R, G, and B) separately. That is, the result of an arithmetic or comparison operation in one channel does not affect the others; each channel is treated as a separate 8-bit entity, rather than a part of a 24-bit quantity. For example, if the *opcode* parameter is set to the LO_MAX value, if the current pixel has an RGB intensity of (119, 34, 230), and the incoming pixel has an intensity of (180, 180, 180), the final pixel value is (180, 180, 230).

The pixel color comparisons do not affect the way the z-buffer works. Z-buffer comparisons and color comparisons can be done simultaneously. The color of a pixel is updated only if the z-buffer comparison passed first; the color is then updated if the color comparison test passes. This function is more general than that provided by the **zsource** subroutine with a ZSRC_COLOR setting because it does *not* disable z-comparisons the way that the **zsource** subroutine does.

The arithmetic operations in each channel saturate, rather than wrap around. Because each color channel is 8 bits deep, color values saturate at 255 and 0. For example, if the *opcode* parameter is set to the

LO_SUM value, if the current pixel has an RGB intensity of (119, 34, 230), and the incoming pixel has an intensity of (180, 180, 180), then the final pixel value is (255, 214, 255).

It is not possible to do logical and blending operations simultaneously. When the *opcode* parameter is set to any value other than LO_SRC, the **blendfunction** subroutine values *sfactor* and *dfactor* are forced to their default values, BF_ONE and BF_ZERO, respectively. Likewise, calling the **blendfunction** subroutine with arguments other than BF_ONE and BF_ZERO forces the logical opcode to a value of LO_SRC.

Unlike the **blendfunction** subroutine, the **logicop** subroutine is valid in all drawing modes (NORMALDRAW, UNDERDRAW, OVERDRAW, PUPDRAW, CURSORDRAW) and in both color map and RGB modes. Like the **blendfunction** subroutine, it affects all drawing operations, including points, lines, polygons, and pixel area transfers. The **logicop** subroutine does *not* affect pixel block transfers (blits) into the z-buffer.

The **logicop** subroutine functions in systems without alpha bitplanes.

> **Note:** Use the **logicop** subroutine when drawing antialiased lines to improve the appearance of intersecting lines.

Parameter

opcode
Token indicating which of the following operations should be in effect:

C	FORTRAN	Operation
LO_ZERO	LOZERO	0
LO_AND	LOAND	src AND dst
LO_ANDR	LOANDR	src AND (NOT dst)
LO_SRC	LOSRC	src
LO_ANDI	LOANDI	(NOT src) AND dst
LO_DST	LODST	dst
LO_XOR	LOXOR	src XOR dst
LO_OR	LOOR	src OR dst
LO_NOR	LONOR	NOT (src OR dst)
LO_XNOR	LOXNOR	NOT (src XOR dst)
LO_NDST	LONDST	NOT dst
LO_ORR	LOORR	src OR (NOT dst)
LO_NSRC	LONSRC	NOT src
LO_ORI	LOORI	(NOT src) OR dst
LO_NAND	LONAND	NOT (src AND dst)
LO_ONE	LOONE	1
LO_MAX	LOMAX	D=Maximum (S, D)
LO_MIN	LOMIN	D=Minimum (S, D)
LO_SUM	LOSUM	D= D + S (saturated)
LO_DMS	LODMS	D= D - S (saturated)
LO_SMD	LOSMD	D= S - D (saturated)
LO_AVG	LOAVG	D=(S + D) / 2

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The numeric assignment of the 22 operation names were chosen to be identical to those defined by the AIXwindows system.

The LO_MAX, LO_MIN, LO_SUM, LO_DMS, LO_SMD, and LO_AVG values are not supported on the 3-D Color Graphics Processor, the Supergraphics Processor Subsystem, or the POWER GXT1000 adapter.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the alpha blending ratio with the **blendfunction** subroutine.

Specifying either depth or color as the source for *z* comparisons with the **zsource** subroutine.

Configuring the Frame Buffer.

Writemasks and Logical Operations and Working in Color Map and RGB Modes.

---

lookat Subroutine

Purpose

Defines a viewing transformation.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lookat
(Coord  viewx,Coord  viewy, Coord  viewz,
Coord  pointx, Coord  pointy, Coord  pointz,
Angle  twist)
```

FORTRAN Syntax

```
SUBROUTINE LOOKAT(viewx, viewy, viewz, pointx, pointy, pointz, twist)
REAL*4 viewx, viewy, viewz, pointx, pointy, pointz
INTEGER*4 twist
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **lookat** subroutine defines the viewpoint and a reference point on the line of sight in world coordinates. The viewpoint is at (*viewx*, *viewy*, *viewz*), and is the position from which you are looking. The reference point is at (*pointx, pointy*, *pointz*), and is the location on which the viewpoint is centered. If *pointx*, *pointy*, and *pointz* are 0, you are looking at the origin of the world coordinate system. The viewpoint and reference point define the line of sight. The *twist* parameter measures right-hand rotation about the line of sight.

Normally, the **lookat** subrotuine is used to set up the mapping from world coordinates to eye coordinates (equivalently, to define the location of the viewer's eye in world coordinates). If the **lookat** subroutine is the first transformation subroutine called after projection matrix is set up and the matrix stack is initialized, it sets up such a mapping.

The **lookat** subroutine can also be used as a modeling transformation. Whether it behaves as a viewing transformation or a modeling transformation depends on the order in which it is called with respect to the other transformation subroutines and with respect to the drawing subroutines.

Parameters

viewx	Specifies the *x* coordinate of the viewing point.
viewy	Specifies the *y* coordinate of the viewing point.
viewz	Specifies the *z* coordinate of the viewing point.
pointx	Specifies the *x* coordinate of the reference point.
pointy	Specifies the *y* coordinate of the reference point.
pointz	Specifies the *z* coordinate of the reference point.
twist	Specifies the angle of rotation.

Example

The example C language program **zbuffer1.c** uses the **lookat** subroutine to define the point of view.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining the viewer's position in polar coordinates with the **polarview** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

IRGBrange Subroutine

Purpose

Sets the range of colors used for depth-cueing.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lRGBrange
(Int16  rmin, Int16  gmin,
Int16  bmin, Int16  rmax,
Int16  gmax, Int16  bmax,
Int32  znear, Int32  zfar)
```

FORTRAN Syntax

```
SUBROUTINE LRGBRA(rmin, gmin, bmin,
          rmax, gmax, bmax, znear, zfar)
INTEGER*4 rmin, gmin, bmin, rmax,
         gmax, bmax, znear, zfar
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **lRGBrange** subroutine sets the range of colors used in depth-cueing in RGB mode. The range is mapped linearly into the RGB color range. Any *z* values less than the value of the *znear* parameter are mapped to *rmax*, *gmax*, and *bmax* parameters. Any *z* values greater than the value of the *zfar* parameter are mapped to *rmin*, *gmin*, and *bmin* parameters.

The valid range of values for the *znear* and *zfar* parameters can be found by using the **getgdesc** subroutine to query **GD_ZMIN** and **GD_ZMAX** attributes. (See the **getgdesc** subroutine and ″Tokens for the getgdesc Subroutine″ .)

Note: The value for the *znear* parameter *must* be less than the value for the *zfar* parameter.

Parameters

rmin	Specifies the minimum value to be stored in the red bitplanes.
gmin	Specifies the minimum value to be stored in the green bitplanes.
bmin	Specifies the minimum value to be stored in the blue bitplanes.
rmax	Specifies the maximum value to be stored in the red bitplanes.
gmax	Specifies the maximum value to be stored in the green bitplanes.
bmax	Specifies the maximum value to be stored in the blue bitplanes.
znear	Specifies the minimum z value to be used as the criterion for linear mapping.
zfar	Specifies the maximum z value to be used as the criterion for linear mapping.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning depth-cueing on and off with the **depthcue** subroutine.

Getting information about the currently installed graphics hardware with the **getgdesc** subroutine.

Setting the viewport depth range with the **lsetdepth** subroutine.

Setting the range of color indexes used for depth-cueing with the **lshaderange** subroutine.

Graphics Library Overview, Performing Depth-Cueing, and Working in Color Map and RGB Modes.

---

lsetdepth Subroutine

Purpose

Sets up a 3-D viewport.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lsetdepth(Int32  near, Int32  far)
```

FORTRAN Syntax

```
SUBROUTINE LSETDE(near, far)
INTEGER*4 near, far
```

Description

The **lsetdepth** subroutine takes the mapping furnished by the **viewport** subroutine and completes this mapping for homogeneous coordinates. The **viewport** subroutine specifies the mapping of the left, right, bottom, and top clipping planes into screen coordinates. The **lsetdepth** subroutine then specifies the mapping of the near and far clipping planes into values stored in the z-buffer. This subroutine is used in z-buffering and depth-cueing.

Acceptable mappings include all those where the values of the *near* and *far* parameters are within the supported range, including mappings where *near* > *far*.

1. Error accumulation in the iteration of the *z* coordinate can cause wrapping when the full-depth range supported by the graphics hardware is used. (An iteration wraps when it accidentally converts a large positive value into a negative value or vice versa.) The effects of wrapping, although typically not observed, can be eliminated by reducing the depth range by a small percentage.
2. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″;.)

The valid range of values for the **lsetdepth** subroutine can be found by querying with the **getgdesc** subroutine with the **GD_ZMIN** and **GD_ZMAX** tokens.

> **Note:** The POWER Gt4x adapter supports a full 24-bit z range, from 0x0 to 0xFFFFFF. It does not support negative-signed integer values.

Parameters

near Specifies the screen coordinate of the near clipping plane.
far Specifies the screen coordinate of the far clipping plane.

Example

The example C language program **depthcue.c** uses the **lsetdepth** subroutine to set the range of z-axis values to store in the bitplanes.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 supports a repeat factor from 1 to 255. Query the maximum supported repeat factor with getgdesc(GD_MAX_LSREPEAT).

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning depth-cueing on and off with the **depthcue** subroutine.

Get information about the installed graphics hardware with the **getgdesc** subroutine.

Setting the viewport depth range with the **lsetdepth** subroutine.

Graphics Library Overview, Configuring the Frame Buffer, Performing Depth-Cueing, Using Viewports and Screenmasks, and Working with Coordinate Systems.

---

lshaderange Subroutine

Purpose

Sets the range of color indexes used in depth-cueing.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void lshaderange
(Colorindex  lowindex, Colorindex  highindex,
Int32  znear, Int32  zfar)
```

FORTRAN Syntax

```
SUBROUTINE LSHADE(lowindex, highindex, znear, zfar)
INTEGER*4 lowindex, highindex, znear, zfar
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **lshaderange** subroutine sets the range of color indexes used for depth-cueing. The range is mapped linearly into the color index range. Any *z* values less than the value of the *znear* parameter map to the *highindex* parameter; *z* values greater than the *zfar* parameter map to the *lowindex* parameter.

The valid range of values for the *znear* and *zfar* parameters can be found by using the **getgdesc** subroutine to query **GD_ZMIN** and **GD_ZMAX** attributes. The default is **lshaderange(0, 7,** *zmin*, *zmax***)**. (See also ″Tokens for the getgdesc Subroutine″ .)

Note: The value for the *znear* parameter must be less than the value for the *zfar* parameter.

Parameters

lowindex	Specifies the low-intensity color map index.
highindex	Specifies the high-intensity color map index.
znear	Specifies the low *z* value.
zfar	Specifies the high *z* value.

Example

The example C language program **depthcue.c** uses the **lshaderange** subroutine to map the z-axis values to a spread of colors in the color map.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning depth-cueing on and off with the **depthcue** subroutine.

Getting information about the currently installed graphics hardware with the **getgdesc** subroutine .

Setting the range of RGB colors used for depth-cueing with the **lRGBrange** subroutine.

Setting the viewport depth range with the **lsetdepth** subroutine.

Graphics Library Overview, Performing Depth-Cueing, and Working in Color Map and RGB Modes.

---

lsrepeat Subroutine

Purpose
Sets the repeat factor for the current linestyle.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void lsrepeat(Int32  factor)
```

FORTRAN Syntax
```
SUBROUTINE LSREPE(factor)
INTEGER*4 factor
```

Description
The **lsrepeat** subroutine sets the repeat factor for the current linestyle.

This subroutine is used to create linestyles that are longer than 16 bits by multiplying each bit in the pattern by the value of the *factor* parameter.

Each bit in the pattern is multiplied successively. For example, if the line pattern is `0000000111111111` and the *factor* parameter equals `3`, the resulting linestyle is `27` bits on followed by `21` bits off. Line patterns start from the least significant bit.

When a line is drawn, pixels are turned on if there is a 1 (one) in the corresponding position of the linestyle mask and turned off if there is a 0 (zero) in the corresponding position. The valid range of the repeat factor is from 1 to 255.

> **Note:** The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See Hardware Considerations in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.)

Parameter
factor Multiplier of the linestyle pattern.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

On the POWER Gt4 and POWER Gt4x adapters, the valid range for the repeat factor is from 1 to 64.

On the POWERstation 730 and POWERgraphics GTO, linestyle repeat factors must be a multiple of 4. The valid range is from 4 to 252 in multiples of 4 with a lowest possible repeat factor of 4.

On the Supergraphics Processor Subsystem, the valid range for the repeat factor is 4 to 252 in multiples of 4. Any other value specified is rounded to the nearest multiple of 4.

On the POWER GXT1000, the valid range for the repeat factor is 1 to 255.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a linestyle with the **deflinestyle** subroutine.

Returning the linestyle repeat count with the **getlsrepeat** subroutine.

Selecting a linestyle with the **setlinestyle** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Drawing NURBS Curves and Surfaces, Drawing Wire Frame Curves and Surface Patches, Setting Drawing Attributes, Understanding the Hardware Used by GL, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

makeobj Subroutine

Purpose

Creates an object (display list).

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void makeobj(Int32  object)
```

FORTRAN Syntax

```
SUBROUTINE MAKEOB(object)
INTEGER*4 object
```

Description

The **makeobj** subroutine creates and names a new display list (object). If the *object* parameter is the number of an existing object, the contents of that object are deleted.

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

Notes:

1. This editing subroutine itself cannot be added to a display list.
2. The **makeobj** subroutine cannot be called within an object; that is, an existing object cannot be used to create a new object.

Parameter

object Specifies the numeric identifier for the object being defined.

Example

The example C language program **depthcue.c** uses the **makeobj** subroutine to specify the beginning of a display list defining a graphical object.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing an instance of an object with the **callobj** subroutine.

Specifying the amount of memory allocated for an object with the **chunksize** subroutine.

Closing an object with the **closeobj** subroutine.

Opening an object for editing with the **editobj** subroutine.

Returning a unique integer for use as an object identifier with the **genobj** subroutine.

Establishing the uniqueness of an object number with the **isobj** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

maketag Subroutine

Purpose

Inserts a marker tag into a display list.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void maketag(Int32  tag)
```

FORTRAN Syntax

```
SUBROUTINE MAKETA(tag)
INTEGER*4 tag
```

Description

The **maketag** subroutine inserts a marker, or tag, at the current editing location of the currently open display list. The current editing position is usually at the end of the display list, if the display list was recently opened with the **makeobj** subroutine or **editobj** subroutine.

The current position can be maneuvered with the **objdelete**, **objinsert**, and **objreplace** subroutines. Tags can be used to help in the maneuvering through and editing of display lists.

There are two predefined tags, STARTTAG and ENDTAG, which mark the beginning and end of the display list. Tags should be unique within a single object; however, there is no restriction regarding uniqueness across different objects. Unique tags can be generated with the **gentag** subroutine.

> **Note:** This editing subroutine itself cannot be added to a display list.

Parameter

tag Specifies a 31-bit numeric identifier assigned to a routine in the display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Deleting tags from a display list with the **deltag** subroutine.

Returning a unique integer for use as a tag number with the **gentag** subroutine.

Establishing the uniqueness of a tag number with the **istag** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

mapcolor Subroutine

Purpose

Changes a color map entry to a specified RGB value.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void mapcolor
(Colorindex  index,
Int16  red, Int16  green, Int16  blue)
```

FORTRAN Syntax

```
SUBROUTINE MAPCOL (index, red, green, blue)
INTEGER*4 index, red, green, blue
```

Note: For FORTRAN users: this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **mapcolor** subroutine changes a single color map entry for the current drawing mode to a specified RGB value. The *index* parameter loads the color map entry that corresponds with the specified RGB intensities. The valid range for the *index* parameter depends on the drawing mode:

NORMALDRAW	0 to 4095, or 0 to 255 if the system has only eight color bitplanes or is in multimap mode.
OVERDRAW	1 to 3 if the **overlay(2)** subroutine has been called, or 1 to 15 if the **overlay(4)** subroutine has been called.
UNDERDRAW	0 to 3 if the **underlay(2)** subroutine has been called, or 0 to 15 if the **underlay(4)** subroutine has been called.
PUPDRAW	1 to 3.
CURSORDRAW	1 to 3.

In multimap mode, the **mapcolor** subroutine updates only the small color map currently selected by the **setmap** subroutine. The system ignores invalid indexes.

On most systems, this subroutine does not set the current drawing color, and so should not be used for this purpose. To set the current drawing color, use the **c**, **color**, **cpack**, or **RGBcolor** subroutine.

Instead of the **mapcolor** subroutine, it is suggested that you use the **mapcolors** subroutine for new development, because of its significantly improved performance.

Note: This subroutine cannot be used to add to a display list.

Parameters

index	Specifies the index into the color map.
red	Specifies the intensity of red associated with the index.
green	Specifies the intensity of green associated with the index.
blue	Specifies the intensity of blue associated with the index.

Example
The example C language program **colored.c** uses the **mapcolor** subroutine to edit the color map.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 supports a 12-bit colorindex frame buffer.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Choosing a set of bitplanes for drawing with the **drawmode** subroutine.

Defining a color map ramp for gamma correction with the **gammaramp** subroutine.

Getting a copy of the RGB values for a color map entry with the **getmcolor** subroutine.

Organizing the color map as one large map with the **onemap** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Setting the number of bitplanes used for overlay colors with the **overlay** subroutine.

Selecting one of 16 small color maps with the **setmap** subroutine.

Setting the number of bitplanes used for underlay colors with the **underlay** subroutine.

Working in Color Map and RGB Modes.

---

mapcolors Subroutine

Purpose
Loads a range of the color map.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
void mapcolors(Int16 *start_idx*, **Int16** *end_idx*, **Int16** *r* **[ ], Int16** *g* **[ ], Int16** *b* **[ ])**

FORTRAN Syntax

```
SUBROUTINE MAPCOLORS (start_idx, end_idx, r, g, b)
INTEGER*4 start_idx, end_idx
INTEGER*2 r(1), g(1), b(1)
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **mapcolors** subroutine loads a range of color map table entries. The range that is loaded begins with *start_idx* and ends with *end_idx*, inclusive. The length of the array must be equal to *end_idx* - *start_idx* + 1. For instance to load only one color map entry, set *start_idx* and *end_idx* to the same number. To load two entries, set *end_idx* to *start_idx* + 1.

The **mapcolors** subroutine is functionally equivalent to the C code shown in the following fragment, although it will execute considerably faster.

```
{
  int i;
  for (i=0; i< (end_idx - start_idx + 1); i++)
     mapcolor (start_idx+i, r[i],g[i],b[i]);
}
```

The **mapcolors** subroutine can be used to load the underlay, overlay, cursor, popup, or main frame buffer color map. Which map is loaded depends on the current drawing mode (as set by the **drawmode** subroutine).

NORMALDRAW	0 to 4095, or 0 to 255 if the system has only eight color bitplanes or is in multimap mode.
OVERDRAW	1 to 3 if the **overlay(2)** subroutine has been called, or 1 to 15 if the **overlay(4)** subroutine has been called.
UNDERDRAW	0 to 3 if the **underlay(2)** subroutine has been called, or 0 to 15 if the **underlay(4)** subroutine has been called.
PUPDRAW	1 to 3.
CURSORDRAW	1 to 3.

In multimap mode, the **mapcolors** subroutine updates only the small color map currently selected by the **setmap** subroutine. The system ignores invalid indexes.

Note: This subroutine cannot be used to add to a display list.

Parameters

start_idx	Specifies the starting index in the color map to be loaded.
end_idx	Specifies the ending index in the color map to be loaded.
r	Specifies an array containing the intensity of the red component.
g	Specifies an array containing the intensity of the green component.
b	Specifies an array containing the intensity of the blue component.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWERgraphics GXT1000 supports a 12-bit colorindex frame buffer.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Choosing a set of bitplanes for drawing with the **drawmode** subroutine.

Defining a color map ramp for gamma correction with the **gammaramp** subroutine.

Getting a range of color map RGB entries with the **getmcolors** subroutine.

Organizing the color map as one large map with the **onemap** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Setting the number of bitplanes used for overlay colors with the **overlay** subroutine.

Selecting one of 16 small color maps with the **setmap** subroutine.

Setting the number of bitplanes used for underlay colors with the **underlay** subroutine.

Working in Color Map and RGB Modes.

---

mapw Subroutine

Purpose

Computes the inverse mapping from screen coordinates to modeling coordinates.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void mapw
(Int32  viewobj,
Screencoord  screenx, Screencoord  screeny,
Coord * modelx1, Coord * modely1, Coord * modelz1,
Coord * modelx2, Coord * modely2, Coord *  modelz2)
```

FORTRAN Syntax

SUBROUTINE MAPW(*viewobj*, *screenx*, *screeny*, *modelx1*, *modely1*, *modelz1*, *modelx2*, *modely2*, *modelz2***)**

INTEGER*4 *viewobj*, *screenx*, *screeny*

REAL *modelx1*, *modely1*, *modelz1*, *modelx2*, *modely2*, *modelz2*

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

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

The system returns a modeling coordinate line, which is computed from the (*screenx*, *screeny*) parameters and the *viewobj* parameter as two points and stored in the locations addressed by the *modelx1*, *modely1*, *modelz1* parameters and the *modelx2*, *modely2*, *modelz2* parameters.

Note: This subroutine cannot be used to add to a display list.

Parameters

viewobj	Specifies a viewing object containing the transformations that map the current displayed objects to the screen.
screenx	Specifies the *x* coordinate of the screen point to be mapped.
screeny	Specifies the *y* coordinate of the screen point to be mapped.
modelx1	Specifies the *x* model coordinate of one endpoint of a line.
modely1	Specifies the *y* model coordinate of one endpoint of a line.
modelz1	Specifies the *z* model coordinate of one endpoint of a line.
modelx2	Specifies the *x* model coordinate of the remaining endpoint of a line.
modely2	Specifies the *y* model coordinate of the remaining endpoint of a line.
modelz2	Specifies the *z* model coordinate of the remaining endpoint of a line.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Creating a new object in the display list with the **makeobj** subroutine.

Computing the inverse mapping from screen coordinates to 2-D modeling coordinates with the **mapw2** subroutine.

Graphics Library Overview, Picking and Selecting Overview, and Working with Coordinate Systems.

---

mapw2 Subroutine

Purpose

Computes the inverse mapping from screen coordinates to 2-D modeling coordinates.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void mapw2
(Int32  viewobj,
Screencoord  screenx, Screencoord  screeny,
Coord * modelx, Coord * modely)
```

FORTRAN Syntax

```
SUBROUTINE MAPW2(viewobj, screenx, screeny, modelx, modely)
INTEGER*4 viewobj, screenx, screeny
REAL modelx, modely
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **mapw2** subroutine maps a point on the screen into 2-D modeling coordinates. This subroutine is the 2-D version of the **mapw** subroutine.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

viewobj	Specifies a primitive containing the viewport, projection, viewing, and modeling transformations that define modeling space.
screenx	Specifies a point in screen coordinates.
screeny	Specifies a point in screen coordinates.
modelx	Specifies the corresponding *x* model coordinate.
modely	Specifies the corresponding *y* model coordinate.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Creating a new object in the display list with the **makeobj** subroutine.

Computing the inverse mapping from screen coordinates to 3-D modeling coordinates with the **mapw** subroutine.

Graphics Library Overview, Picking and Selecting Overview, and Working with Coordinate Systems.

maxsize Subroutine

Purpose

Specifies the maximum size of a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void maxsize(Int32  x, Int32  y)
```

FORTRAN Syntax

```
SUBROUTINE MAXSIZ(x, y)
INTEGER*4 x, y
```

Description

The **maxsize** subroutine specifies the maximum size (in pixels) of a window. It is called at the beginning of a graphics program, but only takes effect when the **winopen** subroutine is called.

The **maxsize** subroutine can also be called in conjunction with the **winconstraints** subroutine to modify the enforced maximum size after the window is created. The default maximum size is 1280 pixels wide and 1024 pixels high. The window can be reshaped, but cannot become larger than the specified maximum size.

With the **maxsize** subroutine, the programmer can prevent the user from resizing a window to a size larger than the specified size.

Note: This subroutine cannot be used to add to a display list.

Parameters

x Specifies the maximum width in pixels of a window.
y Specifies the maximum height in pixels of a window.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Obtaining the size of the window with the **getsize** subroutine.

Specifying the size of a window icon with the **iconsize** subroutine.

Specifying the minimum size of a window with the **minsize** subroutine.

Constraining the size of a window with the **prefsize** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

minsize Subroutine

Purpose

Specifies the minimum size of a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void minsize(Int32  x, Int32  y)
```

FORTRAN Syntax

```
SUBROUTINE MINSIZ(x, y)
INTEGER*4 x, y
```

Description

The **minsize** subroutine specifies the minimum size (in pixels) of a window. It is called at the beginning of a graphics program, but only takes effect when the **winopen** subroutine is called.

The **minsize** subroutine can also be called in conjunction with the **winconstraints** subroutine to modify the enforced minimum size after the window is created. The default minimum size is 40 pixels wide and 30 pixels high. The window can be reshaped, but cannot become smaller than the specified minimum size.

With the **minsize** subroutine, the programmer can prevent the user from resizing a window to a size smaller than the specified size.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

x Specifies the minimum width in pixels of a window. The lowest legal value for this parameter is 21.
y Specifies the minimum height in pixels of a window. The lowest legal value for this parameter is 21.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Obtaining the size of the window with the **getsize** subroutine.

Specifying the size of a window icon with the **iconsize** subroutine.

Specifying the maximum size of a window with the **maxsize** subroutine.

Constraining the size of a window with the **prefsize** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

mmode Subroutine

Purpose

Sets the current matrix mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void mmode(Int16  mode)
```

FORTRAN Syntax

```
SUBROUTINE MMODE(mode)
INTEGER*4 mode
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when started from a FORTRAN program, although it accepts short integers when started from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **mmode** subroutine sets the current matrix mode.

The system is in single matrix mode after a call to the **winopen**, **ginit**, or **gbegin** subroutines. This mode is sufficient except for lighting calculations; when performing those, use viewing matrix mode.

Because the projection matrix is not stored on the matrix stack, the projection matrix is not normally accessible. However, if you want to define your own projection matrix, put the system into projection matrix mode. You can then use the standard matrix manipulation commands to alter the projection matrix. When you are in projection matrix mode, do not use the **pushmatrix** and **popmatrix** subroutines.

When you have finished modifying the projection matrix, return to viewing matrix mode and load a 4 by 4 identity matrix onto the matrix stack. (The standard transformation subroutines, **rotate**, **rot**, **translate**, **scale**, **lookat**, and **polarview**, all use matrix multiplication. Loading the identity matrix onto the matrix stack is a safe way to make sure that there is a matrix by which to multiply.)

Parameter

mode Specifies the current matrix mode to be set.

There are three possible values for this function:

C	FORTRAN	Mode
MSINGLE	MSINGL	Single matrix
MPROJECTION	MPROJE	Projection matrix
MVIEWING	MVIEWI	Viewing matrix

Example

The example C language program **cylinder1.c** uses the **mmode** subroutine, with MVIEWING as the value of the *mode* parameter, to specify that matrix operations manipulate only the modeling and viewing matrix (not the projection matrix).

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Making a new material, light, or lighting model definition active with the **lmbind** subroutine.

Defining a new material, light, or lighting model with the **lmdef** subroutine.

Graphics Library Overview, Creating Lighting Effects, Setting Pipeline Options, and Working in Color Map and RGB Modes.

move Subroutine

Purpose

Moves the current graphics position to a specified point.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

void move2
(Coord  x, Coord  y)

void move2i
(Icoord  x, Icoord  y)

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

FORTRAN Syntax

```
SUBROUTINE MOVE(x, y, z)
REAL x, y, z
SUBROUTINE MOVEI(x, y, z)
INTEGER*4 x, y, z
SUBROUTINE MOVES(x, y, z)
INTEGER*2 x, y, z
SUBROUTINE MOVE2(x, y)
REAL x, y
SUBROUTINE MOVE2I(x, y)
INTEGER*4 x, y
SUBROUTINE MOVE2S(x, y)
INTEGER*2 x, y
```

Note: For FORTRAN users, the INTEGER*2 versions of this subroutine, **MOVES** and **MOVE2S**, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking one of the short versions of this subroutine with an integer constant will result in unexpected behavior.

Description

The **move** subroutine changes (without drawing) the current graphics position to the point specified by the *x, y*, and *z* parameters. The graphics position is the point from which the next drawing routine starts drawing.

The value of **move2(***x***,** *y***)** is equivalent to **move(***x***,** *y***,** 0.0**)**.

The six different forms for the **move** subroutine are as follows:

	2-D	**3-D**
Int16	move2s	moves
Int32	move2i	movei
float	move2	move

The syntax for each of the subroutine forms is the same except for the parameter types. They differ only in that **move** expects real coordinates, **movei** expects integer coordinates, and **moves** expects short integer coordinates. In addition, the **move2*** routines assume a 2-D point instead of a 3-D point.

Note: This subroutine cannot be used to add to a display list.

Parameters

x Specifies the new *x* coordinate for the current graphics position.
y Specifies the new *y* coordinate for the current graphics position.
z Specifies the new *z* coordinate for the current graphics position.

Example

The example C language program **depthcue.c** uses the **movei** subroutine when defining a graphical object to move the graphics position without drawing any lines.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a line with the **draw** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Drawing a point with the **pnt** subroutine.

Drawing a relative line with the **rdr** subroutine.

Moving the current graphics position to a point relative to the current point with the **rmv** subroutine.

multimap Subroutine

Purpose
Organizes the color map as 16 small maps.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void multimap( )
```

FORTRAN Syntax
```
SUBROUTINE MULTIM
```

Description
The **multimap** subroutine reorganizes the color map as 16 small independent maps. Only one of these small maps can be active, or current, at a time, and only the least significant bits in the frame buffer are passed through the current map.

On the 24-bit High-Performance 3-D Color Graphics Processor, these small maps are each 8-bit color maps having 256 entries. On the 8-bit High-Performance 3-D Color Graphics Processor, the maps are each 4-bit color maps having 16 entries.

After the **multimap** subroutine is called, the reorganization into multiple independent maps does not take effect until the **gconfig** subroutine is called.

Notes:

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Understanding the Adapter″.)
2. This subroutine cannot be used to add to a display list.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER GXT1000 adapter does not support the **multimap** subroutine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Configuring the system with the **gconfig** subroutine.

Returning the organization of the current color map with the **getcmmode** subroutine.

Returning the number of the current color map with the **getmap** subroutine.

Organizing the color map as one large map with the **onemap** subroutine.

Selecting one of 16 small color maps with the **setmap** subroutine.

Working in Color Map and RGB Modes.

---

multmatrix Subroutine

Purpose

Premultiplies the current transformation matrix.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void multmatrix(Matrix  matrix)
```

FORTRAN Syntax

```
SUBROUTINE MULTMA(matrix)
REAL matrix(4,4)
```

Description

The **multmatrix** subroutine 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*. If the system is in projection matrix mode (`mmode(MPROJ);`), the projection matrix is premultiplied.

Be sure to exit projection matrix mode before performing any drawing because drawing is not enabled while in this mode.

> **Note:** The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)

Parameter

matrix Specifies the matrix that is to multiply the current top matrix of the matrix stack.

Example

The example C language program **curve3.c** calls the **multmatrix** subroutine with the control point matrix, the Bezier basis matrix, and the precision matrix to set up the transformation matrix to draw a Bezier curve.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current matrix mode with the **mmode** subroutine.

Getting a copy of the current transformation matrix with the **getmatrix** subroutine.

Loading a transformation matrix with the **loadmatrix** subroutine.

Popping the transformation matrix stack with the **popmatrix** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

n3f Subroutine

Purpose

Specifies a normal vector for lighting calculations.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void n3f(Float32  vector [3] )
```

FORTRAN Syntax

```
SUBROUTINE N3F(vector)
REAL vector (3)
```

Description

The **n3f** subroutine specifies the normal vector to be used for lighting calculations.

The passed parameter (*vector*) becomes the current normal and is used when the lighting algorithm is rerun for all subsequent vertices. It is not necessary to respecify a normal if it is unchanged. For example, a single call to the **n3f** subroutine is sufficient for a flat-shaded polygon. However, the **n3f** subroutine must be called before the first vertex subroutine.

Lighting calculations assume that the specified normal is of unit length, as shown in the following equation:

$x^2 + y^2 + z^2 = 1.0$

If the normal does not equal 1.0, or if no lighting model is active, the results are unpredictable.

When called with unequal arguments, the **scale** subroutine or any other nonorthonomial transformation causes a matrix skew that is corrected by renormalizing every normal. Lighting performance is reduced in this event.

The **normal** and **n3f** subroutines are the same in all respects.

Parameter

vector Specifies the address of an array containing three floating point numbers. These numbers are used to set the value for the current vertex normal.

Vector components are:

Vector Component	C Index	FORTRAN Index
Nx	0	1
Ny	1	2
Nz	2	3

Example

The example C language program **cylinder2.c** calls the **n3f** subroutine between the **bgnpolygon** subroutine and the **endpolygon** subroutine to specify the normal vectors of a polygon.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Binding a new material, light, or lighting model definition with the **lmbind** subroutine.

Defining a new material, light, or lighting model with the **lmdef** subroutine.

Specifying a current normal vector for lighting calculations with the **normal** subroutine.

Graphics Library Overview, Creating Lighting Effects, Drawing with Begin-End Style Subroutines, and Understanding the Hardware Used by GL.

---

newpup Subroutine

Purpose

Allocates and initializes a structure for a new menu.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 newpup( )
```

FORTRAN Syntax

INTEGER*4 FUNCTION NEWPUP

Description

The **newpup** subroutine allocates and initializes a structure for a new menu. The return value is a positive menu identifier.

Use this subroutine with the **addtopup** subroutine to create pop-up menus.

Note: This subroutine cannot be used to add to a display list.

Return Value

An identifier for a new menu.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Adding an item to an existing pop-up menu with the **addtopup** subroutine.

Defining a pop-up menu with the **defpup** subroutine.

Displaying a pop-up menu with the **dopup** subroutine.

Deallocating a pop-up menu and its data structures with the **freepup** subroutine.

Enabling or disabling a given pop-up entry with the **setpup** subroutine.

Graphics Library Overview and Creating and Managing Pop-Up Menus.

---

newtag Subroutine

Purpose

Inserts a tag at an offset from an existing tag.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void newtag(Int32  newt, Int32  oldtag, Int32  offset)
```

FORTRAN Syntax
```
SUBROUTINE NEWTAG(newt, oldtag, offset)
INTEGER*4 newtag, oldtag, offset
```

Description
The **newtag** subroutine inserts a tag at a specified offset from the position marked by the *oldtag* parameter.

Note: This editing subroutine itself cannot be added to a display list.

Parameters

newt	Specifies a numeric identifier.
oldtag	Specifies a pre-existing tag to be used as a reference point for inserting the new tag.
offset	Specifies the number of lines beyond the position of the old tag at which to place the new tag.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Marking a location in a display list with the **maketag** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

noborder Subroutine

Purpose
Specifies a window without any borders.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void noborder( )
```

FORTRAN Syntax

```
SUBROUTINE NOBORD
```

Description

The **noborder** subroutine specifies a window that has no borders around its drawable area. Call this subroutine before opening the window.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating and Managing Windows.

---

noise Subroutine

Purpose

Filters valuator motion.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void noise(Device  valuator, Int16  delta)
```

FORTRAN Syntax

```
SUBROUTINE NOISE(valuator, delta)
INTEGER*4 valuator, delta
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **noise** subroutine provides squelch for noisy valuators. It prevents a valuator from reporting small fluctuations in movement that are not meaningful. For example, **noise(***valuator***,5)** means that the specified valuator must move at least 5 units before it makes a new queue entry.

A valuator must be queued before the **noise** subroutine is called.

The default noise value for all valuators other than the timer devices is 1. The default noise value for the timer devices (TIMERx) is 10,000.

Note: This subroutine cannot be used to add to a display list.

Parameters

valuator	Specifies a single-value input device.
delta	Specifies the number of units of change required before the valuator can make a new queue entry.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Assigning an initial value to a valuator with the **setvaluator** subroutine.

Graphics Library Overview, Controlling Queues and Devices and Using the Keyboard.

---

noport Subroutine

Purpose

Specifies that a program does not require a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void noport( )
```

FORTRAN Syntax

```
SUBROUTINE NOPORT
```

Description

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

The **noport** subroutine is ignored without a call to the **winopen** subroutine.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

normal Subroutine

Purpose

Sets the current normal vector.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void normal(Coord  narray[3])
```

FORTRAN Syntax

```
SUBROUTINE NORMAL(narray)
REAL narray(3)
```

Description

The **normal** subroutine sets the current normal vector. The *x, y,* and *z* modeling coordinates stored in the parameter *narray* are transformed into eye coordinates using the inverse transpose of the current viewing matrix. These numbers are used to set the value for the current vertex normal.

The passed parameter (*narray*) becomes the current normal and is used when the lighting algorithm is rerun for all subsequent vertices. It is not necessary to respecify a normal if it is unchanged. For example, a single call to the **n3f** subroutine is sufficient for a flat-shaded polygon. However, the **n3f** subroutine must be called before the first vertex subroutine.

Lighting calculations assume that the specified normal is of unit length, as shown in the following equation:

$x^2 + y^2 + z^2 = 1.0$

If the normal does not equal 1.0, or if no lighting model is active, the results are unpredictable.

The **normal** and **n3f** subroutines are the same in all respects.

Parameter

narray Specifies the address of an array containing three floating point numbers representing the *x, y,* and *z* model coordinates. Corresponding indexes for these coordinates are:

Model Coordinate	C Index	FORTRAN Index
x	0	1
y	1	2
z	2	3

Implementation Specifics

This subroutine is part GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Binding a new material, light, or lighting model definition with the **lmbind** subroutine.

Defining a new material, light, or lighting model with the **lmdef** subroutine.

Specifying a normal for lighting calculations with the **n3f** subroutine.

Graphics Library Overview, Creating Lighting Effects, Drawing with Begin-End Style Subroutines, and Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

nurbscurve Subroutine

Purpose
Controls the shape of a NURBS trimming curve.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void nurbscurve
(Int32  knot_count,
Float64  knot_list[ ],
Int32  stride,
Float64 * ctlarray,
Int32  order,
Int32  type)
```

FORTRAN Syntax
```
SUBROUTINE NURBSC(knot_count, knot_list,
        stride, ctlarray, order, type)
INTEGER*4 knot_count, stride, order, type
REAL*8 knot_list(knot_count), ctlarray(*)
```

Description
The **nurbscurve** subroutine describes a Non-Uniform Rational B-Spline (NURBS) curve. Use NURBS curves within trimming loop definitions. To mark the beginning and end of a trimming loop definition, use the **bgntrim** and **endtrim** subroutines.

Use trimming loop definitions within NURBS surface definitions, as defined by the **bgnsurface** subroutine. Describe a trimming loop by using a series of NURBS curves, piecewise linear curves (as defined by the **pwlcurve** subroutine), or both.

The system displays the region of the NURBS surface that is to the left of the trimming curves as the parameter increases. Thus, for a counterclockwise-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 *stride* parameter is used when the control points are part of an array of larger structure elements. The **nurbscurve** subroutine searches for the *n*th control point pair or triple beginning at byte address:

**(ctlarray + n x stride)*

Parameters
knot_count
: Specifies the number of knots.

knot_list
: Specifies an array of *knot_count* nondecreasing knot values.

stride Specifies the offset (in bytes) between successive curve control points.

ctlarray
Specifies an array containing control points for the NURBS curve. The coordinates must appear as either *(x, y)* pairs or as *(x, y, w)* triples. The offset between successive control points is given by *stride*.

order Specifies the order of the NURBS curve. The order is one more than the degree, hence, a cubic curve has an order of 4.

type Specifies a value indicating the control point type. Current options are as follows:

C	FORTRAN	Description
N_ST	NST	Indicates 2-dimensional control points.
N_STW	NSTW	Indicates 3-dimensional (rational) control points.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Marking the beginning and end of a NURBS surface definition with the **bgnsurface** and **endsurface** subroutines.

Marking the beginning and end of a NURBS surface trimming loop with the **bgntrim** and **endtrim** subroutines.

Returning the current value of a trimmed NURBS surfaces display property with the **getnurbsproperty** subroutine.

Controlling the shape of a untrimmed NURBS surface with the **nurbssurface** subroutine.

Describing a piecewise linear trimming curve for NURBS surfaces with the **pwlcurve** subroutine.

Setting a property for the display of trimmed NURBS with the **setnurbsproperty** subroutine.

Graphics Library Overview and Drawing NURBS Curves and Surfaces.

---

nurbssurface Subroutine

Purpose

Controls the shape of a NURBS surface

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void nurbssurface
(Int32  s_knot_count,
Float64  s_knots[],
Int32  t_knot_count,
Float64  t_knots[],
Int32  s_stride,
Int32  t_stride,
Float64 * ctlarray,
Int32  s_order,
Int32  t_order,
Int32  type)
```

FORTRAN Syntax

```
SUBROUTINE NURBSS(s_knot_count,
s_knots, t_knot_count, t_knots, s_stride,
t_stride, ctlarray, s_order, t_order, type)
INTEGER*4 s_knot_count, t_knot_count
REAL*8 s_knots(s_knot_count)
REAL*8 t_knots(t_knot_count)
INTEGER*4 s_stride, t_stride
INTEGER*4 s_order, t_order, type
REAL*8 ctlarray(*)
```

Description

The **nurbssurface** subroutine controls the shape of a Non-Uniform Rational B-Spline (NURBS) surface. Use this subroutine within a NURBS surface definition to describe the shape of a NURBS surface before any trimming takes place. To mark the beginning and end of a NURBS surface definition, use the **bgnsurface** and **endsurface** subroutines. Call the **nurbssurface** subroutine only within a NURBS surface definition.

You can trim a NURBS surface by using the **nurbscurve** and **pwlcurve** subroutines between calls to the **bgntrim** and **endtrim** subroutines.

The system renders a NURBS surface as a polygonal mesh and analytically calculates normal vectors at the corners of the polygons within the mesh. Therefore, your program should specify a lighting model when it uses NURBS surfaces, otherwise much detailed surface information is lost. Use the **lmdef** and **lmbind** subroutines to define or modify materials and their properties.

Note: If backfacing is on, the system cannot correctly eliminate backfacing polygons on the surface.

Parameters

s_knot_count
Specifies the number of knots in the parametric *s* direction.

s_knots
Specifies an array of *s_knot_count* nondecreasing knot values in the parametric *s* direction.

t_knot_count
Specifies the number of knots in the parametric *t* direction.

t_knots
Specifies an array of *t_knot_count* nondecreasing knot values in the parametric *t* direction.

s_stride
Specifies the offset (in bytes) between successive control points in the parametric *s* direction in *ctlarray*.

t_stride
Specifies the offset (in bytes) between successive control points in the parametric *t* direction in *ctlarray*.

ctlarray
Specifies an array containing control points for the NURBS surface. The coordinates must appear as either *(x, y, z)* triples or as *(x, y, z, w)* quadruples. The offsets between successive control points in the parametric *s* and *t* directions are given by *s_stride* and *t_stride*.

s_order
Specifies the order of the NURBS surface in the parametric *s* direction. The order is one more than the degree, so a cubic surface has an order of 4.

t_order
Specifies the order of the NURBS surface in the parametric *t* direction. The order is one more than the degree, so a cubic surface has an order of 4.

type Specifies a value indicating the control point type. Current options are:

C	FORTRAN	Description
N_XYZ	NXYZ	Indicates 3-dimensional control points.
N_XYZW	NXYZW	Indicates 4-dimensional (rational) control points.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Marking the beginning and end of a NURBS surface definition with the **bgnsurface** and **endsurface** subroutines.

Marking the beginning and end of a NURBS surface trimming loop with the **bgntrim** and **endtrim** subroutines.

Returning the current value of a trimmed NURBS surfaces display property with the **getnurbsproperty** subroutine.

Controlling the shape of a NURBS trimming curve with the **nurbscurve** subroutine.

Describing a piecewise linear trimming curve for NURBS surfaces with the **pwlcurve** subroutine.

Setting a property for the display of trimmed NURBS with the **setnurbsproperty** subroutine.

Graphics Library Overview and Drawing NURBS Curves and Surfaces.

objdelete Subroutine

Purpose
Deletes subroutines from a display list.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void objdelete(Int32  tag1, Int32  tag2)
```

FORTRAN Syntax
```
SUBROUTINE OBJDEL(tag1, tag2)
INTEGER*4 tag1, tag2
```

Description
The **objdelete** subroutine is an object (display list) editing routine. It deletes the routines between the locations specified by the *tag1* and *tag2* parameters from an object. It also removes any tags defined between the locations specified by the *tag1* and *tag2* parameters, but the *tag1* and *tag2* parameters remain in the text.

The **objdelete** subroutine leaves the editing pointer at `TAG1` location after it executes.

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

> **Note:** This editing subroutine itself cannot be added to a display list.

Parameters
tag1	Specifies the tag indicating where the deletion should start.
tag2	Specifies the tag indicating where the deletion should stop.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Opening an object for editing with the **editobj** subroutine.

Inserting a routine into an object with the **objinsert** subroutine.

Replacing an existing display list routine with a new one with the **objreplace** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

objinsert Subroutine

Purpose
Inserts routines in an object at a specified location.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void objinsert(Int32  tag)
```

FORTRAN Syntax
```
SUBROUTINE OBJINS(tag)
INTEGER*4 tag
```

Description
The **objinsert** subroutine positions the editing pointer at the location specified by the *tag* parameter. All subsequent graphics primitives are inserted into the display list at the current location of the editing pointer. Each graphics primitive increments the editing pointer to point immediately after the most recently inserted subroutine (in the same way that the cursor advances in an ordinary text editing facility as letters are typed).

Use the **closeobj** to terminate insertion and close up the current display list or one of the positioning subroutines (**objdelete**, **objinsert**, or **objreplace**) to reposition the editing pointer.

> **Note:** This editing subroutine itself cannot be added to a display list.

Parameter
tag Specifies a tag within the object definition to be edited.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Closing an object with the **closeobj** subroutine.

Opening an object for editing with the **editobj** subroutine.

Marking a location in the display list with the **maketag** subroutine.

Deleting a routine from an object with the **objdelete** subroutine.

Replacing an existing display list routine with a new one with the **objreplace** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

objreplace Subroutine

Purpose

Replaces existing display list routines with new ones.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void objreplace(Int32  tag)
```

FORTRAN Syntax

```
SUBROUTINE OBJREP(tag)
INTEGER*4 tag
```

Description

The **objreplace** subroutine positions the editing pointer at the location specified by the *tag* parameter. All subsequent graphics primitives overwrite the display list at the current location of the editing pointer.

Each graphics primitive increments the editing pointer to point immediately after the most recently inserted subroutine (in the same way that the cursor advances in an ordinary text editing facility as letters are typed).

The **objreplace** subroutine requires that the new subroutine and the subroutine being overwritten are the same length; otherwise, this or other subroutines in the display list may be partially overwritten, resulting in unpredictable drawing when the object is executed. In general, no two dissimilar GL subroutines are the same length.

The **objreplace** subroutine is best used for changing the parameter values of a subroutine that has already been compiled into a display list. Use the **objdelete** and **objinsert** subroutines for more general replacement.

Use the **closeobj** subroutine to terminate editing and close up the current display list, or one of the positioning subroutines (**objdelete**, **objinsert**, or **objreplace**) to reposition the editing pointer.as a quick method of creating a new version of a routine.

Note: This editing subroutine itself cannot be added to a display list.

Parameter

tag Specifies a tag within the object definition to be edited.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Closing an object with the **closeobj** subroutine.

Opening an object for editing with the **editobj** subroutine.

Marking a location in the display list with the **maketag** subroutine.

Deleting a routine from an object with the **objdelete** subroutine.

Inserting a routine into an object with the **objinsert** subroutine.

Graphics Library Overview and Creating Objects (Display Lists).

---

onemap Subroutine

Purpose

Organizes the color map as one large map.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void onemap( )
```

FORTRAN Syntax

```
SUBROUTINE ONEMAP
```

Description

The **onemap** subroutine reorganizes the color map as a single large color map. On the 24-bit High-Performance 3-D Color Graphics Processor, this map is a 12-bit color map having 4096 entries. On the 8-bit High-Performance 3-D Color Graphics Processor, the map is an 8-bit color map having 256 entries.

The system is initially in onemap mode. However, if the mode has been changed to multimap and the **onemap** subroutine is called, the reorganization of the color map into one large map does not take effect until the **gconfig** subroutine is called.

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Understanding the Adapter″.)
2. This subroutine cannot be used to add to a display list.
3. 12-bit color index mode is not possible on the gt4 family of adapters.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Configuring the system with the **gconfig** subroutine.

Returning the organization of the current color map with the **getcmmode** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Selecting one of 16 small color maps with the **setmap** subroutine.

Working in Color Map and RGB Modes.

---

ortho or ortho2 Subroutine

Purpose

Defines an orthographic transformation.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

FORTRAN Syntax

```
SUBROUTINE ORTHO(left, right, bottom, top, near, far)
REAL left, right, bottom, top, near, far

SUBROUTINE ORTHO2(left, right, bottom, top)
REAL left, right, bottom, top
```

The foregoing routines are functionally the same. They differ only in that the **ortho** subroutine is used for 3-D applications and the **ortho2** subroutine is used for 2-D applications.

Description

The **ortho** and **ortho2** subroutines set the current projection transformation to be an orthographic perspective transformation. With an orthographic projection, figures do not get smaller as they recede in relation to the viewer. Orthographic projections also preserve angles.

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

The **ortho2** subroutine defines a 2-D clipping rectangle. When you use this subroutine with 3-D modeling coordinates, the *z* values are not transformed. Objects with z values outside the range **-**1 <= *z* <=1 are clipped out.

When the system is in single matrix mode, both the **ortho** and **ortho2** subroutines load a matrix onto the matrix stack, thus replacing the current top matrix. When the system is in viewing matrix mode or projection matrix mode, it replaces the current projection matrix without changing the matrix stack.

To be more technically accurate, the **ortho** and **ortho2** subroutines set the mapping from eye coordinates to normalized device coordinates (NDC). Clipping occurs in NDC; all drawing primitives (except for text and blits) are clipped to -w <= x, y, z <= +w. The map is such that the clipping plane -w=x (in NDC) appears to be at +w*left = x in eye coordinates, and so on for the other five sides. For most drawing primitives, w=1.

After the **ortho** subroutine completes, the eye coordinate system is set up so that x is to the right, y is up, and z is towards the viewer (out of the screen).

Parameters

left	Specifies the coordinate for the left vertical clipping plane.
right	Specifies the coordinate for the right vertical clipping plane.
bottom	Specifies the coordinate for the bottom horizontal clipping plane.
top	Specifies the coordinate for the top horizontal clipping plane.
near	Specifies the coordinate for the nearest depth clipping plane.
far	Specifies the coordinate for the farthest depth clipping plane.

Examples

1. The example C language program **paint.c** uses the **ortho2** subroutine to define the two-dimensional world coordinates that exactly fit the defined viewport.

2. The example C language program **backface.c** uses the **ortho** subroutine to map a rectangular volume of space to the viewport.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current matrix mode with the **mmode** subroutine.

Defining a perspective projection transformation in terms of a field of view with the **perspective** subroutine.

Defining a perspective projection transformation in terms of *x* and *y* coordinates with the **window** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

overlay Subroutine

Purpose

Sets the number of user-defined bitplanes used for overlay drawing.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void overlay(Int32  planes)
```

FORTRAN Syntax

```
SUBROUTINE OVERLA(planes)
INTEGER*4 planes
```

Description

The **overlay** subroutine sets the number of user-defined bitplanes used for overlay colors to 0, 2, or 4, depending on the installed adapter. An overlay color is the color that is placed on top of the standard pixel contents. The overlay color appears whenever any of its bits are nonzero. The system has either two or four bitplanes that can be allocated as either underlay or overlay. Call the **overlay** subroutine to set them as overlay bitplanes.

The High-Performance 8-bit 3-D Color Graphics Processor can be configured with either 0 or 2 overlay planes. The 8-bit adapter has a total of 2 auxiliary planes, which can be configured into 2/0 or 0/2 overlay/underlay. For example, setting the number of overlay planes to 2 forces the number of underlay planes to 0.

The High-Performance 24-bit 3-D Color Graphics Processor can be configured with either 0, 2, or 4 overlay planes. The 24-bit adapter has a total of 4 auxiliary planes, which can be configured into 4/0, 2/2, or 0/4 overlay/underlay. For example, setting the number of overlay planes to 4 forces the number of underlay planes to 0.

The POWER GXT1000 adapter has 8 overlay planes, all of which are nonmodifiable.

Call the **gconfig** subroutine after the **overlay** subroutine to activate the overlay setting.

When the drawing mode is OVERDRAW, all drawing occurs in the overlay bitplanes. In OVERDRAW mode, FLAT is the only available shading model.

Notes:

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Understanding the Adapter″)
2. This subroutine cannot be used to add to a display list.

Parameter

planes Specifies the number of bitplanes to use for overlay drawing.

Example

The example C language program **overlay.c** uses the **overlay** subroutine to set the number of overlay bitplanes to two.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER Gt4, and POWER Gt4x, and POWER GXT1000 adapters do not support hidden-line and hidden-surface removal when rendering into the overlay planes. The z-buffer is automatically disabled when drawmode (OVERDRAW) is specified. The z-buffer is automatically enabled when drawmode (NORMALDRAW) is specified.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the target frame buffer for drawing subroutines with the **drawmode** subroutine.

Reconfiguring the system with the **gconfig** subroutine.

Setting the number of bitplanes used for underlay colors with the **underlay** subroutine.

Configuring the Frame Buffer, and Writemasks and Logical Operations.

patch Subroutine

Purpose
Draws a surface patch.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void patch(Matrix  geomx, Matrix  geomy, Matrix  geomz)
```

FORTRAN Syntax
```
SUBROUTINE PATCH(geomx, geomy, geomz)
REAL geomx(4,4), geomy(4,4), geomz(4,4)
```

Description
The **patch** subroutine draws a surface patch using the current settings from the **patchbasis**, **patchprecision**, and **patchcurves** subroutines. The control points *geomx*, *geomy*, and *geomz* determine the shape of the patch. The **rpatch** subroutine is essentially the same except that it draws a rational surface patch.

Parameters

geomx	Specifies a 4x4 matrix containing the *x* coordinates of the 16 control points of the patch.
geomy	Specifies a 4x4 matrix containing the *y* coordinates of the 16 control points of the patch.
geomz	Specifies a 4x4 matrix containing the *z* coordinates of the 16 control points of the patch.

Example
The example C language program **patch1.c** uses the **patch** subroutine to draw three surface patches after defining the settings for each patch.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Defining a cubic spline basis matrix with the **defbasis** subroutine.

Setting the current spline surface basis matrices with the **patchbasis** subroutine.

Setting the number of curves used to represent a patch with the **patchcurves** subroutine.

Setting the precision at which curves are drawn with the **patchprecision** subroutine.

Drawing a rational surface patch with the **rpatch** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

patchbasis Subroutine

Purpose
Sets the current spline surface basis matrices.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void patchbasis(Int32  uid, Int32  vid)
```

FORTRAN Syntax
```
SUBROUTINE PATCHB(uid, vid)
INTEGER*4 uid, vid
```

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

Parameters

uid Specifies the basis that defines how the control points determine the shape of the patch in the *u* direction.
vid Specifies the basis that defines how the control points determine the shape of the patch in the *v* direction.

Example
The example C language program **patch1.c** calls the **patchbasis** subroutine before calling the **patch** subroutine to set the basis matrix for drawing the curves that represent each surface patch.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a cubic spline basis matrix with the **defbasis** subroutine.

Drawing a surface patch with the **patch** subroutine.

Setting the number of curves used to represent a patch with the **patchcurves** subroutine.

Setting the precision at which curves are drawn with the **patchprecision** subroutine.

Drawing a rational surface patch with the **rpatch** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

patchcurves Subroutine

Purpose

Sets the number of curves used to represent a patch.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void patchcurves(Int32  ucurves, Int32  vcurves)
```

FORTRAN Syntax

```
SUBROUTINE PATCHC(ucurves, vcurves)
INTEGER*4 ucurves, vcurves
```

Description

The **patchcurves** subroutine sets the number of *u* and *v* curves that represents a patch as a wire frame.

Parameters

ucurves	Specifies the number of curve segments to be drawn in the *u* direction.
vcurves	Specifies the number of curve segments to be drawn in the *v* direction.

Example

The example C language program **patch1.c** uses the **patchcurves** subroutine to define to the **patch** subroutine the number of curves to use in drawing three surface patches.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a surface patch with the **patch** subroutine.

Setting the current spline surface basis matrices with the **patchbasis** subroutine.

Setting the precision at which curves are drawn with the **patchprecision** subroutine.

Drawing a rational surface patch with the **rpatch** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

patchprecision Subroutine

Purpose

Sets the precision at which curves are drawn in a patch.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void patchprecision(Int32  usegments, Int32  vsegments)
```

FORTRAN Syntax

```
PATCHP(usegments, vsegments)
INTEGER*4 usegments, vsegments
```

Description

The **patchprecision** subroutine sets the precision with which the system draws curves to make up a wire frame patch. Patch precisions are similar to curve precisions; they specify the minimum number of line segments used to draw a patch.

Parameters

usegments	Specifies the number of line segments used to draw a curve in the *u* direction.
vsegments	Specifies the number of line segments used to draw a curve in the *v* direction.

Example

The example C language program **patch1.c** calls the **patchprecision** subroutine to set the precision for drawing the curves that represent surface patches before calling the **patch** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the number of line segments that draw a curve segment with the **curveprecision** subroutine.

Drawing a surface patch with the **patch** subroutine.

Setting the current spline surface basis matrices with the **patchbasis** subroutine.

Setting the number of curves used to represent a patch with the **patchcurves** subroutine.

Drawing a rational surface patch with the **rpatch** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

pclos Subroutine

Purpose

Closes a filled polygon.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void pclos( )
```

FORTRAN Syntax

```
SUBROUTINE PCLOS
```

Description

The **pclos** subroutine closes a filled polygon that has been created by using the **pmv** subroutine and a sequence of calls to the **pdr**, **rpmv**, or **rpdr** subroutines. It is not needed when using the **poly** or **polf** subroutines because these procedures close the polygon within their own routines.

The **pclos** subroutine closes the polygon by connecting the last point with the first. The polygon so defined is filled using the current fill area attributes: pattern, current color, and writemask.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 calls to the **pdr** subroutine between calls to the **pmv** and **pclos** subroutines.

Note: Be careful not to confuse **pclos** with the base operating system subroutine **pclose,** which closes a pipe, in the **libc.a** library.

Example

The example C language program **zbuffer1.c** uses the **pclos** subroutine to specify the end of a filled polygon definition.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the starting point for a polygon with the **pmv** subroutine.

Drawing a filled polygon with the **polf** subroutine.

Drawing a polygon with the **poly** subroutine.

Drawing a relative polygon with the **rpdr** subroutine.

Moving the current graphics position to a starting point for a filled polygon relative to the current point with the **rpmv** subroutine.

Drawing a shaded filled polygon with the **splf** subroutine.

Graphics Library Overview, Drawing with Move-Draw Style Subroutines, and Setting Drawing Attributes.

---

pdr Subroutine

Purpose

Specifies the next point in a filled polygon.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

void pdri
(Icoord  x, Icoord  y, Icoord  z)
```

```
void pdrs
(Scoord  x, Scoord  y, Scoord  z)

void pdr2
(Coord  x, Coord  y)

void pdr2i
(Icoord  x, Icoord  y)

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

FORTRAN Syntax

```
SUBROUTINE PDR(x, y, z)
REAL x, y, z
SUBROUTINE PDRI(x, y, z)
INTEGER*4 x, y, z
SUBROUTINE PDRS(x, y, z)
INTEGER*2 x, y, z
SUBROUTINE PDR2(x, y)
REAL x, y
SUBROUTINE PDR2I(x, y)
INTEGER*4 x, y
SUBROUTINE PDR2S(x, y)
INTEGER*2 x, y
```

> **Note:** For FORTRAN users, the INTEGER*2 versions of this subroutine, **PDRS** and **PDR2S**, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking one of the short versions of this subroutine with an integer constant will result in unexpected behavior.

Description

The **pdr** subroutine specifies the next point of a polygon. When the subroutine is executed, it draws a line to the specified point (*x*, y, z), which then becomes the current graphics position. The next call to the **pdr** subroutine starts drawing from that point.

To draw a typical polygon, start with a call to the **pmv** subroutine, follow it with a sequence of calls to the **pdr** subroutine, and end it with a call to the **pclos** subroutine.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 calls to the **pdr** subroutine between calls to the **pmv** and **pclos** subroutines.

The six different forms for the **pdr** subroutine are as follows:

	2-D	**3-D**
Int16	pdr2s	pdrs
Int32	pdr2i	pdri
float	pdr2	pdr

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **pdr** expects real coordinates, **pdri** expects integer coordinates, and **pdrs** expects short integer coordinates. In addition, the **pdr2** routines assume a 2-D point instead of a 3-D point.

Parameters

x Specifies the *x* coordinate of the next defining point for the polygon.
y Specifies the the *y* coordinate of the next defining point for the polygon.
z Specifies the *z* coordinate of the next defining point for the polygon.

Example

The example C language program **zbuffer1.c** uses the **pdr** subroutine to draw the edges of a filled polygon.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Moving the current graphics position to a point relative to the current point with the **rmv** subroutine.

Drawing a relative polygon with the **rpdr** subroutine.

Moving the current graphics position to a starting point for a filled polygon relative to the current point with the **rpmv** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Graphics Library Overview, Drawing with Move-Draw Style Subroutines, and Setting Drawing Attributes.

---

perspective Subroutine

Purpose

Defines a perspective projection transformation.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void   perspective
(Angle  fovy,
Float32  aspect,
Coord  near, Coord  far)
```

FORTRAN Syntax

```
SUBROUTINE PERSPE(fovy, aspect, near, far)
INTEGER*4 fovy
REAL aspect, near, far
```

Description

The **perspective** subroutine defines a perspective projection transformation by specifying a viewing pyramid in the eye coordinate system. The pyramid comprises:

- The field-of-view angle in the *y* direction of the eye coordinate system
- The aspect ratio that determines the field-of-view in the *x* direction
- The distance to the *near* and *far* clipping planes in the *z* direction.

The field of view is the range of the area that is being viewed. The aspect ratio is the ratio of *x* (width) to *y* (height), and 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 has the same aspect ratio as the frustum, it displays the image without distortion.

The **perspective** subroutine is very similar to the **window** subroutine. The only difference between these two is the manner in which the parameters specify the viewing frustum.

After the **perspective** subroutine completes, the eye coordinate system is set up so that x is to the right, y is up, and z is towards the viewer (out of the screen).

When the system is in single matrix mode, the **perspective** subroutine loads a matrix onto the transformation stack, replacing the current top matrix. When the system is in viewing matrix mode or projection matrix mode, the **perspective** subroutine replaces the current projection matrix and leaves the matrix stack unchanged.

Parameters

fovy	Specifies the field-of-view angle in the *y* direction. The field of view is the range of the area that is being viewed. It is measured in tenths of a degree. The value of *fovy* must be >= 2 (two-tenths of one degree) or an error results.
aspect	Specifies 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	Specifies the distance from the viewer to the closest clipping plane (always positive).
far	Specifies the distance from the viewer to the farthest clipping plane (always positive).

Example

The example C language program **zbuffer1.c** uses the **perspective** subroutine to load a projection transformation as the current transformation matrix.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a 3-D orthographic transformation with the **ortho** subroutine.

Defining a 2-D orthographic transformation with the **ortho2** subroutine.

Defining a perspective projection transformation in terms of *x* and *y* coordinates with the **window** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

pick Subroutine

Purpose

Places the system in picking mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void pick(Int16  buffer [ ], Int32  bufferlen)
```

FORTRAN Syntax

```
SUBROUTINE PICK(buffer, bufferlen)
INTEGER*4 bufferlen
INTEGER*2 buffer(bufferlen)
```

Description

The **pick** subroutine places the system in picking mode. When the system is in picking mode, the extent of all subsequent drawing primitives are compared to a picking volume. The picking volume is defined by the location of the cursor when the **pick** subroutine was called and by the picking volume size.

If a drawing primitive overlaps or intrudes upon the picking volume, a hit has occurred. The hit is recorded only if the name stack has been touched since the last hit. Any of the subroutines **loadname**, **pushname**, or **popname** touch the name stack. The first hit after picking begins is always recorded.

A hit is recorded by placing the depth of the name stack into the next vacant slot in the buffer, followed by the entire contents of the name stack. The bottom of the name stack is transferred to the buffer first, followed by the second from the bottom entry of the name stack, and so forth. In other words, the data from bottom to top is mapped from left to right.

With one exception, all drawing routines cause hits, including clear, points, lines, polygons, arcs, circles, curves, patches, and NURBS. The **charstr** subroutine does not cause a hit, although **cmov** and **cmov2** do cause hits.

Picking does not work if you issue a new viewport in picking mode.

Nothing is drawn to the screen when the system is in picking mode. Instead, drawing commands are piped to the picking mechanism and used to determine pick or select region hits. On most systems, nothing is actually placed in the pick buffer until the **endpick** subroutine is called.

Note: This subroutine cannot be used to add to a display list.

Parameters

buffer	Specifies the array to use for storing names.
bufferlen	Specifies the length of the array specified in the *buffer* parameter.

Example

The example C language program **pick1.c** calls the **pick** subroutine to pick objects when the left mouse button is pressed.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning off picking mode with the **endpick** subroutine.

Putting the system in selecting mode with the **gselect** subroutine.

Initializing the name stack with the **initnames** subroutine.

Loading the name on top of the name stack with the **loadname** subroutine.

Setting the dimensions of the picking region with the **picksize** subroutine.

Popping a name off the name stack with the **popname** subroutine.

Pushing a new name onto the name stack with the **pushname** subroutine.

Graphics Library Overview, Picking and Selecting Overview, and Using Viewports and Screenmasks.

---

picksize Subroutine

Purpose

Sets the dimensions of the picking region.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void picksize(Int16  deltax, Int16  deltay)
```

FORTRAN Syntax

```
SUBROUTINE PICKSI(deltax, deltay)
INTEGER*4 deltax, deltay
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **picksize** subroutine sets the dimensions of the picking region. The default setting is 10x10 pixels. In picking mode, any drawing primitives that intersect the picking region are reported as hits.

If a drawing primitive overlaps or intrudes upon the picking volume, a hit is recorded. The hit is recorded by placing the depth of the name stack into the next vacant slot in the buffer, followed by the entire contents of the name stack.

The bottom of the name stack is transferred to the buffer first, followed by the second from the bottom entry of the name stack, and so forth. In other words, the data from bottom to top is mapped from left to right.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

deltax	Specifies the new width of the picking region.
deltay	Specifies the new height of the picking region.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Putting the system in picking mode with the **pick** subroutine.

Graphics Library Overview and Picking and Selecting Overview.

---

pixmode Subroutine

Purpose

Controls the operation of pixmap transfers with the **lrectread** and **lrectwrite** subroutines.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void pixmode (Int32  mode, Int32  value)
```

FORTRAN Syntax

```
SUBROUTINE PIXMOD (mode, value)
INTEGER*4 mode, value
```

Description

The **pixmode** subroutine controls the manner in which the arguments of the **lrectread** and **lrectwrite** subroutines are interpreted while performing pixel transfers to and from the frame buffer. That is, the **pixmode** subroutine sets the attributes associated with a pixel-transfer operation. The attributes that can be set by this function *do not* control the operation of the **rectcopy** routine.

This subroutine is intended to provide a device-independent interface to the reading and writing of pixmaps. It should be used to specify the format of the pixmaps maintained by the application. Transfers occurring with the **lrectread** and **lrectwrite** subroutines then automatically convert the incoming or outgoing data to or from the appropriate format for the internal device of the frame buffer.

Only the **lrectread** and **lrectwrite** subroutines are affected by these attributes. The operation of the **rectread**, **rectwrite**, and **rectcopy** subroutines are not affected.

The following information describes the various pixel transfer modes that are supported on the POWER Gt4 and POWER Gt4x adapters.

> **Note:** The 3-D Color Graphics Processor, the POWERgraphics GTO, and the POWERstation 730 adapters support only PM_FASTMODE.

Mode	Default and Behavior
PM_TTOB	Default = False Indicates pixel row ordering from top to bottom or bottom to top. A value of True indicates that the top row comes first in the data, followed by remaining rows, top to bottom. A value of False indicates that the bottom row comes first in the data, followed by remaining rows, bottom to top.

Mode	Default and Behavior
PM_STRIDE	Default = 0 Indicates the number of 32-bit words between the starts of successive scan lines. On the POWER Gt4 and POWER Gt4x adapters, the stride must be positive and greater than the width of the pixmap (the width as specified with the **lrectread** and **lrectwrite** subroutines). If the stride is less than the width, the stride is to be changed to equal the width. A stride factor that is different than the width is used to transfer subportions of an image to and from the screen. There are two scenarios: 1) A subportion of a large image on the system is to be transferred to the frame buffer. In this case, the row stride describes the length of a row on the system, while the arguments to the **lrectwrite** subroutine specify the subrectangle to be written. Note that the width specified through the **lrectwrite** subroutine is smaller than the row stride. 2) A subportion of a large image in the frame buffer is to be transferred to system memory. The subregion of the frame buffer is indicated with the **lrectread** subroutine. If the row stride factor is greater than the width specified with the **lrectread** subroutine, then only a *width* number of pixels in each row is transferred from the screen to the system. System memory beyond *width*, but before *stride* is not touched.
PM_SIZE	Default = 32 Indicates the number of bits between successive pixels in the data array. For instance, if the pixel size factor is 8, there are four pixels packed into one 32-bit word so that the system returns packed pixels while reading, and assumes that the user has presented packed pixels in the data when writing. The POWER Gt4 and POWER Gt4x adapters support values of 8 and 32. Other values are ignored. The setting of the PM_SIZE mode is ignored when reading from and writing to the overlay planes and the z-buffer. A size of 32 or one pixel per word is assumed.
PM_FASTMODE	Default = True Indicates that the pixel transfer mode should be set automatically to values that maximize the hardware performance of pixel transfers.
All Adapters	Default = False Pixel transfers proceed from left to right and then from bottom to top. Each pixel occupies one 32-bit word. RGB mode pixels are packed as 0xAABBGGRR. Colorindex mode pixels occupy the lowest order bits of the word.
3-D Color Graphics Processor	Default = True The behavior is the same as PM_FASTMODE with default = False.

Different adapters have different pixel transfer methods. If an adapter has more than one fast pixel transfer mode, values that are closest to the default values are chosen. The following information summarizes the fast transfer modes for the POWER Gt4 and POWER Gt4x adapters.

When PM_FASTMODE is set and True is specified, the pixel format depends on the current frame buffer configuration. When the window is in color-index mode, PM_SIZE is automatically set to 8; when in RGBmode, PM_SIZE is automatically set to 32.

The POWER Gt4 and POWER Gt4x adapters natively support a packed pixel mode, with four 8-bit pixels stuffed into a 32-bit word. To obtain this transfer mode in colorindex mode, set PM_SIZE to 8.

Parameters

mode A token representing the transfer mode to be changed.

The actual behavior of the device when placed in FASTMODE TRUE is device specific.

value The value to be associated with the pixel transfer mode.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The **pixmode** subroutine is currently inoperational on the POWERgraphics GXT1000.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** or **lrectwrite** subroutine.

Reading a rectangular array of pixels into host memory with the **rectread** or **lrectread** subroutine.

Graphics Library Overview and Configuring the Frame Buffer.

Reading and Writing Pixels.

---

pmv Subroutine

Purpose

Moves to the starting point for a filled polygon.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

void pmv2
(Coord  x, Coord  y)

void pmv2i
(Icoord  x, Icoord  y)

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

FORTRAN Syntax

```
SUBROUTINE PMV(x, y, z)
REAL x, y, z
SUBROUTINE PMVI(x, y, z)
INTEGER*4 x, y, z
SUBROUTINE PMVS(x, y, z)
INTEGER*2 x, y, z
SUBROUTINE PMV2(x, y)
REAL x, y
SUBROUTINE PMV2I(x, y)
INTEGER*4 x, y
SUBROUTINE PMV2S(x, y)
INTEGER*2 x, y
```

> **Note:** For FORTRAN users, the INTEGER*2 versions of this subroutine, **PMVS** and **PMV2S**, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking one of the short versions of this subroutine with an integer constant will result in unexpected behavior.

Description

The **pmv** subroutine specifies the starting point of a filled polygon. To draw a typical polygon, start with a call to the **pmv** subroutine, follow it with a sequence of calls to the **pdr** subroutine, and end it with a call to the **pclos** subroutine.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 calls to the **pdr** subroutine between calls to the **pmv** and **pclos** subroutines.

Between calls to the **pmv** and **pclos** subroutines, you can issue calls only to the following GL subroutines:

- **c**
- **color**
- **cpack**
- **lmbind**
- **lmcolor**

- **lmdef**
- **n3f**
- **normal**
- **pdr**
- **RGBcolor**
- **v**

Use the **lmdef** and **lmbind** subroutines to respecify only materials and their properties.

The six different forms for the **pmv** subroutine are as follows:

	2-D	**3-D**
Int16	pmv2s	pmvs
Int32	pmv2i	pmvi
float	pmv2	pmv

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **pmv** expects real coordinates, **pmvi** expects integer coordinates, and **pmvs** expects short integer coordinates. In addition, the **pmv2** routines assume a 2-D point instead of a 3-D point.

Parameters

x Specifies the *x* coordinate of the starting point for the polygon.
y Specifies the *y* coordinate of the starting point for the polygon.
z Specifies the *z* coordinate of the starting point for the polygon.

Example

The example C language program **zbuffer1.c** uses the **pmv** subroutine to move to the beginning position for drawing a filled polygon.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Forcing the system to draw concave polygons correctly with the **concave** subroutine.

Closing a filled polygon with the **pclos** subroutine.

Drawing a polygon with the **poly** subroutine.

Drawing a relative polygon with the **rpdr** subroutine.

Moving the current graphics position to a starting point for a filled polygon relative to the current point with the **rpmv** subroutine.

Selecting the shading model used to draw a polygon with the **shademodel** subroutine.

Graphics Library Overview, Creating Lighting Effects, Drawing with Move-Draw Style Subroutines, Setting Drawing Attributes, and Working in Color Map and RGB Modes.

---

pnt Subroutine

Purpose

Draws a point in modeling coordinates.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

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

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

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

void pnt2
(Coord  x, Coord  y)

void pnt2i
(Icoord  x, Icoord  y)

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

FORTRAN Syntax

```
SUBROUTINE PNT(x, y, z)
REAL x, y, z
SUBROUTINE PNTI(x, y, z)
INTEGER*4 x, y, z
SUBROUTINE PNTS(x, y, z)
INTEGER*2 x, y, z
SUBROUTINE PNT2(x, y)
REAL x, y
SUBROUTINE PNT2I(x, y)
INTEGER*4 x, y
SUBROUTINE PNT2S(x, y)
INTEGER*2 x, y
```

Note: For FORTRAN users, the INTEGER*2 versions of this subroutine, **PNTS** and **PNT2S**, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer

constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking one of the short versions of this subroutine with an integer constant will result in unexpected behavior.

Description

The **pnt** subroutine draws a point in modeling coordinates. If the point is visible in the current viewport, it is shown as one pixel. The pixel is drawn in the current point attributes: color (if in depth-cue mode, the depth-cued color is used) and writemask. The **pnt** subroutine updates the current graphics position after it executes. A drawing routine immediately following the **pnt** subroutine will start drawing from the point specified.

The six different forms for the **pnt** subroutine are as follows:

	2-D	**3-D**
Int16	pnt2s	pnts
Int32	pnt2i	pnti
float	pnt2	pnt

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **pnt** expects real coordinates, **pnti** expects integer coordinates, and **pnts** expects short integer coordinates. In addition, the **pnt2** routines assume a 2-D point instead of a 3-D point.

Parameters

x Specifies the *x* coordinate of the point.
y Specifies the *y* coordinate of the point.
z Specifies the *z* coordinate of the point.

Example

The example C language program **depthcue.c** uses the **pnt** subroutine to draw random points.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a line with the **draw** subroutine.

Moving the current graphics position to a specified point with the **move** subroutine.

Graphics Library Overview, Drawing with Move-Draw Style Subroutines, Performing Depth-Cueing, Setting Drawing Attributes, and Working with Coordinate Systems.

pntsmooth Subroutine

Purpose
Turns point antialiasing on and off.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void pntsmooth(Int32  mode )
```

FORTRAN Syntax
```
SUBROUTINE PNTSMO(mode)
INTEGER*4 mode
```

Description
The **pntsmooth** subroutine allows the drawing of antialiased points in color map and RGB modes.

In RGB mode, the current color is blended with the contents of the frame buffer in a read-modify-write operation. That is, the new pixel color is a calculated blend of the old pixel color and the current color, based on the percentage of overlap of the point with the pixel.

In color map mode, the low-order 4 bits of the current color index are replaced with a value representing the pixel coverage. Therefore, a color ramp must be loaded, with the low-order 4 bits of the color ramp blending between the foreground and background colors.

> **Note:** In order for antialiased lines and points to appear visually smooth, gamma correction MUST be performed. A gamma correction factor in the range of 2.4 to 2.7 is suggested. If gamma correction is not performed, lines do not appear smooth, but exhibit a *roping* or *braiding* effect, as if the line were composed of separate, intertwining strands. For more information on gamma-correcting antialiased lines, see Smoothing Jagged Lines with Antialiasing in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Parameter
mode Sets point-drawing mode as follows:

Mode Constants		
C	**FORTRAN**	**Description**
SMP_ON	SMPON	Turns on point-antialiasing capabilities.
SMP_OFF	SMPOFF	Turns off point-antialiasing capabilities.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

> **Notes:**

1. The 3-D Color Graphics Processor supports antialiased points only in color map mode. On this adapter, the value `0xf` represents maximum pixel coverage, and `0x0` represents minimum pixel coverage. That is, the following color ramp should be loaded:

3-D Color Graphics Processor Color Ramp	
Low-Order 4 Bits	**Ramp**
0000	(1/16) * foreground color + (15/16) * background color
0001	(2/16) * foreground color + (14/16) * background color
0010	(3/16) * foreground color + (13/16) * background color
. . . .	. . . . .
1111	(16/16) * foreground color + (0/16) * background color

   For the 3-D Color Graphics Processor, the appearance of the intersections of color-index-mode antialiased lines and points can be significantly improved by setting the value of the **zsource** subroutine to ZSRC_COLOR and the value of the **zfunction** subroutine to ZF_GREATER. Refer to the subsection on improving intersections within Smoothing Jagged Lines with Antialiasing in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* for more details.
2. The Supergraphics Processor supports antialiased points in RGB mode only.
3. The POWER Gt4 and POWER Gt4x adapters do not support antialiased point rendering.
4. When anti-aliased lines and points are drawn in RGB mode, a coverage factor (alpha value) for pixels on and near the anti-aliased objects is computed. This coverage factor is used to blend the color of the anti-aliased point or line with the contents of the frame buffer. The way in which the blending occurs is controlled by the **blendfunction** subroutine. When rendering anti-aliased objects in RGB mode, set the **blendfunction** to an appropriate value.
5. The POWERgraphics GXT1000 supports anti-aliased point rendering in both color-index and RGB rendering modes.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing vertex-based points with the **bgnpoint** subroutine.

Specifying the alpha blending ratio with the **blendfunction** subroutine.

Ending a series of vertex-based points with the **endpoint** subroutine.

Drawing a point with the **pnt** subroutine.

Controlling the placement of point, line, and polygon vertices with the **subpixel** subroutine.

Transferring a vertex to the graphics pipe with the **v** subroutine.

Smoothing Jagged Lines with Antialiasing, Understanding the Hardware Used by GL, Working in Color Map and RGB Modes, Performing Depth-Cueing, Reading and Writing Pixels.

polarview Subroutine

Purpose
Defines the viewer's position in polar coordinates.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void polarview
(Coord  distance,
Angle  azimuth,
Angle  incidence,
Angle  twist)
```

FORTRAN Syntax
```
VOID POLARV(distance, azimuth, incidence, twist)
REAL distance
INTEGER*4 azimuth, incidence, twist
```

Description
The **polarview** subroutine defines the viewer's position in polar coordinates. Normally, the **polarview** subroutine is used to set up the mapping from world coordinates to eye coordinates (equivalently, to define the location of the viewer's eye in world coordinates).

If the **polarview** subroutine is the first transformation subroutine called after projection matrix is set up and the matrix stack is initialized, it sets up such a mapping. The eye is located a distance, given in the *distance* parameter, from the world space origin. The line of sight extends from the eye through the world space origin (that is, the viewer is looking squarely upon the origin). The incidence and azimuth are measured with respect to the world coordinate system.

The **polarview** subroutine can also be used as a modeling transformation. Whether it behaves as a viewing transformation or a modeling transformation depends entirely on the order in which it is called with respect to the drawing subroutines.

Parameters
distance	Specifies the distance from the eye to the world space origin.
azimuth	Specifies the azimuthal angle in the *x-y* plane, measured clockwise from the positive *y* axis. The angle must be specified as an integer, in tenths of a degree.
incidence	Specifies the angle of incidence in the *y-z* plane, measured from the *z* axis. The incidence angle is the angle of the line from origin to eye with respect to the *z* axis (the angle away from direct vertical, if you think of standing on the *x-y* plane. The angle must be specified as an integer, in tenths of a degree.
twist	Specifies the amount that the viewpoint is to be rotated around the line of sight using the right-hand rule. The angle must be specified as an integer, in tenths of a degree.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a viewing transformation with the **lookat** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

polf Subroutine

Purpose

Draws a filled polygon.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void polf(Int32  n, Coord  parray[ ][3])

void polfi(Int32  n, Icoord  parray[ ][3])

void polfs(Int32  n, Scoord  parray[ ][3])

void polf2(Int32  n, Coord  parray[ ][2])

void polf2i(Int32  n, Icoord  parray[ ][2])

void polf2s(Int32  n, Scoord  parray[ ][2])
```

FORTRAN Syntax

```
SUBROUTINE POLF(n, parray)
INTEGER*4 n
REAL parray(3,n)

SUBROUTINE POLFI(n, parray)
INTEGER*4 n
INTEGER*4 parray(3,n)

SUBROUTINE POLFS(n, parray)
INTEGER*4 n
INTEGER*2 parray(3,n)

SUBROUTINE POLF2(n, parray)
INTEGER*4 n
REAL parray(2,n)
```

```
SUBROUTINE POLF2I(n, parray)
INTEGER*4 n
INTEGER*4 parray(2,n)
SUBROUTINE POLF2S(n, parray)
INTEGER*4 n
INTEGER*2 parray(2,n)
```

Description

The **polf** subroutine draws filled polygons using the current area attributes: pattern, color, and writemask. Polygons are represented as arrays of points. The first and last points automatically connect to close a polygon. After the polygon is filled, the current graphics position is set to the first point in the array.

Polygons in 2-D are drawn with $z = 0$.

The six different forms for the **polf** subroutine are as follows:

	2-D	**3-D**
Int16	polf2s	polfs
Int32	polf2i	polfi
float	polf2	polf

The syntax for each of the subroutine forms is the same except for the first argument. They differ only in that **polf** expects real coordinates, **polfi** expects integer coordinates, and **polfs** expects short integer coordinates. In addition, the **polf2** routines assume a 2-D point instead of a 3-D point.

There can be no more than 256 vertices in a polygon. In addition, the **polf** subroutine cannot correctly draw polygons that intersect themselves.

Parameters

n	Specifies the number of points in the polygon.
parray	Specifies an array containing the vertices of the polygon.

Example

The example C language program **tpbig.c** uses the **polf2i** subroutine to draw an ice cream cone.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allowing the system to draw concave polygons with the **concave** subroutine.

Specifying the next point in a polygon with the **pdr** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Drawing a polygon with the **poly** subroutine.

Drawing a filled rectangle with the **rectf** subroutine.

Drawing a rectangle with the **rect** subroutine.

Drawing a relative polygon with the **rpdr** subroutine.

Moving the current graphics position to a starting point for a filled polygon relative to the current point with the **rpmv** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

poly Subroutine

Purpose

Draws a polygon.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void poly(Int32  n, Coord  parray[ ][3])

void polyi(Int32  n, Icoord  parray[ ][3])

void polys(Int32  n, Scoord  parray[ ][3])

void poly2(Int32  n, Coord  parray[ ][2])

void poly2i(Int32  n, Icoord  parray[ ][2])

void poly2s(Int32  n, Scoord  parray[ ][2])
```

FORTRAN Syntax

```
SUBROUTINE POLY(n, parray)
INTEGER*4 n
REAL parray(3,n)
SUBROUTINE POLYI(n, parray)
INTEGER*4 n
INTEGER*4 parray(3,n)
SUBROUTINE POLYS(n, parray)
INTEGER*4 n
INTEGER*2 parray(3,n)
SUBROUTINE POLY2(n, parray)
INTEGER*4 n
REAL parray(2,n)
```

```
SUBROUTINE POLY2I(n, parray)
INTEGER*4 n
INTEGER*4 parray(2,n)
SUBROUTINE POLY2S(n, parray)
INTEGER*4 n
INTEGER*2 parray(2,n)
```

Description

The **poly** subroutine draws polygons using the current line attributes: linestyle, linewidth, color, and writemask. Polygons are represented as arrays of points. The first and last points automatically connect to close a polygon.

Polygons in 2-D are drawn with $z = 0$.

The six different forms for the **poly** subroutine are as follows:

	2-D	**3-D**
Int16	poly2s	polys
Int32	poly2i	polyi
float	poly2	poly

The syntax for each of the subroutine forms is the same except for the first argument. They differ only in that **poly** expects real coordinates, **polyi** expects integer coordinates, and **polys** expects short integer coordinates. In addition, the **poly2** routines assume a 2-D point instead of a 3-D point.

There can be no more than 256 vertices in a polygon.

Parameters

n	Specifies the number of points in the polygon.
parray	Specifies an array containing the vertices of the polygon.

Example

The example C language program **tpbig.c** uses the **poly2i** subroutine to outline an ice cream cone in black.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allowing the system to draw concave polygons with the **concave** subroutine.

Specifying the next point in a polygon with the **pdr** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Drawing a filled polygon with the **polf** subroutine.

Drawing a rectangle with the **rect** subroutine.

Drawing a filled rectangle with the **rectf** subroutine.

Drawing a relative polygon with the **rpdr** subroutine.

Moving the current graphics position to a starting point for a filled polygon relative to the current point with the **rpmv** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

polygonlist or polylinelist Subroutine

Purpose

Draws multiple, disjointed polygons or polylines.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void polygonlist(Int32 num_polygons, Int32 type, void *se)
void polylinelist(Int32 num_polylines, Int32 type, void *se)
```

FORTRAN Syntax

```
SUBROUTINE POLYLI(num_polygons, type, se)
INTEGER*4 num_polygons, type, se(1)
SUBROUTINE  LINELI(num_polylines, type, se)
INTEGER*4 num_polylines, type, se(1)
```

Description

The **polygonlist** subroutine draws multiple, disjoint polygons.

The **polylinelist** subroutine draws multiple, disjoint polylines. Polylines can be lit and/or shaded; therefore, polylines may be specified with color and/or normal data.

If lighting is not enabled, all normal data is ignored. If no color data is supplied, the polygons or polylines are rendered with the current color. If lighting is enabled, color data is interpreted according to the current setting of the **lmcolor** attribute.

1. The Supergraphics Processor Subsystem does not support lit and/or smooth-shaded polylines. It does support flat-shaded polylines.
2. The Supergraphics Processor Subsystem does not support 4-dimensional data. It does support 2- and 3-dimensional data.
3. These subroutines may not buy a performance improvement over begin-end style rendering on the POWERgraphics GXT1000. Begin-end style rendering has been highly optimized on the GXT1000.

Parameters

num_polygons
Specifies the number of polygons to be defined in the array pointed to by the *se* parameter in the **polygonlist** subroutine.

num_polylines
Specifies the number of polylines to be defined in the array pointed to by the *se* parameter in the **polylinelist** subroutine.

type Specifies a bitflag indicating whether normals and/or colors are present in the data and whether the vertex data is 2-, 3-, or 4-dimensional. This bitflag is formed by using the bitwise OR operator to combine the following tokens:

SE_2D	Indicates that the vertex coordinates are 2-dimensional.
SE_3D	Indicates that the vertex coordinates are 3-dimensional.
SE_4D	Indicates that the vertex coordinates are 4-dimensional.
SE_NORM	Indicates that normal data is present.
SE_COLOR	Indicates that color index data is present.
SE_RGB	Indicates that RGB data is present.

se Points to an array of mixed integer, binary, and floating point data. The array is a concatenation of a number of separate, disjoint polygons (polylines) as follows:

Polygons Data Structure		
Word	**Parameter**	**Type**
0	polygon (1)	polygon format
.	polygon (2)	polygon format
	...	
.	polygon (n)	polygon format

Each polygon consists of an integer indicating the number of vertices in that polygon, followed by vertex, normal, and color information. If the array is considered as a linear array of 32-bit words, the structure of the polygon data must be as follows:

Individual Polygon Data Structure		
Word	**Parameter**	**Type**
+0	num verts	signed 32-bit integer
+1	vertex (1)	vertex data
.	vertex (2)	vertex data
	...	
.	vertex (n)	vertex data

Each vertex has one of a number of formats, depending on the setting of the flag word. The flag bits indicate whether a normal vector is or is not present, whether a color is or is not present, and whether the point data is 2-, 3-, or 4-dimensional. Therefore, the potential data for a vertex data structure is as follows:

Potential Vertex Data Structure		
Word	**Parameter**	**Type**
+ 0	x	IEEE short Floating Point

+1	y	IEEE short Floating Point
+2 if 3D	z	IEEE short Floating Point
+3 if 4D	w	IEEE short Floating Point
2/3/4+0 if NORM	nx	IEEE short Floating Point
2/3/4+1 if NORM	ny	IEEE short Floating Point
2/3/4+2 if NORM	nz	IEEE short Floating Point
2...7+0 if CI	color index	IEEE integer
2...7+0 if RGB	color r	IEEE short Floating Point
2...7+1 if RGB	color g	IEEE short Floating Point
2...7+2 if RGB	color b	IEEE short Floating Point

More specifically, if there is no normal or color data, and the point contains only 2-dimensional data, a vertex data structure is no more than a pair of floating point numbers as follows:

type = SE_2D		
Word	**Parameter**	**Type**
0	x	IEEE short Floating Point
1	y	IEEE short Floating Point

If a normal vector is to be specified with 3-dimensional data, the vertex data structure would be as follows:

type = SE_3D | SE_NORM		
Word	**Parameter**	**Type**
0	x	IEEE short Floating Point
1	y	IEEE short Floating Point
2	z	IEEE short Floating Point
3	nx	IEEE short Floating Point
4	ny	IEEE short Floating Point
5	nz	IEEE short Floating Point

A 4-dimensional vertex with color index data would have the following data structure:

type = SE_4D | SE_COLOR		
Word	**Parameter**	**Type**
0	x	IEEE short Floating Point
1	y	IEEE short Floating Point
2	z	IEEE short Floating Point
3	w	IEEE short Floating Point
4	color index	32-bit integer

A 3-dimensional vertex with both normal and color data would have the following data structure:

type = SE_3D | SE_NORM | SE_RGB	

Word	Parameter	Type
0	x	IEEE short Floating Point
1	y	IEEE short Floating Point
2	z	IEEE short Floating Point
3	nx	IEEE short Floating Point
4	ny	IEEE short Floating Point
5	nz	IEEE short Floating Point
6	R	IEEE short Floating Point
7	G	IEEE short Floating Point
8	B	IEEE short Floating Point

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allowing the system to draw concave polygons with the **concave** subroutine.

Specifying the next point in a polygon with the **pdr** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Drawing a relative polygon with the **rpdr** subroutine.

Moving the current graphics position to a starting point for a filled polygon relative to the current point with the **rpmv** subroutine.

Graphics Library Overview, Setting Drawing Attributes, Creating Lighting Effects.

Drawing Rectangles, Circles, Arcs, and Polygons.

---

popattributes Subroutine

Purpose

Pops the attribute stack.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void popattributes( )
```

FORTRAN Syntax

```
SUBROUTINE POPATT
```

Description

The **popattributes** subroutine pops the attributes stack, restoring the values of the global state attributes most recently saved with the **pushattributes** subroutine:

- backbuffer enable (T or F)
- color map number (one of 16 *small* maps)
- colormode (Colormap or RGB)
- current color
- current font
- current linestyle
- current linestyle repeat factor
- current linewidth
- current pattern
- current writemask
- drawmode (overlay, underlay, or main buffers)
- draw_to_z_buffer enable (T or F)
- frontbuffer enable (T or F)
- logicop function
- shademodel (Flat or Gouraud)

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling drawing in the back buffer with the **backbuffer** subroutine.

Setting color map mode as the current mode with the **cmode** subroutine.

Setting the color index in the current mode with the **color** subroutine.

Specifying the target framebuffer of the drawing subroutines with the **drawmode** subroutine.

Enabling drawing in the front buffer with the **frontbuffer** subroutine.

Specifying the line width with the **linewidth** subroutine.

Setting the repeat factor for the current linestyle with the **lsrepeat** subroutine.

Pushing down the attribute stack with the **pushattributes** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Granting write access to a subset of available bitplanes in RGB mode with the **RGBwritemask** subroutine.

Selecting a linestyle pattern with the **setlinestyle** subroutine.

Selecting a pattern for filling polygons and rectangles with the **setpattern** subroutine.

Selecting the shading model used to draw a polygon with the **shademodel** subroutine.

Granting write permission to a subset of available bitplanes in color map mode with the **writemask** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

popmatrix Subroutine

Purpose
Pops the transformation matrix stack.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void popmatrix( )
```

FORTRAN Syntax
```
SUBROUTINE POPMAT
```

Description
The **popmatrix** subroutine pops the viewing transformation matrix stack.

This subroutine is not valid when the system is in projection matrix mode (`mmode(MPROJ);`), because the matrix stack is not accessible in this mode.

Example
The example C language program **backface.c** uses the **popmatrix** subroutine, to restore the previous transformation matrix after altering it with the **translate** modeling subroutine.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current matrix mode with the **mmode** subroutine.

Getting a copy of the current transformation matrix with the **getmatrix** subroutine.

Loading a transformation matrix with the **loadmatrix** subroutine.

Premultiplying the current transformation matrix with the **multmatrix** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

popname Subroutine

Purpose

Pops a name off the name stack.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void popname( )
```

FORTRAN Syntax

```
SUBROUTINE POPNAM
```

Description

The **popname** subroutine removes the top name from the name stack. It is used in both picking and selecting.

This subroutine is ignored outside of picking and selecting modes.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Putting the system in selecting mode with the **gselect** subroutine.

Initializing the name stack with the **initnames** subroutine.

Loading the name on top of the name stack with the **loadname** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Pushing a new name onto the name stack with the **pushname** subroutine.

Graphics Library Overview and Picking and Selecting Overview.

---

popviewport Subroutine

Purpose

Pops the viewport stack.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void popviewport( )
```

FORTRAN Syntax

```
SUBROUTINE POPVIE( )
```

Description

The **popviewport** subroutine pops the viewport stack, restoring the values of the viewport, screenmask, and depth range most recently saved with the **pushviewport** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the viewport depth range with the **lsetdepth** subroutine.

Pushing the viewport onto the viewport stack with the **pushviewport** subroutine.

Defining a rectangular 2-D clipping mask with the **scrmask** subroutine.

Setting the area of the window used for all drawing with the **viewport** subroutine.

Graphics Library Overview and Using Viewports and Screenmasks.

---

prefposition Subroutine

Purpose

Constrains the size of a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void prefposition
(Int32  x1, Int32  x2,
Int32  y1, Int32  y2)
```

FORTRAN Syntax

```
SUBROUTINE PREFPO(x1, x2, y1, y2)
INTEGER*4 x1, x2, y1, y2
```

Description

The **prefposition** subroutine sets the location and constrains the size, in pixels, of a window. With the **prefposition** subroutine, the applications programmer can prevent the user from resizing a window.

To remove constraints from a window, call the **winconstraints** subroutine immediately after calling the **winopen** subroutine. Calling the **winconstraints** subroutine twice in a row also removes the constraints from the window.

> **Note:** This process nullifies all constraints, not just that of size.

If the **prefposition** subroutine call is followed by a call to the **winopen** subroutine, the window manager creates and maps the window immediately. A rubberband outline is not shown.

If the **prefposition** subroutine call is followed by a call to the **winconstraints** subroutine, the current window is resized. The resizing occurs at the time of the **winconstraints** call; the **prefposition** subroutine has no effect on the current window until that time.

The window manager controls both how the **prefposition** subroutine parameters are interpreted and where a window is finally placed. For more information on the window manager operation, see the discussion on ″Window Placement″ in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

x1 Specifies the *x*-coordinate position (in pixels) of one corner of the window.
y1 Specifies the *y*-coordinate position (in pixels) of the point of one corner of the window.

x2 Specifies the *x*-coordinate position (in pixels) of the opposite corner of the window.
y2 Specifies the *y*-coordinate position (in pixels) of the opposite corner of the window.

Example

The example C language program **colored.c** uses the **prefposition** subroutine to specify a window's original location size.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Obtaining the position of a window with the **getorigin** subroutine.

Obtaining the size of a window with the **getsize** subroutine.

Constraining the size of a window with the **prefsize** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

prefsize Subroutine

Purpose

Constrains the size of a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void prefsize(Int32  x, Int32  y)
```

FORTRAN Syntax

```
SUBROUTINE PREFSI(x, y)
INTEGER*4 x, y
```

Description

The **prefsize** subroutine constrains the size, in pixels, of a window. With the **prefsize** subroutine, the applications programmer can prevent the user from resizing a window.

To remove constraints from a window, call the **winconstraints** subroutine immediately after calling the **winopen** subroutine. Calling **winconstraints** twice in a row also removes constraints from the window. Note that doing this nullifies all constraints, not just that of size.

If the **prefsize** subroutine call is followed by a call to the **winopen** subroutine, the window manager displays a window outline of the suggested size, allowing the user to position the window with the cursor.

If the **prefsize** subroutine call is followed by a call to the **winconstraints** subroutine, the current window is resized. The resizing occurs at the time of the **winconstraints** call; the **prefsize** subroutine has no effect on the current window until that time.

Note: This subroutine cannot be used to add to a display list.

Parameters

x Specifies the width of the window in pixels.
y Specifies the height of the window in pixels.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Obtaining the size of the window with the **getsize** subroutine.

Specifying the maximum size of a window with the **maxsize** subroutine.

Specifying the minimum size of a window with the **minsize** subroutine.

Constraining the window position and size with the **prefposition** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating a window with the **winopen** subroutine.

pushattributes Subroutine

Purpose
Pushes down the attribute stack.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void pushattributes( )
```

FORTRAN Syntax
```
SUBROUTINE PUSHAT
```

Description
The **pushattributes** subroutine pushes down the attribute stack. That is, the record at the top of the stack is duplicated and pushed onto the stack. The following attributes reside on the stack and are thus pushable and popable:

- backbuffer enable (T or F)
- color map number (one of 16 *small* maps)
- colormode (Colormap or RGB)
- current color
- current font
- current linestyle
- current linestyle repeat factor
- current linewidth
- current pattern
- current writemask
- drawmode (overlay, underlay, or main buffers)
- draw_to_z_buffer enable (T or F)
- frontbuffer enable (T or F)
- logicop function
- shademodel (Flat or Gouraud)

The saved values can be restored using the **popattributes** subroutine.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling drawing in the back buffer with the **backbuffer** subroutine.

Setting color map mode as the current mode with the **cmode** subroutine.

Setting the color index in the current mode with the **color** subroutine.

Specifying the target framebuffer of the drawing subroutines with the **drawmode** subroutine.

Enabling drawing in the front buffer with the **frontbuffer** subroutine.

Specifying the line width with the **linewidth** subroutine.

Setting the repeat factor for the current linestyle with the **lsrepeat** subroutine.

Popping the attribute stack with the **popattributes** subroutine.

Setting the current color in RGB mode with the **RGBcolor** subroutine.

Granting write access to a subset of available bitplanes in RGB mode with the **RGBwritemask** subroutine.

Selecting a linestyle pattern with the **setlinestyle** subroutine.

Selecting one of 16 small color maps with the **setmap** subroutine.

Selecting a pattern for filling polygons and rectangles with the **setpattern** subroutine.

Selecting the shading model used to draw a polygon with the **shademodel** subroutine.

Granting write permission to a subset of available bitplanes in color map mode with the **writemask** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

pushmatrix Subroutine

Purpose

Pushes down the transformation matrix stack.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void pushmatrix( )
```

FORTRAN Syntax

```
SUBROUTINE PUSHMA
```

Description

The **pushmatrix** subroutine pushes down the transformation matrix stack, duplicating the current matrix. For example, if the stack contains one matrix, *M,* after a call to the **pushmatrix** subroutine, the matrix contains two copies of *M.* The top copy can be modified.

This subroutine is not valid when the system is in projection matrix mode (`mmode(MPROJ);`), because the matrix stack is not accessible in this mode.

Example

The example C language program **backface.c** uses the **pushmatrix** subroutine to save the current transformation matrix before altering it with the **translate** modeling subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Changing the matrix mode with the **mmode** subroutine.

Getting a copy of the current transformation matrix with the **getmatrix** subroutine.

Loading a transformation matrix with the **loadmatrix** subroutine.

Premultiplying the current transformation matrix with the **multmatrix** subroutine.

Popping the transformation matrix stack with the **popmatrix** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

pushname Subroutine

Purpose

Pushes a new name onto the name stack.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void pushname(Int16  name)
```

FORTRAN Syntax

```
SUBROUTINE PUSHNA(name)
INTEGER*4 name
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **pushname** subroutine pushes the name stack down one level and puts a new 16-bit name on top.

The name stack must first have been initialized with the **initnames** subroutine. At least one name must have been loaded onto the stack with the **loadname** subroutine. The system stores the contents of the name stack in a buffer if a hit has occurred since the late time that the name stack was touched.

This subroutine is ignored outside of picking or selecting mode.

Parameter

name Specifies the name to add to the name stack.

Example

The example C language program **pick1.c** calls the **pushname** subroutine to push the name stack and put a new name on the top.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Putting the system in selecting mode with the **gselect** subroutine.

Initializing the name stack with the **initnames** subroutine.

Loading the name on top of the name stack with the **loadname** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Popping a name off name stack with the **popname** subroutine.

Graphics Library Overview and Picking and Selecting Overview.

pushviewport Subroutine

Purpose
Duplicates the current viewport.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void pushviewport( )
```

FORTRAN Syntax
```
SUBROUTINE PUSHVI( )
```

Description
The **pushviewport** subroutine pushes down the viewport stack, duplicating the current viewport, screenmask, and depth range. These saved values can be restored using the **popviewport** subroutine.

The viewport stack is VPSTACKDEPTH levels deep. The **pushviewport** subroutine is ignored if the stack is full. The VPSTACKDEPTH symbol is defined in the **/usr/include/gl/gl.h** file.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Setting the viewport depth range with the **lsetdepth** subroutine.

Popping the viewport stack with the **popviewport** subroutine.

Defining a rectangular 2-D clipping mask with the **scrmask** subroutine.

Setting the area of the window used for all drawing with the **viewport** subroutine.

Graphics Library Overview and Using Viewports and Screenmasks.

pwlcurve Subroutine

Purpose
Describes a piecewise linear trimming curve for NURBS surfaces.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void pwlcurve
(Int32  count,
Float64 * data_array,
Int32  stride,
Int32  type)
```

FORTRAN Syntax

```
SUBROUTINE PWLCUR(count, data_array, stride, type)
INTEGER*4 count, stride, type
REAL*8 data_array(*)
```

Description

The **pwlcurve** subroutine describes a piecewise linear curve, which consists of a list of coordinate pairs in the parameter space for a Non-Uniform Rational B-Spline (NURBS) surface. A piecewise linear curve can be used to describe a trimming loop. Use trimming loop definitions within surface definitions, as defined by the **bgnsurface** subroutine.

The trimming loops are closed curves that the system uses to set the boundaries of a NURBS surface. Describe a trimming loop by using a series of NURBS curves (as defined by the **nurbscurve** subroutine), piecewise linear curves, or both. 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.

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** subroutine. To mark the end of a trimming loop definition, use an **endtrim** subroutine.

The system displays the region of the NURBS surface that is to the left of the trimming curves as the parameter increases. Thus, for a counterclockwise-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.

Parameters

count	Specifies the number of points on the curve.
data_array	Specifies an array containing the curve points.
stride	Specifies the offset (in bytes) between points on the curve.
type	Specifies a value indicating the point type. Currently, the only data type supported is N_ST, corresponding to pairs of s-t coordinates. The *stride* parameter is used in case the curve points are part of an array of larger structure elements. The **pwlcurve** subroutine searches for the *count*-th coordinate pair beginning at *data_array* + *count* * *stride*.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Marking the beginning and end of a NURBS surface trimming loop with the **bgntrim** and **endtrim** subroutines.

Marking the beginning and end of a NURBS surface definition with the **bgnsurface** and **endsurface** subroutines.

Returning the current value of a trimmed NURBS surfaces display property with the **getnurbsproperty** subroutine.

Controlling the shape of a NURBS trimming curve with the **nurbscurve** subroutine.

Controlling the shape of a NURBS surface with the **nurbssurface** subroutine.

Setting a property for the display of trimmed NURBS with the **setnurbsproperty** subroutine.

Graphics Library Overview and Drawing NURBS Curves and Surfaces.

---

qdevice Subroutine

Purpose

Enables a input device (keyboard, button, or valuator) for event queuing.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void qdevice(Device  device)
```

FORTRAN Syntax

```
SUBROUTINE QDEVIC(device)
INTEGER*4 device
```

Description

The **qdevice** subroutine changes the state of the specified device so that events occurring within the device are entered in the event queue. The device can be the keyboard, a button, a valuator, or certain other pseudo-devices. The maximum number of queue entries is 50.

Note: This subroutine cannot be used to add to a display list.

Parameter

device Specifies the device whose state is to be changed so that it enters events into the event queue.

Example

The example C language program **scrn_rotate.c** uses the **qdevice** subroutine to enable input from various devices.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Filtering valuator motion with the **noise** subroutine.

Tying two valuators to a button with the **tie** subroutine.

Disabling an input device for event queuing with the **unqdevice** subroutine.

Graphics Library Overview, Controlling Queues and Devices and Using the Keyboard.

---

qenter Subroutine

Purpose

Creates an event queue entry.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void qenter(Int16  qtype, Int16  value)
```

FORTRAN Syntax

```
SUBROUTINE QENTER(qtype, value)
INTEGER*4 qtype, value
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when started from a FORTRAN program, although it accepts short integers when started from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **qenter** subroutine takes two 16-bit integers, the *dev* and *val* parameters, and enters them as an event in the event queue. There is no way to distinguish user-defined from system-defined entries unless disjointed sets of device numbers are used.

An event entered with the **qenter** subroutine cannot be distinguished from an event automatically generated by the system if the event type is equal to one of the predefined system event types. The system defines a large number of event types in the **gl/device.h** header file. In brief, types `0` through `0xfff` are predefined by the system; types `0x1000` through `0x7fff` are reserved for use by the user, while `0x8000` through `0xffff` are reserved for future use by the system. Devices in these ranges are defined by the **IsButton**, **IsValuator** and other macros in the **gl/device.h** header file.

Events entered with the **qenter** subroutine are *always* placed in the event queue. These events are never filtered, whether or not the event type falls into the range predefined by the system, or reserved for the user. In other words, it is not necessary to use the **qdevice** subroutine prior to using the **qenter** subroutine.

When the **qenter** subroutine is used from a signal handler to enter an event into the GL queue, and the GL application is blocked on the **qread** subroutine, the **qread** subroutine cannot unblock. The **qread** subroutine can be unblocked by using the **XSendEvent** subroutine within the signal handler to send some arbitrary event (preferably, a ClientMessage event) to a window belonging to the application.

Note: This subroutine cannot be used to add to a display list.

Parameters

qtype	Specifies a number indicating the device making the queue entry.
value	Specifies the value to be entered into the event queue.

Example

The example C language program **scrn_rotate.c** uses the **qenter** subroutine to enter a REDRAW device event into the queue.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Reading the first entry in the event queue with the **qread** subroutine.

Emptying the event queue with the **qreset** subroutine.

Checking the contents of the event queue with the **qtest** subroutine.

Graphics Library Overview and Controlling Queues and Devices.

---

qread Subroutine

Purpose
Reads the first entry in the event queue.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 qread(Int16 * data)
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION QREAD(data)
INTEGER*2 data
```

Description
The **qread** subroutine returns the first entry on the event queue. If the event queue is empty, the **qread** subroutine blocks. That is, this subroutine does not return until an event occurs. If the event queue is not empty, the **qread** subroutine returns the device number associated with the first event in the queue, writes the associated data into the *data* parameter, and removes that entry from the queue.

Note: This subroutine cannot be used to add to a display list.

Parameter
data Specifies a pointer to the variable that is to receive the data in the event queue.

Return Value
The identifier for the device read.

Example
The example C language program **scrn_rotate.c** uses the **qread** subroutine to read input from the event queue.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling an input device for event queuing with the **qdevice** subroutine.

Disabling an input device for event queuing with the **unqdevice** subroutine.

Graphics Library Overview and Controlling Queues and Devices.

---

qreset Subroutine

Purpose

Empties the event queue.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void qreset( )
```

FORTRAN Syntax

```
SUBROUTINE QRESET
```

Description

The **qreset** subroutine removes all entries from the event queue and discards them.

Note: This subroutine cannot be used to add to a display list.

Example

The example C language program **scrn_rotate.c** uses the **qreset** subroutine to delete all input events from any devices.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Creating an event queue entry with the **qenter** subroutine.

Reading the first entry in the event queue with the **qread** subroutine.

Check the content of the event queue with the **qtest** subroutine.

Graphics Library Overview and Controlling Queues and Devices.

---

qtest Subroutine

Purpose

Checks the contents of the event queue.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 qtest( )
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION QTEST
```

Description

The **qtest** subroutine returns zero if the event queue is empty. Otherwise, it returns the device number of the first entry. The queue remains unchanged.

Note: This subroutine cannot be used to add to a display list.

Return Value

The device number of the first entry (0 if the event queue is empty).

Example

The example C language program **scrn_rotate.c** uses the **qtest** subroutine to check if there is any input from the queued devices.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Creating an event queue entry with the **qenter** subroutine.

Reading the first entry in the event queue with the **qread** subroutine.

Emptying the event queue with the **qreset** subroutine.

Graphics Library Overview and Controlling Queues and Devices.

---

rcrv Subroutine

Purpose

Draws a rational cubic spline curve.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rcrv(Coord  geom[4][4])
```

FORTRAN Syntax

```
SUBROUTINE RCRV(geom)
REAL geom(4,4)
```

Description

The **rcrv** subroutine draws a rational cubic spline curve segment using the current curve basis and precision.

The *geom* parameter specifies the four control points of the curve segment.

Parameters

geom Specifies an array containing the four control points of the curve segment.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a cubic spline curve with the **crv** subroutine.

Drawing a series of curve segments with the **crvn** subroutine.

Setting the current cubic spline curve basis matrix for drawing curves with the **curvebasis** subroutine.

Setting the number of line segments that compose a cubic spline curve with the **curveprecision** subroutine.

Defining a cubic spline basis matrix with the **defbasis** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

rcrvn Subroutine

Purpose

Draws a series of rational curve segments.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rcrvn(Int32  n, Coord  geom [ ][4])
```

FORTRAN Syntax

```
SUBROUTINE RCRVN(n, geom)
INTEGER*4 n
REAL geom(4,n)
```

Description

The **rcrvn** subroutine draws a series of rational cubic spline curve segments using the current basis and precision.

The control points specified in the *geom* parameter determine the shapes of the curve segments and are used four at a time. The *n* parameter specifies the number of control points to be used in drawing the curve. For example, if *n* is 6, three curve segments are drawn:

1. Using points 0,1,2,3 as control points.
2. Using points 1,2,3,4 as control points.
3. Using points 2,3,4,5 as control points.

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.

Parameters

n	Specifies the number of control points to be used in drawing the curve.
geom	Specifies the matrix containing the control points of the curve segments.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a cubic spline curve with the **crv** subroutine.

Drawing a series of curve segments with the **crvn** subroutine.

Setting the current cubic spline curve basis matrix with the **curvebasis** subroutine.

Setting the number of line segments that compose a cubic spline curve with the **curveprecision** subroutine.

Defining a cubic spline basis matrix with the **defbasis** subroutine.

Drawing a rational curve with the **rcrv** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

---

rdr Subroutine

Purpose

Draws a line relative to the current graphics point.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rdr
(Coord  dx, Coord  dy, Coord  dz)

void rdri
(Icoord  dx, Icoord  dy, Icoord  dz)
```

```
void rdrs
(Scoord  dx, Scoord  dy, Scoord  dz)

void rdr2
(Coord  dx, Coord  dy)

void rdr2i
(Icoord  dx, Icoord  dy)

void rdr2s
(Scoord  dx, Scoord  dy)
```

FORTRAN Syntax

```
SUBROUTINE RDR(dx, dy, dz)
REAL dx, dy, dz

SUBROUTINE RDRI(dx, dy, dz)
INTEGER*4 dx, dy, dz

SUBROUTINE RDRS(dx, dy, dz)
INTEGER*2 dx, dy, dz

SUBROUTINE RDR2(dx, dy)
REAL dx, dy

SUBROUTINE RDR2I(dx, dy)
INTEGER*4 dx, dy

SUBROUTINE RDR2S(dx, dy)
INTEGER*2 dx, dy
```

Description

The **rdr** subroutine is the relative version of the **draw** subroutine. It connects the point *dx*, *dy*, *dz* and the current graphics position with a line segment, using the current line attributes: linestyle, linewidth, color (if in depth-cue mode, the depth-cued color is used), and writemask.

The **rdr** subroutine updates the current graphics position to the specified point.

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

The six different forms for the **rdr** subroutine are as follows:

	2-D	**3-D**
Int16	rdr2s	rdrs
Int32	rdr2i	rdri
float	rdr2	rdr

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **rdr** expects real coordinates, **rdri** expects integer coordinates, and **rdrs** expects short integer coordinates. In addition, the **rdr2*** routines assume a 2-D point instead of a 3-D point.

Parameters

dx Specifies the distance from the *x* coordinate of the current graphics position to the *x* coordinate of the new point.

dy Specifies the distance from the *y* coordinate of the current graphics position to the *y* coordinate of the new point.

dz Specifies the distance from the *z* coordinate of the current graphics position to the *z* coordinate of the new point.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a line with the **draw** subroutine.

Moving the current graphics position to a specified point with the **move** subroutine.

Drawing a point with the **pnt** subroutine.

Moving the current graphics position to a point relative to the current point with the **rmv** subroutine.

Graphics Library Overview, Drawing with Move-Draw Style Subroutines, Performing Depth-Cueing, Setting Drawing Attributes, and Working with Coordinate Systems.

---

readpixels Subroutine

Purpose

Returns values of specific pixels in color map mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 readpixels(Int16  number, Colorindex  colors[ ])
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION READPI(number, colors)
INTEGER*4 number
INTEGER*2 colors(number)
```

Description

The **readpixels** subroutine returns values of specific pixels from the frame buffer in color map mode. It reads them into the array starting from the current character position along a single scan line (constant y) in the direction of increasing x.

The *number* parameter returns the number of pixels read, which is the number requested if the starting point is a valid character position (inside the current viewport.

The system must be in color map mode for the **readpixels** subroutine to function. Use the **readRGB** subroutine to read pixels in RGB mode.

The **readpixels** subroutine 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. The subroutine 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. Use the **readsource** subroutine to specify which buffer is read.

When the system is in SINGLEMAP mode, only the lowest 12 bits contain valid data, and the 4 upper bits of a color value (an element of the array in the *colors* parameter) are undefined. When the system is in MULTIMAP mode, only the lowest 8 bits contain valid data, and the upper 8 bits of a color value are undefined.

The **rectread** subroutine provides significantly better performance for pixel block transfers. Even when only one row of pixels needs to be read, use the **rectread** subroutine.

1. This subroutine is available only in color map mode.
2. This subroutine cannot be used to add to a display list.
3. The use of this subroutine is deprecated. Do not use the **readpixels** subroutine in new development.

Parameters

number	Specifies the number of pixels to be read by the function.
colors	Specifies the array in which the pixel values are to be stored.

Return Value

The number of pixels actually read. A returned function value of 0 (zero) indicates that the starting point is not a valid character position.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the value of specific pixels in RGB mode with the **readRGB** subroutine.

Specifying the source for pixels to be read with the **readsource** subroutine.

Copying a rectangle of pixels with an optional zoom with the **rectcopy** subroutine.

Reading a rectangular array of pixels into host memory with the **rectread** subroutine.

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** subroutine.

Painting a row of pixels on the screen in color map mode with the **writepixels** subroutine.

Graphics Library Overview, Reading and Writing Pixels, Using Viewports and Screenmasks, and Working in Color Map and RGB Modes.

---

readRGB Subroutine

Purpose

Returns values of specific pixels in RGB mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 readRGB
(Int16  number,
RGBvalue  red[ ], RGBvalue  green[ ], RGBvalue  blue[ ])
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION READRG(number, red, green, blue)
INTEGER*4 number
CHARACTER*(*) red, green, blue;
```

> **Note:** For FORTRAN users, this function accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **readRGB** subroutine attempts to read specific pixel values from the frame buffer in RGB mode. The returned value of this function is the number of pixels actually read. A returned function value of 0 (zero) indicates that the starting point is not a valid character position.

The **readRGB** subroutine reads the pixel values into the arrays specified by the *red*, *green*, and *blue* parameters starting from the current character position along a single scan line (constant y) in the direction of increasing x.

The **readRGB** subroutine returns the number of pixels read, which is the number requested if the starting point is a valid character position (inside the current viewport). The subroutine returns 0 (zero) if the starting point is not a valid character position. The values of pixels read outside of the viewport or screenmask are undefined.

The **readRGB** subroutine 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.

Use the **readsource** subroutine to specify which buffer is read. In RGB double buffer mode, by default the back buffer is read.

The **rectread** subroutine provides significantly better performance for pixel block transfers. Even when only one row of pixels needs to be read, use the **rectread** subroutine.

1. This subroutine is available only in RGB mode. For new development, use the **lrectread** subroutine.
2. This subroutine cannot be used to add to a display list.
3. The use of this subroutine is deprecated. Do not use the **readRGB** subroutine in new development.

Parameters

number	Specifies the number of pixels read by the function.
red	Specifies the array in which the red pixel values will be stored.
green	Specifies the array in which the green pixel values will be stored.
blue	Specifies the array in which the blue pixel values will be stored.

Return Value

The returned value of this function is the number of pixels that the system actually reads. The returned function value is 0 (zero) if any part of the specified rectangle is off the screen.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the value of specific pixels in color map mode with the **readpixels** subroutine.

Specifying the source for pixels to be read with the **readsource** subroutine.

Copying a rectangle of pixels with an optional zoom with the **rectcopy** subroutine.

Reading a rectangular array of pixels into host memory with the **rectread** subroutine.

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** subroutine.

Painting a row of pixels on the screen in RGB mode with the **writeRGB** subroutine.

Graphics Library Overview, Configuring the Frame Buffer, Creating Animated Scenes, Reading and Writing Pixels, Using Viewports and Screenmasks, and Working in Color Map and RGB Modes.

---

readsource Subroutine

Purpose

Specifies the source for pixels to be read by various subroutines.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void readsource(Int32  source)
```

FORTRAN Syntax

```
SUBROUTINE READSO(source)
INTEGER*4 source
```

Description

The **readsource** subroutine specifies the exact pixel source (front buffer, back buffer, or z-buffer) that the **rectcopy**, **readpixels**, **readRGB**, and **rectread** subroutines use.

Note: This subroutine cannot be used to add to a display list.

Parameter

source The defined values for this parameter are listed in the following table:

Values for the source Parameter		
C	**FORTRAN**	**Function**
SRC_AUTO	SRCAUT	Selects the front buffer in single buffer mode and the back buffer in double buffer mode.
SRC_FRONT	SRCFRO	Front color buffer. Valid for both single and double buffer operation.
SRC_BACK	SRCBAC	Back color buffer. Valid during double buffer operation only.
SRC_ZBUFFER	SRCZBU	Z-buffer. Valid during both single and double buffer operation.
SRC_OVER	SRCOVE	Overlay planes.

Note: The 3-D Color Graphics Processor does not support SRC_OVER.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

BLITs to and from the z-buffer are currently inoperational on the POWER GXT1000 adapter.

When performing BLITs to and from the overlay planes on the GXT1000, the transparent pixel value on the GXT1000 is 255, not 0.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.

/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Copying a rectangle of pixels with an optional zoom with the **rectcopy** subroutine.

Reading a rectangular array of pixels into host memory with the **rectread** subroutine.

Graphics Library Overview, Reading and Writing Pixels, and Configuring the Frame Buffer.

---

rect Subroutine

Purpose

Draws an unfilled rectangle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rect
(Coord  x1, Coord  y1,
Coord  x2, Coord  y2)

void recti
(Icoord  x1, Icoord  y1,
Icoord  x2, Icoord  y2)

void rects
(Scoord  x1, Scoord  y1,
Scoord  x2, Scoord  y2)
```

FORTRAN Syntax

```
SUBROUTINE RECT(x1, y1, x2, y2)
REAL x1, y1, x2, y2
SUBROUTINE RECTI(x1, y1, x2, y2)
INTEGER*4 x1, y1, x2, y2
SUBROUTINE RECTS(x1, y1, x2, y2)
INTEGER*2 x1, y1, x2, y2
```

Note: For FORTRAN users, the INTEGER*2 version, the **RECTS** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

Description

The **rect** subroutine draws a rectangle using the current line attributes: linestyle, linewidth, color, and writemask. The sides of the rectangle are parallel to the *x* and *y* axes. Since a rectangle is a 2-D shape, the **rect** subroutine takes only 2-D arguments, and sets the *z* coordinate to zero. The current graphics position is set to (*x1, y1*) after the rectangle is drawn.

The syntax for each of the subroutine forms is the same except for the first argument. They differ only in that **rect** expects real coordinates, **recti** expects integer coordinates, and **rects** expects short integer coordinates.

Parameters

x1 Specifies the *x* coordinate of one corner of the rectangle.
y1 Specifies the *y* coordinate of one corner of the rectangle.
x2 Specifies the *x* coordinate of the opposite corner of the rectangle.
y2 Specifies the *y* coordinate of the opposite corner of the rectangle.

Example

The example C language program **tpbig.c** uses the **recti** subroutine to draw the outline of a rectangle in green.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a polygon with the **poly** subroutine.

Drawing a filled rectangle with the **rectf** subroutine.

Graphics Library Overview, Setting Drawing Attributes, Drawing Rectangles, Circles, Arcs, and Polygons.

---

rectcopy Subroutine

Purpose

Copies a rectangle of pixels with an optional zoom.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rectcopy
(Screencoord  xll, Screencoord  yll,
Screencoord  xur, Screencoord  yur,
Screencoord  newx, Screencoord  newy)
```

FORTRAN Syntax

```
SUBROUTINE RECTCO(xll, yll, xur, yur, newx, newy)
INTEGER*4 xll, yll, xur, yur, newx, newy
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **rectcopy** subroutine copies a rectangular array of pixels 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 the **rectzoom** subroutine to zoom the destination independently in both the x and y directions. Self-intersecting copies also work correctly with zooming.

Use the **readsource** subroutine to specify the front buffer, the back buffer, or the z-buffer as the source.

On machines that support it, you can use the **rectzoom** subroutine to zoom the destination independently in both the x and y directions. Self-intersecting copies also work correctly with zooming.

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 controller cannot correctly interpret it.

Use the **frontbuffer**, **backbuffer**, and **zdraw** subroutines to specify the destination.

All coordinates are relative to the lower left corner of the window, not the screen or viewport.

The **rectcopy** subroutine leaves the current character position unpredictable. The result of the **rectcopy** subroutine is undefined if the value of the *bool* parameter in the **zbuffer** subroutine is TRUE.

Note: This subroutine cannot be used to add to a display list.

Parameters

xll	Specifies the *x* coordinate of one corner of the rectangle.
yll	Specifies the *y* coordinate of one corner of the rectangle.
xur	Specifies the *x* coordinate of the opposite corner of the rectangle.
yur	Specifies the *y* coordinate of the opposite corner of the rectangle.
newx	Specifies the *x* coordinate of the lower left corner of the new position of the rectangle.
newy	Specifies the *y* coordinate of the lower left corner of the new position of the rectangle.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the source for pixels to be read with the **readsource** subroutine.

Reading a rectangular array of pixels into host memory with the **rectread** or **lrectread** subroutine.

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** subroutine.

Specifying a zoom factor for rectangle copies and writes with the **rectzoom** subroutine.

Graphics Library Overview and Reading and Writing Pixels.

---

rectf Subroutine

Purpose

Draws a filled rectangle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rectf
(Coord  x1, Coord  y1,
Coord  x2, Coord  y2)

void rectfi
(Icoord  x1, Icoord  y1,
Icoord  x2, Icoord  y2)

void rectfs
(Scoord  x1, Scoord  y1,
Scoord  x2, Scoord  y2)
```

FORTRAN Syntax

```
SUBROUTINE RECTF(x1, y1, x2, y2)
REAL x1, y1, x2, y2
SUBROUTINE RECTFI(x1, y1, x2, y2)
INTEGER*4 x1, y1, x2, y2
SUBROUTINE RECTFS(x1, y1, x2, y2)
INTEGER*2 x1, y1, x2, y2
```

Note: For FORTRAN users, the INTEGER*2 version, the **RECTFS** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long

integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

Description

The **rectf** subroutine produces a filled rectangular region, using the current area attributes: pattern, color, and writemask. The sides of the rectangle are parallel to the *x* and *y* axes of the object coordinate system.

Since a rectangle is a 2-D shape, the **rectf** subroutine takes only 2-D arguments and sets the coordinate to zero. The current graphics position is set to (*x1, y1*) after the region is drawn. Backfacing polygon removal works correctly if (*x1, y1*) specifies the lower left corner and (*x2, y2*) the upper right corner of the rectangle.

The syntax for each of the subroutine forms is the same except for the first argument. They differ only in that **rectf** expects real coordinates, **rectfi** expects integer coordinates, and **rectfs** expects short integer coordinates.

Parameters

x1 Specifies the *x* coordinate of one corner of the rectangle that is to be drawn.
y1 Specifies the *y* coordinate of one corner of the rectangle that is to be drawn.
x2 Specifies the *x* coordinate of the opposite corner of the rectangle that is to be drawn.
y2 Specifies the *y* coordinate of the opposite corner of the rectangle that is to be drawn.

Example

The example C language program **tpbig.c** uses the **rectfi** subroutine to draw a filled, red rectangle.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a filled polygon with the **polf** subroutine.

Drawing a rectangle with the **rect** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

rectread or lrectread Subroutine

Purpose

Reads a rectangular array of pixels into host memory.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 rectread
(Screencoord  xll, Screencoord  yll,
Screencoord  xur, Screencoord  yur,
Int16 * parray)

Int32 lrectread
(Screencoord  xll, Screencoord  yll,
Screencoord  xur, Screencoord  yur,
Int32 * parray)
```

FORTRAN Syntax

```
INTEGER*4 RECTRE(xll, yll, xur, yur, parray)
INTEGER*4 xll, yll, xur, yur
INTEGER*2 parray(1)

INTEGER*4 LRECTR(xll, yll, xur, yur, parray)
INTEGER*4 xll, yll, xur, yur, parray(1)
```

> **Note:** For FORTRAN users, these subroutines accept long integer parameters (INTEGER*4) when started from a FORTRAN program, although they accept short integers when started from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **rectread** and **lrectread** subroutines each return a rectangular array of 16- and 32-bit pixels, respectively, to the host array specified in the *parray* parameter. For the **lrectread** subroutine, the *parray* parameter contains 32-bit packed RGB, RGBA, or *z* values. The returned value of this function is the number of pixels that the system actually reads, left to right, then bottom to top. The returned function value is 0 (zero) if any part of the specified rectangle is off the screen.

The returned data is undefined if the *xll* and *yll* parameters do not specify the lower-left corner of a rectangle that appears completely on the screen. All coordinates are relative to the lower left corner of the window and not to the screen or viewport.

When performing operations with the **lrectread** subroutine on the POWER Gt4 and POWER Gt4x in RGBmode with a specified pixel size of 32, pixels are returned with one 24-bit pixel packed in a 32-bit word. The high-order byte (the A byte) is set to the value 0xff.

The **readsource** subroutine specifies the pixel source from which the pixels are read.

The behavior of the **lrectread** subroutine is modified by the attributes set with the **pixmode** subroutine. Please refer to the **pixmode** subroutine for more detail.

1. Both the **rectread** and **lrectread** subroutines leave the current character position unpredictable.
2. Do not put the pixel array on the stack. Allocate the memory space for the pixel array dynamically, by using the **malloc** or **calloc** command (that is, place it on the heap), or specify it as a static variable. Do not specify the pixel array memory space as a block scope variable.
3. These subroutines cannot be used to add to a display list.

Parameters

xll	Specifies the *x* coordinate of the lower-left corner of the rectangle to be read.
yll	Specifies the *y* coordinate of the lower-left corner of the rectangle to be read.
xur	Specifies the *x* coordinate of the upper-right corner of the rectangle to be read.
yur	Specifies the *y* coordinate of the upper-right corner of the rectangle to be read.
parray	Specifies the array to receive the pixels that are read. The returned data in the *parray* parameter are undefined if the *xll* and *yll* parameters do not specify the lower-left corner of a rectangle that appears completely on the screen.

All coordinates are relative to the lower-left corner of the window and not to the screen or viewport.

Return Value

The returned value of this function is the number of pixels that the system actually reads. The returned function value is 0 (zero) if any part of the specified rectangle is off the screen.

On the POWER GXT1000 adapter, the returned value of this function is always the width times height of the specified rectangle, or (*xur* - *xll*)(*yur* - *yll*).

Example

The example C language program **paint.c** uses the **rectread** subroutine to get the color index value for a single pixel.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the source for pixels to be read with the **readsource** subroutine.

Controlling the transfer of pixels with the **pixmode** subroutine.

Copying a rectangle of pixels with an optional zoom with the **rectcopy** subroutine.

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** or **lrectwrite** subroutine.

Graphics Library Overview, Reading and Writing Pixels, Using Viewports and Screenmasks, and Working in Color Map and RGB Modes.

---

rectwrite or lrectwrite Subroutine

Purpose

Draws a rectangular array of pixels into the frame buffer.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rectwrite
(Screencoord  xll, Screencoord  yll,
Screencoord  xur, Screencoord  yur,
Int16 * parray)

void lrectwrite
(Screencoord  xll, Screencoord  yll,
Screencoord  xur, Screencoord  yur,
Int32 * parray)
```

FORTRAN Syntax

```
SUBROUTINE RECTWR(xll, yll, xur, yur, parray)
INTEGER*4 xll, yll, xur, yur
INTEGER*2 parray(1 )

SUBROUTINE LRECTW(xll, yll, xur, yur, parray)
INTEGER*4 xll, yll, xur, yur, parray( 1)
```

> **Note:** For FORTRAN users, these subroutines accept long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although they accept short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **rectwrite** and **lrectwrite** subroutines draw pixels taken from the host array specified in the *parray* parameter into the specified rectangular framebuffer region. Both procedures are functionally the same. They differ only in that the **rectwrite** subroutine expects 16-bit values, and the **lrectwrite** subroutine expects 32-bit values. The system draws pixels left to right, then bottom to top. All normal drawing modes apply.

When the frame buffer is configured to be 8 bits deep, only the lowest 8 bits of the *parray* parameter are used to fill the frame buffer. If the frame buffer has been configured to be 12 bits deep (for example, if the system is in color map, singlemap, doublebuffer mode), only the lowest 12 bits are written. If the frame buffer has been configured to be 24 bits deep (for instance, in singlebuffer RGB mode), only the lowest 24 bits are written.

Because of the foregoing reasons, using the **rectwrite** subroutine in 24-bit mode is not logical. Use the **lrectwrite** subroutine instead. Likewise, do not use the **rectwrite** subroutine to write into the z-buffer.

In a similar manner, these subroutines can be used to write into the overlay or underlay planes. Note that not all supported graphics adapters have 24-bit-deep frame buffers or have z-buffers.

> **Note:** Both the **rectwrite** and **lrectwrite** subroutines leave the current character position unpredictable.

If the zoom factors set by the **rectzoom** subroutine are both 1.0, the screen region *xll* through *xur*, *yll* through *yur*, are filled. Other zoom factors result in filling past *xur* and/or past *yur* (*xll, yll* is always the lower left corner of the filled region).

The behavior of the **lrectwrite** subroutine is modified by the attributes set with the **pixmode** subroutine. Please refer to the **pixmode** subroutine for more detail.

1. Do not put the pixel array on the stack. Allocate the memory space for the pixel array dynamically, by using the **malloc** or **calloc** command (that is, place it on the heap), or specify it as a static variable. Do not specify the pixel array memory space as a block scope variable.
2. These subroutines cannot be used to add to a display list.

Parameters

xll	Specifies the *x* coordinate of the lower-left corner of the rectangular frame-buffer region.
yll	Specifies the *y* coordinate of the lower-left corner of the rectangular frame-buffer region.
xur	Specifies the *x* coordinate of the upper-right corner of the rectangular frame-buffer region.
yur	Specifies the *y* coordinate of the upper-right corner of the rectangular frame-buffer region.
parray	Specifies the array containing the values of the pixels to be drawn. The size of *parray* is always (xur-xll+1) x (yur-yll+1). The format of the data in the *parray* parameter is controlled by the current setting of the **pixmode** subroutine. The returned data is undefined if the *xll* and *yll* parameters do not specify the lower left corner of a rectangle that appears completely on the screen. All coordinates are relative to the lower left corner of the window, not the screen or viewport.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Copying a rectangle of pixels with an optional zoom with the **rectcopy** subroutine.

Controlling the transfer of pixels with the **pixmode** subroutine.

Reading a rectangular array of pixels into host memory with the **rectread** or **lrectread** subroutine.

Specifying a zoom factor for rectangle copies and writes with the **rectzoom** subroutine.

Graphics Library Overview, Reading and Writing Pixels, Configuring the Frame Buffer, and Working in Color Map and RGB Modes.

---

rectzoom Subroutine

Purpose

Specifies a zoom factor for rectangle copies and writes.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rectzoom(Float32  xfactor, Float32  yfactor)
```

FORTRAN Syntax

```
SUBROUTINE RECTZO(xfactor, yfactor)
REAL xfactor, yfactor
```

Description

The **rectzoom** subroutine specifies independent x and y *zoom factors* that the **rectcopy** and **rectwrite** subroutines use. Float values are rounded to the nearest integer.

The default value for the *xfactor* and *yfactor* parameters is 1.0.

Note: This subroutine cannot be used to add to a display list.

Parameters

xfactor	Specifies the multiplier of the rectangle in the x direction.
yfactor	Specifies the multiplier of the rectangle in the y direction.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Copying a rectangle of pixels with an optional zoom with the **rectcopy** subroutine.

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** subroutine.

Graphics Library Overview and Reading and Writing Pixels.

---

reshapeviewport Subroutine

Purpose

Sets the viewport to the dimensions of the current window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void reshapeviewport( )
```

FORTRAN Syntax

```
SUBROUTINE RESHAP( )
```

Description

The **reshapeviewport** subroutine sets the viewport to the dimensions of the current window.

The **reshapeviewport** subroutine is equivalent to:

```
long xsize, ysize;
getsize(&xsize, &ysize);
viewport(0, xsize-1, 0, ysize-1);
```

Use the **reshapeviewport** subroutine when REDRAW events are received. It is most useful in programs that are independent of the size and shape of the viewport.

Note: This subroutine cannot be used to add to a display list.

Example

The example C language program **scrn_rotate.c** uses the **reshapeviewport** subroutine to reshape the viewport on a REDRAW event.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Clearing the viewport with the **clear** subroutine.

Returning the position of a window with the **getorigin** subroutine.

Returning the size of a window with the **getsize** subroutine.

Graphics Library Overview and Using Viewports and Screenmasks.

RGBcolor Subroutine

Purpose

Sets the current color in RGB mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
RGBcolor
(short  red, short  green, short  blue)
```

FORTRAN Syntax

```
SUBROUTINE RGBCOL(red, green, blue)
INTEGER*4 red, green, blue
```

Description

The **RGBcolor** subroutine sets the current color when the system is in RGB mode. The lower-order 8 bits of the three arguments control the intensity of red, green, and blue colors displayed on the screen. The system writes these numbers directly into the bitplanes whenever it draws a pixel.

Note: This subroutine is available only in RGB mode.

Parameters

red	Specifies a value indicating the intensity of the color red to be displayed on the screen.
green	Specifies a value indicating the intensity of the color green to be displayed on the screen.
blue	Specifies a value indicating the intensity of the color blue to be displayed on the screen.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current color in RGB mode with the **c** subroutine.

Setting the current color in color map mode with the **color** subroutine.

Setting the current color as a packed 32-bit integer with the **cpack** subroutine.

Returning the current RGB value with the **gRGBcolor** subroutine.

Setting Drawing Attributes, Understanding the Hardware Used by GL,Working in Color Map and RGB Modes.

RGBmode Subroutine

Purpose

Sets a display mode that bypasses the color map.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void RGBmode( )
```

FORTRAN Syntax

```
SUBROUTINE RGBMOD
```

Description

The **RGBmode** subroutine makes the system interpret the contents of the main frame buffer as RGB values. All drawing subroutines write values of red, green, and blue directly into the bitplanes; the contents of the frame buffer, in turn, directly control the intensity of the color displayed on the monitor.

When the frame buffer is 24 bits deep, 8 bits each of red, green, and blue are stored. When the frame buffer is 12 bits deep, the 4 most significant bits of each of the red, green, and blue are stored. When the frame buffer is 8 bits deep, the 3 most significant bits of each of the red and green, and the 2 most significant bits of blue are stored. To improve the visual quality of the displayed image, dithering is automatically turned on whenever the frame buffer is 8 bits deep.

The depth of the frame buffer depends on the installed adapter, and on how it has been configured. In particular, the **doublebuffer** and **singlebuffer** subroutines affect the configuration of the main frame buffer.

The system will not enter into RGB mode until the **gconfig** subroutine is called.

> **Note:** This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The **RGBmode** subroutine is useful for monitors with 12 or more bitplanes.

The POWERgraphics GXT1000 will ignore the **RGBmode** subroutine when applied to windows created with the **winX** subroutine. The **RGBmode** subroutine can be used with windows created with the **winopen** subroutine.

Example

The example C language program **localatten.c** puts the system in RGB mode with the **RGBmode** subroutine to perform lighting calculations. This lets the system calculate the correct colors for realistic shading.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting color map mode as the current mode with the **cmode** subroutine.

Reconfiguring the system with the **gconfig** subroutine.

Returning the current display mode with the **getdisplaymode** subroutine.

Working in Color Map and RGB Modes.

RGBwritemask Subroutine

Purpose

Grants write access to a subset of available bitplanes.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void RGBwritemask
(Int16  red,
Int16  green,
Int16  blue)
```

FORTRAN Syntax

```
SUBROUTINE RGBWRI(red, green, blue)
INTEGER*4 red, green, blue
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **RGBwritemask** subroutine sets the bitplane writemask in RGB mode. Writemasks are used to shield portions of the frame buffer from being written into. A writemask is a small set of bits (3 masks of 8 bits each in RGB mode), one bit for each bitplane of the frame buffer.

If a bit is set (1), the corresponding bitplane is enabled for writing, and any routine that draws into the frame buffer will be able to write into that bitplane. If a bit is reset (0), the corresponding bitplane is marked as read only, and the values stored in that bitplane are not changed.

Note that the writemask protects planes in the color frame buffer. Thus, writemasks essentially prevent certain colors from being written into the frame buffer. Colors that are drawn while a writemask is enabled appear different, depending on the color, the writemask, and the color value currently stored in the frame buffer (at a given pixel).

Writemasks are useful for emulating overlay/underlay planes.

The **RGBwritemask** subroutine is intended for use in RGB mode only. To set the writemask in color map mode, use the **writemask** or **wmpack** subroutine.

Parameters

red	Specifies the mask for the corresponding red bitplanes.
green	Specifies the mask for the corresponding green bitplanes.
blue	Specifies the mask for the corresponding blue bitplanes.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER GXT1000 adapter supports only on/off control of individual R, G, and B bitplanes. Individual bits cannot be masked in the color planes.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current RGB value with the **gRGBcolor** subroutine.

Returning the current RGB writemask with the **gRGBmask** subroutine.

Specifying the RGBA writemask with a single packed integer with the **wmpack** subroutine.

Granting write permission to available bitplanes with the **writemask** subroutine.

Configuring the Frame Buffer.

Writemasks and Logical Operations and Working in Color Map and RGB Modes.

---

ringbell Subroutine

Purpose

Rings the keyboard bell.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void ringbell( )
```

FORTRAN Syntax

```
SUBROUTINE RINGBE
```

Description

The **ringbell** subroutine rings the keyboard bell for the length of time set by the **setbell** subroutine.

Note: This subroutine cannot be used to add to a display list.

Example

The example C language program **ovrlay.c** uses the **ringbell** subroutine to ring the bell as a ball moves.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning off the keyboard click with the **clkoff** subroutine.

Turning on the keyboard click with the **clkon** subroutine.

Turning on the keyboard display lights with the **lampon** subroutine.

Setting the duration of the keyboard bell sound with the **setbell** subroutine.

Graphics Library Overview and Using the Keyboard.

---

rmv Subroutine

Purpose

Moves the graphics position relative to the current point.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rmv
(Coord  dx, Coord  dy, Coord  dz)

void rmvi
(Icoord  dx, Icoord  dy, Icoord  dz)
```

```
void rmvs
(Scoord  dx, Scoord  dy, Scoord  dz)

void rmv2
(Coord  dx, Coord  dy)

void rmv2i
(Icoord  dx, Icoord  dy)

void rmv2s
(Scoord  dx, Scoord  dy)
```

FORTRAN Syntax

```
SUBROUTINE RMV(dx, dy, dz)
REAL dx, dy, dz

SUBROUTINE RMVI(dx, dy, dz)
INTEGER*4 dx, dy, dz

SUBROUTINE RMVS(dx, dy, dz)
INTEGER*2 dx, dy, dz

SUBROUTINE RMV2(dx, dy)
REAL dx, dy

SUBROUTINE RMV2I(dx, dy)
INTEGER*4 dx, dy

SUBROUTINE RMV2S(dx, dy)
INTEGER*2 dx, dy
```

Description

The **rmv** subroutine is the relative version of the **move** subroutine. It moves the graphics position (without drawing) the specified amount relative to its current value. The value of **rmv2(***x***,** *y***)** is equivalent to **rmv(***x***,** *y***,** 0.0**)**.

The six different forms for the **rmv** subroutine are as follows:

	2-D	**3-D**
Int16	rmv2s	rmvs
Int32	rmv2i	rmvi
float	rmv2	rmv

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **rmv** expects real coordinates, **rmvi** expects integer coordinates, and **rmvs** expects short integer coordinates. In addition, the **rmv2*** routines assume a 2-D point instead of a 3-D point.

Parameters

dx Specifies the distance from the *x* coordinate of the current graphics position to the *x* coordinate of the new point.

dy Specifies the distance from the *y* coordinate of the current graphics position to the *y* coordinate of the new point.

dz Specifies the distance from the *z* coordinate of the current graphics position to the *z* coordinate of the new point.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a line with the **draw** subroutine.

Moving the current graphics position to a specified point with the **move** subroutine.

Drawing a relative line with the **rdr** subroutine.

Graphics Library Overview and Drawing with Move-Draw Style Subroutines.

---

rot Subroutine

Purpose

Rotates graphical primitives (floating-point version).

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rot(Float 32  angle, Char8  axis)
```

FORTRAN Syntax

```
SUBROUTINE ROT(angle, axis)
REAL angle
CHARACTER*1 axis
```

Description

The **rot** subroutine specifies an angle (*angle*) and an axis of rotation (*axis*). The floating point angle is given in degrees according to the right-hand rule.

The **rot** subroutine is a modeling routine; it changes the current transformation matrix. All objects drawn after the **rot** subroutine executes are rotated. Use the **pushmatrix** and **popmatrix** subroutines to preserve and restore unrotated modeling coordinates.

Parameters

angle	Specifies the angle of rotation of an object.
axis	Specifies the relative axis of rotation. There are three values defined for this parameter (the character may be upper- or lowercase):

x, X indicates the x-axis.
y, Y indicates the y-axis.
z, Z indicates the z-axis.

Example

The example C language program **cylinder1.c** uses the **rot** subroutine to rotate a cylinder about the y- and z-axes.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Popping the transformation matrix stack with the **popmatrix** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Rotating a graphical primitive (fixed-point version) with the **rotate** subroutine.

Scaling and mirroring objects with the **scale** subroutine.

Translating a graphical primitive with the **translate** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

rotate Subroutine

Purpose

Rotates graphical primitives (fixed-point version).

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rotate(Angle  angle, Char8  axis)
```

FORTRAN Syntax

```
SUBROUTINE ROTATE(angle, axis)
INTEGER*4 angle
CHARACTER*1 axis
```

Description

The **rotate** subroutine specifies an angle (*angle*) and an axis of rotation (*axis*). The fixed- point angle is given in tenths of a degree according to the right-hand rule.

The **rotate** subroutine is a modeling routine; it changes the current transformation matrix. All objects drawn after the **rotate** subroutine executes are rotated. Use the **pushmatrix** and **popmatrix** subroutines to preserve and restore unrotated modeling coordinates.

Parameters

angle Specifies the angle of rotation of an object.

axis Specifies the relative axis of rotation. There are six values defined for this parameter (the character may be upper- or lowercase):
x, **X** indicates the x-axis.
y, **Y** indicates the y-axis.
z, **Z** indicates the z-axis.

Example

The example C language program **backface.c** uses the **rotate** modeling subroutine to alter the current transformation matrix when modelling the sides of a cube using a square.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Popping the transformation matrix stack with the **popmatrix** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Rotating a graphical primitive (floating-point version) with the **rot** subroutine.

Scaling and mirroring objects with the **scale** subroutine.

Translating a graphical primitive with the **translate** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

rpatch Subroutine

Purpose

Draws a rational surface patch.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rpatch
(Matrix  geomx, Matrix  geomy,
Matrix  geomz, Matrix  geomw)
```

FORTRAN Syntax

```
SUBROUTINE RPATCH(geomx, geomy, geomz, geomw)
REAL geomx(4,4), geomy(4,4), geomz(4,4), geomw(4,4)
```

Description

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

Parameters

geomx	Specifies a 4x4 matrix containing the *x* coordinates of the 16 control points of the patch.
geomy	Specifies a 4x4 matrix containing the *y* coordinates of the 16 control points of the patch.
geomz	Specifies a 4x4 matrix containing the *z* coordinates of the 16 control points of the patch.
geomw	Specifies a 4x4 matrix containing the *w* coordinates of the 16 control points of the patch.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a cubic spline basis matrix with the **defbasis** subroutine.

Drawing a surface patch with the **patch** subroutine.

Setting the current spline surface basis matrices with the **patchbasis** subroutine.

Setting the number of curves used to represent a patch with the **patchcurves** subroutine.

Setting the precision at which curves are drawn with the **patchprecision** subroutine.

Graphics Library Overview, Drawing NURBS Curves and Surfaces, and Drawing Wire Frame Curves and Surface Patches.

rpdr Subroutine

Purpose
Performs a relative polygon draw.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void rpdr
(Coord  dx, Coord  dy, Coord  dz)

void rpdri
(Icoord  dx, Icoord  dy, Icoord  dz)

void rpdrs
(Scoord  dx, Scoord  dy, Scoord  dz)

void rpdr2
(Coord  dx, Coord  dy)

void rpdr2i
(Icoord  dx, Icoord  dy)

void rpdr2s
(Scoord  dx, Scoord  dy)
```

FORTRAN Syntax
```
SUBROUTINE RPDR(dx, dy, dz)
REAL dx, dy, dz
SUBROUTINE RPDRI(dx, dy, dz)
INTEGER*4 dx, dy, dz
SUBROUTINE RPDRS(dx, dy, dz)
INTEGER*2 dx, dy, dz
SUBROUTINE RPDR2(dx, dy)
REAL dx, dy
SUBROUTINE RPDR2I(dx, dy)
INTEGER*4 dx, dy
SUBROUTINE RPDR2S(dx, dy)
INTEGER*2 dx, dy
```

Note: For FORTRAN users, the INTEGER*2 versions of this subroutine, **RPDRS** and **RPDR2S**, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking one of the short versions of this subroutine with an integer constant will result in unexpected behavior.

Description

The **rpdr** subroutine is the relative version of the **pdr** subroutine. It specifies the next point in a filled polygon, using the previous point (the current graphics position) as the origin.

There can be no more than 256 vertices in a polygon. Therefore, there can be no more than 255 calls to the **rpdr** subroutine between calls to the **rpmv** and **pclos** subroutines.

The **rpdr** subroutine updates the current graphics position. The next routine starts drawing from that point.

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

The six different forms for the **rpdr** subroutine are as follows:

	2-D	**3-D**
Int16	rpdr2s	rpdrs
Int32	rpdr2i	rpdri
float	rpdr2	rpdr

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **rpdr** expects real coordinates, **rpdri** expects integer coordinates, and **rpdrs** expects short integer coordinates. In addition, the **rpdr2** routines assume a 2-D point instead of a 3-D point.

Parameters

dx Specifies the distance from the *x* coordinate of the current graphics position to the *x* coordinate of the next corner of the polygon.

dy Specifies the distance from the *y* coordinate of the current graphics position to the *y* coordinate of the next corner of the polygon.

dz Specifies the distance from the *z* coordinate of the current graphics position to the *z* coordinate of the next corner of the polygon.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allowing the system to draw concave polygons with the **concave** subroutine.

Closing a filled polygon with the **pclos** subroutine.

Specifying the next point in a polygon with the **pdr** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Moving the current graphics position to a starting point for a filled polygon relative to the current point with the **rpmv** subroutine.

Selecting the shading model used to draw a polygon with the **shademodel** subroutine.

Graphics Library Overview, Drawing with Move-Draw Style Subroutines, and Setting Drawing Attributes.

---

rpmv Subroutine

Purpose

Performs a relative move to the starting point of a filled polygon.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void rpmv
(Coord  dx, Coord  dy, Coord  dz)

void rpmvi
(Icoord  dx, Icoord  dy, Icoord  dz)

void rpmvs
(Scoord  dx, Scoord  dy, Scoord  dz)

void rpmv2
(Coord  dx, Coord  dy)

void rpmv2i
(Icoord  dx, Icoord  dy)

void rpmv2s
(Scoord  dx, Scoord  dy)
```

FORTRAN Syntax

```
SUBROUTINE RPMV(dx, dy, dz)
REAL dx, dy, dz
SUBROUTINE RPMVI(dx, dy, dz)
INTEGER*4 dx, dy, dz
SUBROUTINE RPMVS(dx, dy, dz)
INTEGER*2 dx, dy, dz
SUBROUTINE RPMV2(dx, dy)
REAL dx, dy
SUBROUTINE RPMV2I(dx, dy)
INTEGER*4 dx, dy
SUBROUTINE RPMV2S(dx, dy)
INTEGER*2 dx, dy
```

Note: For FORTRAN users, the INTEGER*2 versions of this subroutine, **RPMVS** and **RPMV2S**, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking one of the short versions of this subroutine with an integer constant will result in unexpected behavior.

Description

The **rpmv** subroutine is the relative version of the **pmv** subroutine. It specifies a relative move to the starting point in a filled polygon, using the current graphics position as the origin. The **rpmv** subroutine updates the current graphics position to the new point.

Between calls to the **rpmv** and **pclos** subroutines, you can issue calls to the following Graphics Library subroutines only:

- **c**
- **color**
- **cpack**
- **lmbind**
- **lmcolor**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

Use the **lmdef** and **lmbind** subroutines to respecify only materials and their properties.

The six different forms for the **rpmv** subroutine are as follows:

	2-D	**3-D**
Int16	rpmv2s	rpmvs
Int32	rpmv2i	rpmvi
float	rpmv2	rpmv

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **rpmv** expects real coordinates, **rpmvi** expects integer coordinates, and **rpmvs** expects short integer coordinates. In addition, the **rpmv2** routines assume a 2-D point instead of a 3-D point.

Parameters

dx Specifies the distance from the *x* coordinate of the current graphics position to the *x* coordinate of the first corner of the polygon.

dy Specifies the distance from the *y* coordinate of the current graphics position to the *y* coordinate of the first corner of the polygon.

dz Specifies the distance from the *z* coordinate of the current graphics position to the *z* coordinate of the first corner of the polygon.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allowing the system to draw concave polygons with the **concave** subroutine.

Closing a filled polygon with the **pclos** subroutine.

Specifying the next point in a polygon with the **pdr** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Drawing a relative polygon with the **rpdr** subroutine.

Selecting the shading model used to draw a polygon with the **shademodel** subroutine.

Graphics Library Overview, Drawing with Move-Draw Style Subroutines, and Setting Drawing Attributes.

---

sbox, sboxi, or sboxs Subroutine

Purpose

Draws a screen-aligned rectangle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void sbox
(Coord  x1, Coord  y1,
Coord  x2, Coord  y2)

void sboxi
(Icoord  x1, Icoord  y1,
Icoord  x2, Icoord  y2)

void sboxs
(Scoord  x1, Scoord  y1,
Scoord  x2, Scoord  y2)
```

FORTRAN Syntax

```
SUBROUTINE SBOX(x1, y1, x2, y2)
REAL x1, y1, x2, y2
SUBROUTINE SBOXI(x1, y1, x2, y2)
INTEGER*4 x1, y1, x2, y2
SUBROUTINE SBOXS(x1, y1, x2, y2)
INTEGER*2 x1, y1, x2, y2
```

Note: For FORTRAN users, the INTEGER*2 version, the **SBOXS** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

All of the foregoing functions are essentially the same except for the type declarations of the parameters.

Description

The **sbox** subroutine draws a two-dimensional, screen-aligned rectangle using the current color, writemask, linestyle, and linestyle repeat. Only these attributes, not the normal line attributes, are used. Most of the lighting/shading/viewing pipeline is bypassed.

The sides of the rectangle are parallel to the screen *x* and *y* axes. This rectangle cannot be rotated. The *z* coordinate is set to zero.

When you use the **sbox** subroutine, you must not use lighting, backfacing, depth-cueing, z-buffering, Gouraud shading, or alphablending.

This subroutine may be faster than the **rect** subroutine. It is useful for drawing a large number of rectangles that do not require rotating.

Parameters

x1 Specifies the *x* coordinate of a corner of the box.
y1 Specifies the *y* coordinate of a corner of the box.
x2 Specifies the *x* coordinate of the opposite corner of the box.
y2 Specifies the *y* coordinate of the opposite corner of the box.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing a rectangle with the **rect** subroutine.

Drawing a filled screen-aligned rectangle with the **sboxf** subroutine.

Graphics Library Overview, Setting Drawing Attributes, Drawing Rectangles, Circles, Arcs, and Polygons.

---

sboxf, sboxfi, or sboxfs Subroutine

Purpose

Draws a filled screen-aligned rectangle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void sboxf
(Coord  x1, Coord  y1,
Coord  x2, Coord  y2)

void sboxfi
(Icoord  x1, Icoord  y1,
Icoord  x2, Icoord  y2)

void sboxfs
(Scoord  x1, Scoord  y1,
Scoord  x2, Scoord  y2)
```

FORTRAN Syntax

```
SUBROUTINE SBOXF(x1, y1, x2, y2)
REAL x1, y1, x2, y2

SUBROUTINE SBOXFI(x1, y1, x2, y2)
INTEGER*4 x1, y1, x2, y2

SUBROUTINE SBOXFS(x1, y1, x2, y2)
INTEGER*2 x1, y1, x2, y2
```

> **Note:** For FORTRAN users, the INTEGER*2 version, the **SBOXFS** subroutine, should not be called with integer constant parameters. For example, 2 is an integer constant; JJ is an integer variable. The XL FORTRAN compiler, invoked by the **xlf** command, stores all integer constants as long integers (INTEGER*4), not as short integers (INTEGER*2). Invoking the short version of this subroutine with an integer constant will result in unexpected behavior.

All of the foregoing functions are essentially the same except for the type declarations of the parameters.

Description

The **sboxf** subroutine draws a filled, two-dimensional, screen-aligned rectangle using the current color, writemask, and pattern. Only these attributes, not the normal area-fill attributes, are used. Most of the lighting/shading/viewing pipeline is bypassed.

The sides of the rectangle are parallel to the screen *x* and *y* axes. This rectangle cannot be rotated. The *z* coordinate is set to zero.

The **sboxf** subroutine performs the same function as the **clear** subroutine. A function equivalent to the **sboxf** subroutine can be obtained by setting the screenmask to the desired size, calling the **clear** subroutine, and then resetting the screenmask. Note that when you use the **clear** subroutine, the lighting, backfacing, depth-cueing, z-buffering, or Gouraud shading does not need to be turned off.

When you use the **sboxf** subroutine, you must not use lighting, backfacing, depth-cueing, z-buffering, Gouraud shading, or alpha blending.

Parameters

x1 Specifies the *x* screen coordinate of a corner of the filled box.
y1 Specifies the *y* screen coordinate of a corner of the filled box.
x2 Specifies the *x* screen coordinate of the opposite corner of the filled box.
y2 Specifies the *y* screen coordinate of the opposite corner of the filled box.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The operation of the **sbox** and **sboxf** subroutines are identical to the **rect** and **rectf** subroutines on the POWERgraphics Gt4 and the GXT1000 adapters.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Clearing to the screenmask with the **clear** subroutine.

Drawing a filled rectangle with the **rectf** subroutine.

Drawing a screen-aligned rectangle with the **sbox** subroutine.

Graphics Library Overview, Setting Drawing Attributes, and Drawing Rectangles, Circles, Arcs, and Polygons.

---

scale Subroutine

Purpose

Scales and mirrors drawing primitives.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void scale(Float32  x, Float32  y, Float32  z)
```

FORTRAN Syntax

```
SUBROUTINE SCALE(x, y, z)
REAL x, y, z
```

Description

The **scale** subroutine shrinks, expands, and mirrors drawing primitives. Values with a magnitude greater than 1 expand the drawing primitive; values with a magnitude less than 1 shrink it. Negative values mirror the primitive. Mirroring will left-right reverse, up-down reverse, or front-back reverse a drawing primitive, depending on which of the three directions is given a negative value.

The **scale** subroutine is a modeling routine; it changes the current transformation matrix. All drawing primitives drawn after the **scale** subroutine executes are affected.

Use the **pushmatrix** and **popmatrix** subroutines to limit the scope of the **scale** subroutine.

If lighting is on, and the **scale** subroutine is called with unequal arguments, the lighting pipeline is forced to renormalize every normal automatically. This may degrade performance on some adapters.

Parameters

x Specifies scaling of the drawing primitive in the *x* direction.
y Specifies scaling of the drawing primitive in the *y* direction.
z Specifies scaling of the drawing primitive in the *z* direction.

Example

The example C language program **cylinder2.c** draws one cylinder, then uses the **scale** subroutine before drawing a second, differently sized cylinder.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Popping the transformation matrix stack with the **popmatrix** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Rotating a graphical primitive (floating-point version) with the **rot** subroutine.

Rotating a graphical primitive (fixed-point version) with the **rotate** subroutine.

Translating a graphical primitive with the **translate** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

screenspace Subroutine

Purpose

Makes a program interpret graphics positions as absolute screen coordinates.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void screenspace( )
```

FORTRAN Syntax

```
SUBROUTINE SCREEN
```

Description

The **screenspace** subroutine makes a program interpret graphics positions as absolute screen coordinates. This allows pixels and locations outside a program's window to be read. The origin, in screen coordinates, is at the lower left corner of the screen. In window coordinates the origin is at the lower left corner of the user-defined window.

The **screenspace** subroutine is equivalent to:

```
int xmin, ymin;

getorigin(&xmin, &ymin);
viewport(-xmin, XMAXSCREEN-xmin,-ymin,YMAXSCREEN-ymin);
ortho2(-0.5,1279.5,-0.5,1023.5);
```

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the position of a window with the **getorigin** subroutine.

Defining a 2-D orthographic transformation with the **ortho2** subroutine.

Setting the area of the window used for all drawing with the **viewport** subroutine.

Graphics Library Overview, Using Viewports and Screenmasks, and Working with Coordinate Systems.

---

scrmask Subroutine

Purpose

Defines a a rectangular 2-D clipping mask.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void scrmask
(Screencoord  left,
Screencoord  right,
Screencoord  bottom,
Screencoord  top)
```

FORTRAN Syntax

```
SUBROUTINE SCRMAS(left, right, bottom, top)
INTEGER*4 left, right, bottom, top
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **scrmask** subroutine defines a rectangular, two-dimensional clipping mask. It is intended to be used primarily for fine character clipping, although it clips all drawing primitives, including **clear**.

By default, the **viewport** subroutine sets the same area for both the viewport and screenmask, which the parameters *left*, *right*, *bottom*, *top* define. Strings that begin outside the viewport are clipped out; this is called gross clipping. Strings that begin inside the viewport, but outside the screenmask, are clipped to the pixel boundaries of the screenmask; this is called fine clipping.

All drawing routines are also clipped to the viewport, but the **scrmask** subroutine is useful only for characters. Gross clipping is sufficient for all other primitives.

Parameters

left	Specifies the coordinate of the left clipping plane of the screenmask.
right	Specifies the coordinate of the right clipping plane of the screenmask.
bottom	Specifies the coordinate of the bottom clipping plane of the screenmask.
top	Specifies the coordinate of the top clipping plane of the screenmask.

> **Note:** The *left* parameter must not be greater than the *right* parameter, nor the *bottom* parameter greater than the *top* parameter, otherwise no text can appear on the screen.

Example

The example C language program **prompt.c** uses the **scrmask** subroutine to define a new screenmask.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current screenmask with the **getscrmask** subroutine.

Setting the area of the window used for all drawing with the **viewport** subroutine.

Graphics Library Overview, Using Viewports and Screenmasks, and Working with Coordinate Systems.

---

setbell Subroutine

Purpose

Sets the duration of the keyboard bell sound.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void setbell(Char8  durat)
```

FORTRAN Syntax

```
SUBROUTINE SETBEL(durat)
CHARACTER*1 durat
```

Description

The **setbell** subroutine sets the duration of the keyboard bell sound. The keyboard bell is activated by the **ringbell** subroutine. Settings for the *durat* parameter are as follows:

Value	Meaning
0	Off
1	Short beep
2	Long beep

Note: This subroutine cannot be used to add to a display list.

Parameter

durat Specifies the duration of the keyboard bell.

Example

The example C language program **ovrlay.c** uses the **setbell** subroutine to set the duration of the bell sound for a short beep.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning off the keyboard click with the **clkoff** subroutine.

Turning on the keyboard click with the **clkon** subroutine.

Turning on the keyboard display lights with the **lampon** subroutine.

Ringing the keyboard bell with the **ringbell** subroutine.

Graphics Library Overview and Using the Keyboard.

---

setcursor Subroutine

Purpose

Sets the cursor characteristics.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void setcursor
(Int16  index, Colorindex  color, Colorindex   writemask)
```

FORTRAN Syntax

```
SUBROUTINE SETCUR(index, color, writemask)
INTEGER*4 index, color, writemask
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **setcursor** subroutine selects a cursor from among those defined with the **defcursor** subroutine. To set the color for the cursor, use the **mapcolor** and **drawmode** subroutines.

Note: This subroutine cannot be used to add to a display list.

Parameters

index	Specifies an index that was previously associated with a bitmap by the **defcursor** subroutine.
color	Retained for compatibility, but disregarded.
writemask	Retained for compatibility, but disregarded.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the origin of a cursor with the **curorigin** subroutine.

Controlling cursor visibility by window with the **curson** or **cursoff** subroutine.

Defining the type and size of a cursor with the **curstype** subroutine.

Defining a cursor with the **defcursor** subroutine.

Setting the drawing mode to CURSORDRAW with the **drawmode** subroutine.

Returning the cursor characteristics with the **getcursor** subroutine.

Changing a color map entry with the **mapcolor** subroutine.

Putting the system in picking mode with the **pick** subroutine.

Graphics Library Overview, Creating a Cursor, and Creating and Managing Windows.

---

setdblights Subroutine

Purpose

Sets the lights on the dial and switch box.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void setdblights(Int32 *mask***)**

FORTRAN Syntax

SUBROUTINE SETDBL(*mask***)**
INTEGER*4 *mask*

Description

The **setdblights** subroutine turns on a combination of the lights on the dial and switch box. Each bit in the mask corresponds to a light. For example, to turn on lights 4, 7, and 22 (and leave all the others off), set the mask to (1<<4) | (1<<7) | (1<<22) = 0x400090.

Note: This subroutine cannot be used to add to a display list.

Parameter

mask Specifies 32 packed bits indicating which lights to turn on.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Graphics Library Overview and Using the Keyboard in GL in *GL3.2 Version 4 for AIX: Programming Concepts*.

---

set_dither Subroutine

Purpose

Controls the dithering of polygons and blits.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void set_dither (Int32 *mode*)

FORTRAN Syntax

SUBROUTINE SETDITHER (*mode*)
INTEGER*4 *mode*

Description

The **set_dither** subroutine controls whether dithering is applied to blits and polygons. Dithering is a type of rounding or truncating of 24-bit pixel color values to make them fit in a smaller (typically 8-bit) frame buffer. Dithering avoids the sharp changes in color (Mach banding) that normally occur when all pixel color values are truncated in the same way. Dithering helps maintain the impression of smooth shading in small frame buffers.

> **Note:** To enable dithering of both polygons and blits, call **set_dither** with the bitwise-OR of both the DITHER_POLYGONS and DITHER_BLITS flags. Calling **set_dither** with just one of these two flags will automatically disable the other mode.

Parameters

mode Token representing the type of dithering to be enabled.
DITHER_OFF (Dithering is disabled for both polygons and blits.)
DITHER_POLYGONS (Dithering is enabled for polygons and triangles.)
DITHER_BLITS (Dithering is enabled for blits.)

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Graphics Library Overview and Using the Keyboard.

---

setlinestyle Subroutine

Purpose

Selects a linestyle.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void setlinestyle(Int32  index)
```

FORTRAN Syntax

```
SUBROUTINE SETLIN(index)
INTEGER*4 index
```

Description

The **setlinestyle** subroutine selects a linestyle. The linestyle, a line attribute, is used whenever a line-drawing primitive is invoked. These include lines, curves, rectangles, polygons, circles, and arcs.

The default linestyle is 0 (zero), which is a solid line. It cannot be redefined.

> **Note:** The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)

Parameter

index Specifies an index into the table of linestyles built by the **deflinestyle** subroutine

Example

The example C language program **colored.c** uses the **setlinestyle** subroutine to use a linestyle previously defined by the **deflinestyle** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a linestyle with the **deflinestyle** subroutine.

Returning the current linestyle with the **getlstyle** subroutine.

Specifying the linewidth with the **linewidth** subroutine.

Drawing Wire Frame Curves and Surface Patches, Drawing NURBS Curves and Surfaces, Drawing with Move-Draw Style Subroutines, Drawing with Begin-End Style Subroutines, Setting Drawing Attributes, Understanding the Hardware Used by GL, Drawing Rectangles, Circles, Arcs, and Polygons.

---

setmap Subroutine

Purpose

Selects one of 16 small color maps in multimap mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void setmap(Int16  mapnum)
```

FORTRAN Syntax

```
SUBROUTINE SETMAP(mapnum)
INTEGER*4 mapnum
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **setmap** subroutine selects one of the 16 small independent maps to be the current color map. Whenever the system is in color map mode, the contents of the frame buffer is interpreted throught a color map to arrive at the color displayed on the screen. When the system is in multimap mode, there are 16 small independent color maps; this subroutine chooses which of these is to be active. Only the least significant bits in the frame buffer are used to look up entries in the current color map.

The 16 small color maps are numbered 0 to 15.

This subroutine is valid only in multimap mode, and is ignored in onemap mode. It will not function in RGB mode.

> **Note:** This subroutine cannot be used to add to a display list.

Parameter

mapnum Number of the color map selected, 0 to 15.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER GXT1000 adapter does not support the **setmap** subroutine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the number of the current color map with the **getmap** subroutine.

Organizing the color map as 16 small maps with the **multimap** subroutine.

Working in Color Map and RGB Modes.

---

setnurbsproperty Subroutine

Purpose

Sets a property for the display of trimmed NURBS surfaces.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void setnurbsproperty(Int32  Property, Float32  Value)
```

FORTRAN Syntax

```
SUBROUTINE SETNUR(property, value)
INTEGER*4 property
REAL value
```

Description

The **setnurbsproperty** subroutine sets a property for the display of a trimmed Non-Uniform Rational B-Spline (NURBS) surfaces. The display of NURBS surfaces can be controlled in different ways. The following is a list of the display properties that can be affected.

C	FORTRAN	Description
N_ERRORCHECKING	NERROR	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	NPIXEL	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.

Parameters

property Specifies the name of the property to be set.
value Specifies the value to which the named property is to be set.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Marking the beginning and end of a NURBS surface definition with the **bgnsurface** and **endsurface** subroutines.

Marking the beginning and end of a NURBS surface trimming loop with the **bgntrim** and **endtrim** subroutines.

Returning the current value of a trimmed NURBS surfaces display property with the **getnurbsproperty** subroutine.

Controlling the shape of a NURBS trimming curve with the **nurbscurve** subroutine.

Controlling the shape of a NURBS surface with the **nurbssurface** subroutine.

Describing a piecewise linear trimming curve for NURBS surfaces with the **pwlcurve** subroutine.

Graphics Library Overview and Drawing NURBS Curves and Surfaces.

---

setpattern Subroutine

Purpose

Selects a pattern for filling polygons and rectangles.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void setpattern(Int32  index)
```

FORTRAN Syntax

```
SUBROUTINE SETPAT(index)
INTEGER*4 index
```

Description

The **setpattern** subroutine selects a pattern from a table of patterns previously defined by the **defpattern** subroutine. The pattern, an area-fill attribute, is used whenever an area-fill primitive is invoked. These primitives include filled polygons, rectangles, circles, and arcs.

The default pattern is 0 (zero), which is solid. If you specify an undefined pattern, the default pattern is selected.

> **Note:** The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)

Parameter

index Specifies the index into the table of defined patterns.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the current color index in color map mode with the **color** subroutine.

Defining a pattern with the **defpattern** subroutine.

Returning the index of the current fill pattern with the **getpattern** subroutine.

Granting write permission to a subset of available bitplanes in color map mode with the **writemask** subroutine.

Drawing Wire Frame Curves and Surface Patches, Drawing NURBS Curves and Surfaces, Drawing with Move-Draw Style Subroutines, Drawing with Begin-End Style Subroutines, Setting Drawing Attributes, Understanding the Hardware Used by GL, Drawing Rectangles, Circles, Arcs, and Polygons.

---

setpup Subroutine

Purpose

Enables or disables a given pop-up menu entry.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void setpup (Int32 *pup***, Int 32** *entry***, Int32** *mode***)**

FORTRAN Syntax

SUBROUTINE SETPUP (*pup***,** *entry***,** *mode)*

INTEGER*4 *pup*, *entry*, *mode*

Description

The **setpup** subroutine enables or disables a given pop-up menu entry. Disabled pop-up menu entries are greyed out and cannot be chosen or selected. When an entry is disabled, the **dopup** subroutine does not return the value of the disabled entry. If the **setpup** subroutine is used properly, submenus associated with the disabled entry are also not accessible.

Enabled entries operate as normal.

Note: This subroutine cannot be added to a display list.

Parameters

pup Specifies the pop-up menu containing the entry to be enabled or disabled.

entry Specifies the cardinal number of the entry to be disabled (use 1 for the first entry, 2 for the second entry, and so on).

mode Sets the mode for the specified menu entry:

C	FORTRAN	Description
PUP_NONE	PUPNON	Enables a menu entry.
PUP_GREY	PUPGRE	Disables a menu entry.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Adding an item to an existing pop-up menu with the **addtopup** subroutine.

Defining a pop-up menu with the **defpup** subroutine.

Displaying a pop-up menu with the **dopup** subroutine.

Allocating and initializing a structure for a new pop-up menu with the **newpup** subroutine.

Graphics Library Overview and Creating and Managing Pop-Up Menus.

---

setvaluator Subroutine

Purpose

Assigns an initial value to a valuator.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void setvaluator
(Device  val,
Int16  init, Int16  min, Int16  max)
```

FORTRAN Syntax

```
SUBROUTINE SETVAL(val, init, min, max)
INTEGER*4 val, init, min, max
```

Note: For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **setvaluator** subroutine sets the initial value and the minimum and maximum values the device can assume.

Notes:

1. Some devices, such as tablets, report values fixed to a grid. In such a case, the device defines an initial position and is ignored.
2. This subroutine cannot be used to add to a display list.
3. When there is a conflict between GL standards and AIXwindows standards, the AIXwindows standard overrides the GL standard. The **setvaluator** subroutine cannot be used to restrict the movement of the cursor because the AIXwindows standard states that no client process (such as GL) can restrict the cursor.

Parameters

val	Specifies the device number for the valuator being set.
init	Specifies the initial value to be assigned to the valuator.
min	Specifies the minimum value that the device can assume.
max	Specifies the maximum value that the device can assume.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current state of a valuator with the **getvaluator** subroutine.

Graphics Library Overview, Using the Keyboard, and Controlling Queues and Devices.

---

shademodel Subroutine

Purpose

Selects the shading style used to draw polygons and lines.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void shademodel(Int32  mode)
```

FORTRAN Syntax

```
SUBROUTINE SHADEM(mode)
INTEGER*4 mode
```

Description

The **shademodel** subroutine determines the shading style that the system uses to render lines and draw filled polygons. When the system uses Gouraud shading, the colors along a line segment or in the interior of a polygon are a linear interpolation of the colors at the vertices.

The shading model can be either FLAT or GOURAUD. When GOURAUD is specified, the colors along a line segment or the interior of a polygon are smooth shaded; that is, the color of a given pixel is a bi-linear interpolation of the colors at the vertices. When FLAT is specified, the entire primitive is drawn with one color.

When drawing flat-shaded polylines, each line segment is drawn with the current color. The current color is the color that was most recently specified before the vertex that defines the segment. That is, if a color is specified before the first vertex, and a second color is specified before the second (final) vertex, the line segment is drawn with the second color.

Parameter

mode Specifies one of two possible flags:

C	FORTRAN	Description
FLAT	FLAT	Instructs the system to draw polygons and lines in a constant color.
GOURAUD	GOURAU	Instructs the system to draw polygons and lines with Gouraud shading. (This is the default shading model.)

Example

The example C language program **backface.c** calls the **shademodel** subroutine, with the value of the *mode* parameter set to **FLAT,** to render the cube faster. This setting of the *mode* parameter shades each side of the cube in a constant color.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The 3-D Color Graphics Processor does not support Gouraud-shaded lines. That is, the setting of the shademodel attribute is ignored when lines are drawn; all lines are drawn flat-shaded. An approximation can be achieved by drawing a 2-vertex polygon (with the **bgnpolygon** and **endpolygon** subroutines); however, 2-vertex polygons ignore the current setting of the linestyle, linewidth, and other line attributes.

When more than one color is specified for a primitive, and flat shading is enabled, the GXT1000 uses the OpenGL semantics to determine the color of the resulting primitive. For triangle meshes, the color used is

the *current color* or the color most recently specified before the *provoking vertex* that caused the triangle to be drawn. For polygons, the polygon is drawn with the color that was current before the first vertex of the polygon was specified.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the shading model used to draw polygons with the **getsm** subroutine.

Drawing with Move-Draw Style Subroutines, Drawing with Begin-End Style Subroutines, Setting Drawing Attributes, Understanding the Hardware Used by GL, Drawing Rectangles, Circles, Arcs, and Polygons.

---

singlebuffer Subroutine

Purpose

Invokes single buffer mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void singlebuffer( )
```

FORTRAN Syntax

```
SUBROUTINE SINGLE
```

Description

The **singlebuffer** subroutine starts single buffer mode, in which the system simultaneously updates and displays the image data in the active bitplanes. Consequently incomplete or changing pictures can be displayed on the screen. The actual repartitioning of the frame buffer into single buffer mode does not occur until the **gconfig** subroutine is called.

Smooth animation, in which all drawing is hidden until it is complete, can be achieved in double buffer mode.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Returning the current display mode with the **getdisplaymode** subroutine.

Reconfiguring the system with the **gconfig** subroutine.

Waiting for a vertical retrace with the **gsync** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Configuring the Frame Buffer and Creating Animated Scenes.

---

splf Subroutine

Purpose

Draws a shaded filled polygon.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void splf
(Int32  n,
Coord  parray[ ][3],
Colorindex  iarray[ ])

void splfi
(Int32  n,
Icoord  parray[ ][3],
Colorindex  iarray[ ])

void splfs
(Int32  n,
Scoord  parray[ ][3],
Colorindex  iarray[ ])

void splf2
(Int32  n,
Coord  parray[ ][2],
Colorindex  iarray[ ])
```

```
void splf2i
(Int32  n,
Icoord  parray[ ][2],
Colorindex  iarray[ ])

void splf2s
(Int32  n,
Scoord  parray[ ][2],
Colorindex  iarray[ ])
```

FORTRAN Syntax

```
SUBROUTINE SPLF(n, parray, iarray)
INTEGER*4 n
REAL parray(3,n)
INTEGER*2 iarray(n)

SUBROUTINE SPLFI(n, parray, iarray)
INTEGER*4 n
INTEGER*4 parray(3,n)
INTEGER*2 iarray(n)

SUBROUTINE SPLFS(n, parray, iarray)
INTEGER*4 n
INTEGER*2 parray(3,n)
INTEGER*2 iarray(n)

SUBROUTINE SPLF2(n, parray, iarray)
INTEGER*4 n
REAL parray(2,n)
INTEGER*2 iarray(n)

SUBROUTINE SPLF2I(n, parray, iarray)
INTEGER*4 n
INTEGER*4 parray(2,n)
INTEGER*2 iarray(n)

SUBROUTINE SPLF2S(n, parray, iarray)
INTEGER*4 n
INTEGER*2 parray(2,n)
INTEGER*2 iarray(n)
```

All of the preceding 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 screen.

Description

The **splf** subroutine 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.

The six different forms for the **splf** subroutine are as follows:

	2-D	**3-D**
Int16	splf2s	splfs
Int32	splf2i	splfi
float	splf2	splf

The syntax for each of the subroutine forms is the same except for the parameter type. They differ only in that **splf** expects real coordinates, **splfi** expects integer coordinates, and **splfs** expects short integer coordinates. In addition, the **splf2*** routines assume a 2-D point instead of a 3-D point.

1. This subroutine must be used in color map mode.

2. This subroutine cannot be used to add to a display list.

Parameters

n	Specifies the number of vertices in the polygon. There can be no more than 256 vertices in a single polygon.
parray	Specifies an array containing the vertices of a polygon.
iarray	Specifies an array containing the color map indexes that determine the intensities of the vertices of the polygon.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting color map mode as the current mode with the **cmode** subroutine.

Allowing the system to draw concave polygons with the **concave** subroutine.

Specifying the next point in a polygon with the **pdr** subroutine.

Specifying the starting point for a polygon with the **pmv** subroutine.

Drawing a polygon with the **poly** subroutine.

Drawing a filled rectangle with the **rectf** subroutine.

Drawing a relative polygon with the **rpdr** subroutine.

Moving the current graphics position to a starting point for a filled polygon relative to the current point with the **rpmv** subroutine.

Graphics Library Overview, Setting Drawing Attributes, Drawing Rectangles, Circles, Arcs, and Polygons.

---

stepunit Subroutine

Purpose

Specifies that a window change size in discrete steps.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void stepunit(Int32  xunit, Int32  yunit)
```

FORTRAN Syntax

```
SUBROUTINE STEPUN(xunit, yunit)
INTEGER*4 xunit, yunit
```

Description

The **stepunit** subroutine specifies the smallest steps (in pixels) by which a window can be resized. This subroutine is called at the beginning of a subroutine, but takes effect only when the **winopen** subroutine is called.

The **stepunit** subroutine can also be called in conjunction with the **winconstraints** subroutine to modify the enforced step size after the window is created. The default step unit is one pixel by one pixel. In other words, by default, the window can be resized arbitrarily.

With the **stepunit** subroutine, the programmer can prevent the user from resizing a window except in discrete jumps. If the step unit is large, this subroutine essentially limits the sizes and shapes of a window.

Note: This subroutine cannot be used to add to a display list.

Parameters

xunit	Specifies the amount of change per unit in the *x* direction, measured in pixels.
yunit	Specifies the amount of change per unit in the *y* direction, measured in pixels.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

strwidth Subroutine

Purpose

Returns the width of the specified text string.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 strwidth(Char8 *  string)
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION STRWID( string,  length)
CHARACTER*(*) string
INTEGER*4 length
```

Description

The **strwidth** subroutine returns the width of a text string in pixels, using the character-spacing parameters of the current raster font. This subroutine is useful when you do a simple mapping from screen space to modeling space.

Undefined characters have zero width.

Note: This subroutine cannot be used to add to a display list.

Parameters

string	Specifies the name of the string.
length	Specifies the number of characters in the string.

Example

The example C language program **prompt.c** uses the **strwidth** subroutine to get the number of pixels needed to draw a prompt string.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Mapping a point on the screen into a line in 3-D modeling coordinates with the **mapw** subroutine.

Mapping a point on the screen into a line in 2-D modeling coordinates with the **mapw2** subroutine.

Graphics Library Overview, Creating Text Characters, Picking and Selecting Overview, Working with Coordinate Systems, and Querying the System.

subpixel Subroutine

Controls the placement of point, line, and polygon vertices

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void subpixel(Int32  bool )
```

FORTRAN Syntax

```
SUBROUTINE SUBPIX(bool)
LOGICAL bool
```

Description

The **subpixel** subroutine controls the placement of point, line, and poygon vertices in screen coordinates. The default value of the *bool* parameter 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 noticeable when points or lines are drawn smooth (see the **pntsmooth** and **linesmooth** subroutines). The **subpixel** subroutine is typically set to True while smooth points or smooth lines are being drawn.

In addition to its effect on vertex position, the **subpixel** subroutine also modifies the scan conversion of lines. Specifically, non-subpixel-positioned lines are drawn *closed,* meaning that connected line segments 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 the state of the **subpixel** subroutine.)

Subpixel-positioned lines produce better results when you use the **logicop** or **blendfunction** subroutines, 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:

```
ortho2(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).

Parameter

bool Either True or False.
False = forces screen vertices to the centers of pixels (default).
True = positions screen vertices exactly.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

On the High-Performance 3-D Graphics Processor polygons are always subpixel positioned, regardless of the value of the **subpixel** subroutine. Subpixel-positioned nonsmooth lines are not implemented.

On the Supergraphics Processor, lines and polygons are never subpixel-positioned, regardless of the value of the **subpixel** subroutine. Vertices are always snapped to the center of a pixel before rasterization begins.

The POWER Gt4 and POWER Gt4x adapters do not support subpixel positioning.

On the POWER GXT1000 adapter, subpixel positioning is always on.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying antialiasing of lines with the **linesmooth** subroutine.

Specifying antialiasing of points with the **pntsmooth** subroutine.

Smoothing Jagged Lines with Antialiasing, Understanding the Hardware Used by GL, Configuring the Frame Buffer.

---

swapbuffers Subroutine

Purpose

Exchanges the front and back buffers in double buffer mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void swapbuffers( )
```

FORTRAN Syntax

```
SUBROUTINE SWAPBU
```

Description

The **swapbuffers** subroutine exchanges the front and back buffers in double buffer mode. Once an image is fully drawn in the back buffer, the **swapbuffers** subroutine displays that image.

The swapping of buffers occurs only during a vertical retrace. A minimum of the number of vertical retraces specified by the **swapinterval** subroutine must have elapsed since the last call to **swapbuffers** before the current request is honored. After this call is made, all drawing to the front and back buffers is disabled until the swap occurs.

This subroutine is ignored in single buffer mode.

Example

The example C language program **backface.c** uses the **swapbuffers** subroutine to display a cube after drawing it in the back buffer.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling drawing in the back buffer with the **backbuffer** subroutine.

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Enabling drawing in the front buffer with the **frontbuffer** subroutine.

Defining a minimum time between buffer swaps with the **swapinterval** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Configuring the Frame Buffer and Creating Animated Scenes.

---

swapinterval Subroutine

Purpose

Defines a minimum time between buffer swaps.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void swapinterval(Int16  interval )
```

FORTRAN Syntax

SUBROUTINE SWAPINT(*interval***)**
INTEGER*4 *interval*

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **swapinterval** subroutine defines a minimum time between buffer swaps. The time is measured in units of vertical retraces, with the default interval being 1. For example, for a swap interval of 5, the system refreshes the screen at least five times between successive buffer swaps.

The **swapinterval** subroutine changes frames at a steady rate if a new image can be created within one swap interval. This subroutine is valid only in double buffer mode and is ignored in single buffer mode.

> **Note:** This subroutine cannot be used to add to a display list.

Parameter

interval	Specifies the number of retraces to wait before swapping the front and back buffers.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER GXT1000 adapter does not support the **swapinterval** subroutine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Setting the display mode to double buffer mode with the **doublebuffer** subroutine.

Exchanging the front and back buffers with the **swapbuffers** subroutine.

Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

Configuring the Frame Buffer and Creating Animated Scenes.

---

swaptmesh Subroutine

Purpose

Toggles the triangle mesh register pointer.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void swaptmesh( )
```

FORTRAN Syntax
```
SUBROUTINE SWAPTM
```

Description
The **swaptmesh** subroutine toggles the triangle mesh register pointer.

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, replacing the alternate stored vertices. The **swaptmesh** subroutine toggles the pointer without specification of a new vertex (and no triangle is drawn).

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Drawing triangle mesh vertices with the **bgntmesh** subroutine.

Ending a series of triangle mesh vertices with the **endtmesh** subroutine.

Transferring a vertex to the graphics pipe with the **v** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, and Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

swinopen Subroutine

Purpose
Creates a restricted subwindow.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

Int32 swinopen(Int32 * *parentid***)**

FORTRAN Syntax

INTEGER*4 FUNCTION SWINOP(*parentid***)**
INTEGER*4 *parentid*

Description

The **swinopen** subroutine creates a subwindow inside a parent window. The subwindow is mapped (created and displayed) immediately upon receipt of this call. The subwindow then becomes the current window.

Subwindows have no border and therefore cannot be moved by the applications user. Other than this, subwindows behave very much like windows created with the **winopen** subroutine.

When a subwindow is created, a completely independent graphics context is also created for it. That is, subwindows have their own current color linestyle, pattern, matrix stack, viewport stack, attribute stack, name stack, and so on. These attributes are not shared with the parent. The attributes are initialized to their default values (for example, empty stack). See the **greset** subroutine for the list of default values.

The position of a subwindow is measured relative to the origin of the parent window. Subwindows move with the parent window. If a parent window is moved, all of its subwindows move with it.

Subwindows are clipped to the boundaries of the parent window. That is, by drawing inside a subwindow, one can never draw outside of the boundaries of the parent window. Otherwise, subwindows can be positioned arbitrarily inside a parent window, even if such positioning means that the subwindow is completely clipped. If a subwindow is exposed or otherwise needs redrawing, both the parent and the subwindow receive redraw events.

Because subwindows cannot be moved or resized by the user, none of the window constraint subroutines have any meaning and therefore do not affect the state of a subwindow. Window constraint subroutines include the **minsize**, **maxsize**, and **stepunit** subroutines. Because subwindows have no border, they cannot be given a title. Subwindows also cannot be iconified.

Subwindows can be pushed and popped. Subwindows can be positioned or moved with either the **prefposition** or **prefsize** subroutine followed by a call to the **winconstraints** subroutine, or by using the **winposition** or **winmove** subroutine.

Subwindows can have sub-subwindows. The origin of a sub-subwindow is measured relative to the origin of the subwindow (its parent).

Note: This subroutine cannot be used to add to a display list.

Parameter

parentid Specifies the identifier, or handle, of the parent window.

Return Value

The identifier, or handle, for the subwindow.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Creating a new window with the **winopen** subroutine.

Creating and Managing Windows.

---

textport Subroutine

Purpose

Allocates an area of the screen for the textport.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void textport
(Screencoord  left, Screencoord  right,
Screencoord  bottom, Screencoord  top)
```

FORTRAN Syntax

```
SUBROUTINE TEXTPO(left, right, bottom, top)
INTEGER*4 left, right, bottom, top
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when started from a FORTRAN program, although it accepts short integers when started from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **textport** subroutine allocates an area on the screen for the textport window.

> **Note:** This subroutine cannot be used to add to a display list.

Parameters

left	Specifies the *x* screen coordinate for the left side of the textport.
right	Specifies the *x* screen coordinate for the right side of the textport.
bottom	Specifies the *y* screen coordinate for the bottom of the textport.
top	Specifies the *y* screen coordinate for the top of the textport.

Example

The example C language program **tpbig.c** uses the **textport** subroutine to define a textport.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Turning off the textport with the **tpoff** subroutine.

Turning on the textport with the **tpon** subroutine.

Graphics Library Overview and Working with the Textport.

---

tie Subroutine

Purpose

Ties two valuators to a button.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void tie(Device  button, Device  val1, Device  val2)
```

FORTRAN Syntax

```
SUBROUTINE TIE(button, val1, val2)
INTEGER*4 button, val1, val2
```

Description

The **tie** subroutine requires a button and two valuators, specified by the *val1* and *val2* parameters. 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.

Tie one valuator to a button by making the *val2* parameter = 0. Untie a button by making both the *val1* and *val2* parameters = 0. The *button* parameter precedes both the *val1* and *val2* parameters in the event queue. The *val1* parameter appears before the *val2* parameter.

Note: This subroutine cannot be used to add to a display list.

Parameters

button	Specifies the current state of a button.
val1	Specifies the current position of the first valuator.
val2	Specifies the current position of the second valuator.

Example

The example C language program **vlsi.c** uses the **tie** subroutine to enter the current mouse coordinates into the event queue whenever the left or middle mouse buttons are pressed.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current state of a button with the **getbutton** subroutine.

Graphics Library Overview, Using the Keyboard, and Controlling Queues and Devices.

---

tpoff Subroutine

Purpose

Turns off the textport.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void tpoff( )
```

FORTRAN Syntax

```
SUBROUTINE TPOFF
```

Description

The **tpoff** subroutine pushes the textport, the window associated with the shell that invoked the graphics program, behind all other windows.

When the textport is off, characters are not written to it, nor does it display on the screen. The textport automatically turns on when a program completes execution.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allocating an area of the screen for the textport with the **textport** subroutine.

Turning on the textport with the **tpon** subroutine.

Graphics Library Overview and Working with the Textport.

---

tpon Subroutine

Purpose

Turns on the textport.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void tpon( )
```

FORTRAN Syntax

```
SUBROUTINE TPON
```

Description

The **tpon** subroutine brings the textport, the window associated with the shell that invoked the graphics program, to the front of any windows that conceal it.

Note: This subroutine cannot be used to add to a display list.

Example

The example C language program **tpbig.c** uses the **tpon** subroutine to enable a textport for drawing character strings into.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Allocating an area of the screen for the textport with the **textport** subroutine.

Turning off the textport with the **tpoff** subroutine.

Graphics Library Overview and Working with the Textport.

---

translate Subroutine

Purpose

Translates graphical primitives.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void translate(Coord  x, Coord  y, Coord  z)
```

FORTRAN Syntax

```
SUBROUTINE TRANSL(x, y, z)
REAL x, y, z
```

Description

The **translate** subroutine moves the modeling space origin to a new point relative to the current origin. The point (x, y, z) specified by the parameters becomes the new modeling space origin. Because all drawing primitives draw relative to the origin of the current modeling coordinate system, all primitives called after this subroutine will appear to have been translated.

The **translate** subroutine is a modeling routine that changes the current transformation matrix. All primitives drawn after this subroutine executes are translated. Use the **pushmatrix** and **popmatrix** subroutines to preserve an untranslated modeling space.

Parameters

x	Specifies the *x* coordinate of a point in modeling coordinates.
y	Specifies the *y* coordinate of a point in modeling coordinates.
z	Specifies the *z* coordinate of a point in modeling coordinates.

Example

The example C language program **backface.c** uses the **translate** modeling subroutine and alters the current transformation matrix to model the sides of a cube using a square.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Popping the transformation matrix stack with the **popmatrix** subroutine.

Pushing down the transformation matrix stack with the **pushmatrix** subroutine.

Rotating a graphical primitive (floating-point version) with the **rot** subroutine.

Rotating a graphical primitive (fixed-point version) with the **rotate** subroutine.

Scaling and mirroring objects with the **scale** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

---

underlay Subroutine

Purpose

Sets the number of user-defined bitplanes used for underlay drawing.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void underlay(Int32  planes)
```

FORTRAN Syntax

```
SUBROUTINE UNDERL(planes)
INTEGER*4 planes
```

Description

The **underlay** subroutine sets the number of user-defined bitplanes used for underlay colors to 0, 2, or 4, depending on the adapter. The underlay color appears whenever all the bits in the color bitplanes are 0 (zero). The system has either two or four bitplanes that can be allocated as either underlay or overlay. Call the **underlay** subroutine to set them as underlay bitplanes.

The High-Performance 8-bit 3-D Color Graphics Processor can be configured with either 0 or 2 underlay planes. The 8-bit adapter has a total of 2 auxiliary planes, which can be configured into 2/0 or 0/2 overlay/underlay. For example, setting the number of underlay planes to 2 forces the number of overlay planes to 0.

The High-Performance 24-bit 3-D Color Graphics Processor can be configured with either 0, 2, or 4 underlay planes. The 24-bit adapter has a total of 4 auxiliary planes, which can be configured into 4/0, 2/2, or 0/4 overlay/underlay. For example, setting the number of underlay planes to 4 forces the number of overlay planes to 0.

Call the **gconfig** subroutine after the **underlay** subroutine to activate the underlay setting.

When the drawing mode is UNDERDRAW, all drawing occurs in the underlay bitplanes. In UNDERDRAW mode, FLAT is the only available shading model.

Notes:

1. The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See Understanding the Adapter.)
2. This subroutine cannot be used to add to a display list.

Parameter

planes Specifies the number of bitplanes to use for underlay drawing.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Only the Color Graphics Processor has underlay planes.

The POWERgraphics GXT1000 does not support the **underlay** subroutine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the target frame buffer for drawing subroutines with the **drawmode** subroutine.

Reconfiguring the system with the **gconfig** subroutine.

Setting the number of bitplanes used for overlay colors with the **overlay** subroutine.

Configuring the Frame Buffer, and Writemasks and Logical Operations.

---

unqdevice Subroutine

Purpose

Disables an input device for event queuing

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void unqdevice(Device  dev)
```

FORTRAN Syntax

```
SUBROUTINE UNQDEV(dev)
INTEGER*4 dev
```

Description

The **unqdevice** subroutine 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 the **qreset** subroutine to flush the event queue.

Note: This subroutine cannot be used to add to a display list.

Parameter

dev Specifies an identifier for the device to be disabled.

Example

The example C language program **prompt.c** uses the **unqdevice** subroutine to disable input from the keyboard.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fdevice.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling an input device for event queuing with the **qdevice** subroutine.

Emptying the event queue with the **qreset** subroutine.

Graphics Library Overview and Controlling Queues and Devices.

v Subroutine

Purpose

Transfers a 2-D, 3-D, or 4-D vertex to the graphics pipe.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void v2s(Int16 *vector***[2])**

void v2f(Float32 *vector***[2])**

void v2i(Int32 *vector***[2])**

void v2d(Float64 *vector***[2])**

void v3s(Int16 *vector***[3])**

void v3f(Float32 *vector***[3])**

void v3i(Int32 *vector***[3])**

void v3d(Float64 *vector***[3])**

void v4s(Int16 *vector***[4])**

void v4f(Float32 *vector***[4])**

void v4i(Int32 *vector***[4])**

void v4d(Float64 *vector***[4])**

FORTRAN Syntax

SUBROUTINE V2S(*vector***)**
INTEGER*2 *vector***(2)**

SUBROUTINE V2I(*vector***)**
INTEGER*4 *vector***(2)**

SUBROUTINE V2F(*vector***)**
REAL *vector***(2)**

SUBROUTINE V2D(*vector***)**
REAL*8 *vector***(2)**

SUBROUTINE V3S(*vector***)**
INTEGER*2 *vector***(3)**

SUBROUTINE V3I(*vector***)**
INTEGER*4 *vector***(3)**

SUBROUTINE V3F(*vector***)**
REAL *vector***(3)**

SUBROUTINE V3D(*vector***)**
REAL*8 *vector***(3)**

SUBROUTINE V4S(*vector***)**
INTEGER*2 *vector***(4)**

SUBROUTINE V4I(*vector***)**
INTEGER*4 *vector***(4)**

SUBROUTINE V4F(*vector***)**
REAL *vector***(4)**

SUBROUTINE V4D(*vector***)**
DOUBLE *vector***(4)**

Description

The **v** subroutine transfers a single 2-D (**v2**), 3-D (**v3**), or 4-D (**v4**) vertex to the graphics pipeline. 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. The *w* coordinate defaults to 1.0.

The Graphics Library subroutines **bgnpoint**, **endpoint**, **bgnline**, **endline**, **bgnclosedline**, **endclosedline**, **bgnpolygon**, **endpolygon**, **bgntmesh**, and **endtmesh** determine how the vertex is interpreted. For example, vertices specified between the **bgnpoint** and **endpoint** subroutines draw single pixels (points) on the screen. Likewise, those specified between the **bgnline** and **endline** subroutines 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 the **bgnpoint**, **bgnline**, **bgnclosedline**, **bgnpolygon**, and **bgntmesh** subroutines are active set the current graphics position. They do not have any effect on the frame buffer contents. The **endpoint**, **endline**, **endclosedline**, **endpolygon**, and **endtmesh** subroutines have varied effects on the current graphics position.

Parameters

vector Specifies 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) 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**).

The nine different forms for the **v** subroutine are as follows:

	2-D	**3-D**	**4-D**
Int16	v2s	v3s	v4s
Int32	v2i	v3i	v4i
Float32	v2f	v3f	v4f
Float64	v2d	v3d	v4d

The syntax for each of the subroutine forms is the same except for the first argument. They differ only in that **vi** expects long integer coordinates, **vs** expects short integer coordinates, **vf** expects single-precision floating point coordinates, and **vd** expects double-precision floating point coordinates. In addition, the **v2*** routines assume a 2-D point, the **v3*** routines assume a 3-D point, and the **v4*** routines assume a 4-D point (in homogeneous coordinates.

Example

The example C language program **cylinder2.c** calls the **v3f** subroutine between the **bgnpolygon** subroutine and the **endpolygon** subroutine to specify the vertices of a polygon.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Drawing closed line vertices with the **bgnclosedline** subroutine.

Drawing vertex-based lines with the **bgnline** subroutine.

Drawing vertex-based points with the **bgnpoint** subroutine.

Drawing vertex-based polygons with the **bgnpolygon** subroutine.

Drawing triangle mesh vertices with the **bgntmesh** subroutine.

Ending a series of closed line vertices with the **endclosedline** subroutine.

Ending a series of vertex-based lines with the **endline** subroutine.

Ending a series of vertex-based points with the **endpoint** subroutine.

Ending a vertex-based polygon with the **endpolygon** subroutine.

Ending a series of triangle mesh vertices with the **endtmesh** subroutine.

Graphics Library Overview, Drawing with Begin-End Style Subroutines, and Understanding the Hardware Used by GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

viewport Subroutine

Purpose

Sets the area of the window used for all drawing.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void viewport
(Screencoord  left, Screencoord  right,
Screencoord  bottom, Screencoord  top)
```

FORTRAN Syntax

```
SUBROUTINE VIEWPO(left, right, bottom, top)
INTEGER*4 left, right, bottom, top
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **viewport** subroutine specifies, in pixels, the area of the window in which all drawing occurs. The viewport locations are specified relative to the lower left corner of the window. Specifying the viewport is the last step in mapping modeling coordinates to screen coordinates.

The portion of world space that the **window**, **ortho**, or **perspective** subroutines describe is mapped into the viewport. The *left*, *right*, *bottom*, *top* coordinates define a rectangular area on the screen.

The viewport is set to the size of the window when the window is first opened (thus, a **getviewport** would return the size of the window). However, if the window is resized, the viewport dimensions are not automatically updated (use the **reshapeviewport** subroutine). The viewport may be much larger or much smaller than the window size. All drawing occurs inside the current viewport, but is clipped to the window.

If the *left* parameter is greater than the *right* parameter, the displayed image will be left-right reversed. If the *bottom* parameter is greater than the *top* parameter, the displayed image will be up-down reversed.

The viewport is the mapping from normalized device coordinates (NDC) to device coordinates (DC). The same function is provided for the z-direction with the **lsetdepth** subroutine.

The **viewport** subroutine also resets the screenmask. The screenmask is set to be exactly the same size as the viewport.

Parameters

left	Specifies the *x* location (in pixels) of left side of viewport.
right	Specifies the *x* location (in pixels) of right side of viewport.
bottom	Specifies the *y* location (in pixels) of bottom of viewport.
top	Specifies the *y* location (in pixels) of top of viewport.

Example

The example C language program **paint.c** uses the **viewport** subroutine to define a viewport for displaying an image.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining the viewport depth with the **lsetdepth** subroutine.

Popping the viewport off the viewport stack with the **popviewport** subroutine.

Pushing the viewport onto the viewport stack with the **pushviewport** subroutine.

Changing the viewport shape with the **reshapeviewport** subroutine.

Defining a rectangular 2-D clipping mask with the **scrmask** subroutine.

Graphics Library Overview, Using Viewports and Screenmasks, and Working with Coordinate Systems.

winclose Subroutine

Purpose
Closes the identified window.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void winclose(Int32  windowid)
```

FORTRAN Syntax
```
SUBROUTINE WINCLO(windowid)
INTEGER*4 windowid;
```

Description
The **winclose** subroutine closes the window associated with the *windowid* parameter. The identifier for a window is the function return value from the call to the **winopen** subroutine that created the window.

Notes:

1. If the current window is closed with the **winclose** subroutine, another window is chosen arbitrarily as the current window. All subsequent drawing is directed to this window. To avoid encountering unexpected behavior due to arbitrary window selection, use the **winset** subroutine to choose a new current window.
2. The **winclose** subroutine cannot be used to add to a display list.

Parameter
windowid Specifies which window to close.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information
Creating a window with the **winopen** subroutine.

Setting the current window with the **winset** subroutine.

winconstraints Subroutine

Purpose
Binds window constraints to the current window.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void winconstraints( )
```

FORTRAN Syntax
```
SUBROUTINE WINCON
```

Description
The **winconstraints** subroutine binds the specified constraints to the current window. Because this subroutine assumes the existence of a current window, the **winopen** subroutine must be called before the **winconstraints** subroutine.

The values of the window constraints are set by using the following subroutines:
- **fudge**
- **iconsize**
- **keepaspect**
- **maxsize**
- **minsize**
- **noborder**
- **noport**
- **prefposition**
- **prefsize**
- **stepunit**

After binding these constraints to a window, the **winconstraints** subroutine resets the specified window constraints to their default values. Thus, to reset and bind the current constraints, call the **winconstraints** subroutine twice in a row.

The changes made to window attributes (whether actual constraints imposed by the **maxsize** and **minsize** subroutines, or limits only suggested by the **prefsize** and **prefposition** subroutines) are bound to the window and take effect immediately.

Note: This subroutine cannot be used to add to a display list.

Example
To set the current window's aspect ratio, the example C language program **colored.c** calls the **winconstraints** subroutine after calling the **keepaspect** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Specifying the size of a window icon with the **iconsize** subroutine.

Specifying the aspect ratio of a window with the **keepaspect** subroutine.

Specifying the maximum size of a window with the **maxsize** subroutine.

Specifying the minimum size of a window with the **minsize** subroutine.

Removing the border from a window with the **noborder** subroutine.

Specifying that a program does not require a window with the **noport** subroutine.

Constraining the size of a window with the **prefsize** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

windepth Subroutine

Purpose

Indicates the stacking order of windows on the screen.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 windepth(Int32  windowid)
```

FORTRAN Syntax

```
SUBROUTINE WINDEP(windowid)
INTEGER*4 windowid
```

Description

The **windepth** subroutine returns a number that can be compared against the same return value for other windows to indicate the stacking order of a program's windows on the screen.

Parameter

windowid Specifies which window to test.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Raising the current window on top of all other windows with the **winpop** subroutine.

Lowering the current window beneath all other windows with the **winpush** subroutine.

Creating and Managing Windows.

---

window Subroutine

Purpose

Defines a perspective projection transformation in terms of x and y coordinates.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void window
(Coord  left, Coord  right,
Coord  bottom, Coord  top,
Coord  near, Coord  far)
```

FORTRAN Syntax

```
SUBROUTINE WINDOW(left, right, bottom, top, near, far)
REAL left, right, bottom, top, near, far
```

Description

The **window** subroutine set the current projection transformation to be a perspective transformation. With a perspective transformation, figures do not get smaller as they recede in relation to the viewer.

The **window** subroutine defines a frustum in eye coordinates. All figures outside this frustum are clipped and not drawn on the screen. All objects contained within this frustum are drawn on the screen in perspective.

The front and back of the frustum are the near and far clipping planes, respectively, located at *near* and *far***.** The sides of the frustum are the projection of a rectangle in the near clipping plane. That is, given a rectangle in the near clipping plane, the sides of the frustum are given by extending from the eye (the origin) through the near clipping plane to the far clipping plane. The sides of the frustum are the left, right, bottom, and top clipping planes. The size of the rectangle on the near clipping plane is given by the parameters *left***,** *right***,** *bottom***,** and *top*.

The **window** subroutine is very similar to the **perspective** subroutine. The only difference between these two is the manner in which the arguments specify the viewing frustum.

After the **window** subroutine completes, the eye coordinate system is set up so that x is to the right, y is up, and z is towards the viewer (out of the screen).

In single matrix mode, the **window** subroutine loads a matrix onto the matrix stack, replacing the current top matrix. In viewing matrix mode and projection matrix mode, the system replaces the current projection matrix.

> **Note:** Do not confuse the **window** subroutine and its functions with the GL windowing subroutines, and the Enhanced X-Windows subroutines, which control the placement and size of rectangular windows on the screen.

Parameters

left	Specifies the *x* coordinate of left side of frustum.
right	Specifies the *x* coordinate of right side of frustum.
bottom	Specifies the *y* coordinate of bottom of frustum.
top	Specifies the *y* coordinate of top of frustum.
near	Specifies the *z* coordinate of the near clipping plane.
far	Specifies the *z* coordinate of the far clipping plane.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Defining a 3-D orthographic transformation with the **ortho** subroutine.

Defining a perspective projection transformation in terms of a field of view with the **perspective** subroutine.

Setting the area of the window used for all drawing with the **viewport** subroutine.

Graphics Library Overview and Working with Coordinate Systems.

winget Subroutine

Purpose
Returns the identifier of the current window.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
Int32 winget( )
```

FORTRAN Syntax
```
INTEGER*4 FUNCTION WINGET
```

Description
The **winget** subroutine returns the identifier of the current window. The *current window* is the window into which all drawing is directed, the window most recently selected with the **winset** subroutine, or the window most recently created with the **winopen** subroutine. The *window identifier* of a window is the integer returned by the **winopen** subroutine.

> **Attention:** If the current window is closed by the **winclose** subroutine, another window is chosen arbitrarily as the current window. In such cases, use the **winset** subroutine to choose a new current window.
>
> **Note:** The **winget** subroutine cannot be used to add to a display list.

Return Value
The identifier of the current window.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information
Closing the identified window with the **winclose** subroutine.

Creating a window with the **winopen** subroutine.

Setting the current window with the **winset** subroutine.

Creating and Managing Windows.

winmove Subroutine

Purpose
Moves the current window by its lower-left corner.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void winmove(Int32  originx, Int32  originy)
```

FORTRAN Syntax
```
SUBROUTINE WINMOV(originx, originy)
INTEGER*4 originx, originy
```

Description
The **winmove** subroutine moves the current window so that its origin, the lower-left corner, is at the screen coordinates specified in pixels by the *originx* and *originy* parameters. The **winmove** subroutine removes the current size constraint. After the **winmove** subroutine is started, the user can interactively change the position and size of the window.

The **winmove** subroutine does not remove the other window constraints, such as the minimum and maximum size constraints, the aspect ratio, or the stepunit.

To keep the window size constrained, use the **prefposition** or the **prefsize** subroutines to move and resize a window.

Note: This subroutine cannot be used to add to a display list.

Parameters
originx	Specifies the *x* coordinate of the lower-left corner of the new location for the current window.
originy	Specifies the *y* coordinate of the lower-left corner of the new location for the current window.

Implementation Specifics
This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files
/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Binding window constraints to the current window with the **winconstraints** subroutine.

Raising the current window on top of all other windows with the **winpop** subroutine.

Changing the current location and size of a window with the **winposition** subroutine.

Lowering the current window beneath all other windows with the **winpush** subroutine.

Creating and Managing Windows.

---

winopen Subroutine

Purpose

Creates a window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
Int32 winopen(Char8 * name)
```

FORTRAN Syntax

```
INTEGER*4 FUNCTION WINOPE( name,  length)
CHARACTER*(*) name
INTEGER*4 length
```

Description

The **winopen** subroutine creates a window as defined by the current values of the window constraints. This new window becomes the current window. If this is the first time that a program has called the **winopen** subroutine, the system also initializes the graphics system.

The returned value for this function is the window identifier (gwid) for the window just created. Use this value to identify the window to other graphics functions. If no additional windows are available, this function returns -1 (negative one).

If called before the **winopen** subroutine, the following subroutines change the default window constraints:

- **fudge**
- **iconsize**
- **keepaspect**
- **maxsize**
- **minsize**
- **noborder**
- **noport**
- **prefposition**
- **prefsize**

- **stepunit**

If a window's size and location are left unconstrained, the system allows the user to place and size the window.

All drawing (lines, polygons, and NURBS) is done in the current window. Lighting, depth-cueing, and z-buffering all apply to the current window. Every window has an independent set of stacks: matrix stack, name stack, attribute stack, and viewport stack, and all stack manipulation routines such as matrix multiplies are directed at the current window.

The only attributes that are shared across windows are those defined with the **defcursor**, **deflinestyle**, **defpattern**, **defrasterfont**, **lmdef**, **loadXfont**, and **makeobj** subroutines.

The **winopen** subroutine examines the environment variable DISPLAY to determine to which AIXwindows server to make a connection. The default DISPLAY value is `unix:0`. Only local sessions are currently supported. A GL session can only run on an AIXwindows server that has extensions to support GL. Not all AIXwindows servers support GL.

The **winopen** subroutine queues the INPUTCHANGE and REDRAW pseudo devices.

Note: This subroutine cannot be used to add to a display list.

Parameters

name	Specifies the window title displayed on the left hand side of the title bar. A zero-length string displays no title.
length	Specifies the length of the string in the *name* parameter.

Example

The example C language program **colored.c** uses the **winopen** subroutine to create a window with characteristics previously defined in the program.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying pixel values to be added to a window with the **fudge** subroutine.

Giving a program the entire screen as a window with the **fullscrn** subroutine.

Obtaining the position of a window with the **getorigin** subroutine.

Obtaining the size of the window with the **getsize** subroutine.

Specifying the size of a window icon with the **iconsize** subroutine.

Specifying the title of a window icon with the **icontitle** subroutine.

Specifying the aspect ratio of a window with the **keepaspect** subroutine.

Specifying the maximum size of a window with the **maxsize** subroutine.

Specifying the minimum size of a window with the **minsize** subroutine.

Removing the border from a window with the **noborder** subroutine.

Specifying that a program does not require a window with the **noport** subroutine.

Constraining the window position and size with the **prefposition** subroutine.

Constraining the size of a window with the **prefsize** subroutine.

Specifying a window size change in discrete steps with the **stepunit** subroutine.

Creating a restricted subwindow with the **swinopen** subroutines.

Closing the identified window with the **winclose** subroutine.

Binding window constraints to the current window with the **winconstraints** subroutine.

Adding a title bar to the current window with the **wintitle** subroutine.

Creating and Managing Windows.

---

winpop Subroutine

Purpose

Raises the current window on top of all other windows.

> **Note:** The GL window management routines are built on top the X11 Window System library. Because X11 is asynchronous in nature, there is no guarantee that windowing commands and drawing commands will be performed in the order issued. In particular, do *not* assume that the current window will be on top when the **winpop** subroutine returns (it will be moved to the top eventually but asynchronously from the subroutine call). When using the **winpop()** subroutine, wait for a REDRAW event for the associated window before drawing into it. Otherwise, early drawing (drawing before the window has been moved to the top) will be clipped to the old window boundaries.

Use the **gflush()** subroutine to ensure drawing is complete. Unless this is done, window pushes and pops may be scheduled and completed before drawing is complete.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void winpop( )
```

FORTRAN Syntax

```
SUBROUTINE WINPOP
```

Description

The **winpop** subroutine raises the current window from anywhere in the stack of windows to the top position.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Lowering the current window beneath all other windows with the **winpush** subroutine.

Creating and Managing Windows.

---

winposition Subroutine

Purpose

Changes the size and position of the current window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void winposition
(Int32  x1, Int32  x2,
Int32  y1, Int32  y2)
```

FORTRAN Syntax

```
SUBROUTINE WINPOS(x1, x2, y1, y2)
INTEGER*4 x1, x2, y1, y2;
```

Description

The **winposition** subroutine repositions and resizes the current window. The window is positioned at the new location as soon as the system receives the **winposition** subroutine call; the reposition is not deferred to a later time. If no window is open, this subroutine has no effect.

The **winposition** subroutine removes the current size constraint. After the **winposition** subroutine has been started, the user can interactively change the position and size of the window.

The **winposition** subroutine does not remove the other window constraints, such as the minimum and maximum size constraints, the aspect ratio, or the stepunit.

To keep the window size constrained, use the **prefposition** or the **prefsize** subroutines to move and resize a window.

Note: This subroutine cannot be used to add to a display list.

Parameters

x1 Specifies the *x*-pixel screen coordinate of the first corner of the new location for the current window. The first corner is diagonally opposite the second corner.
x2 Specifies the *x*-pixel screen coordinate of the second corner of the new location for the current window. The second corner is diagonally opposite the first corner.
y1 Specifies the *y*-pixel screen coordinate of the first corner of the new location for the current window.
y2 Specifies the *y*-pixel screen coordinate of the second corner of the new location for the current window.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Constraining the window position and size with the **prefposition** subroutine.

Constraining the window size with the **prefsize** subroutine.

Moving the current window by its lower-left corner with the **winmove** subroutine.

Raising the current window on top of all other windows with the **winpop** subroutine.

Lowering the current window beneath all other windows with the **winpush** subroutine.

Creating and Managing Windows.

---

winpush Subroutine

Purpose

Lowers the current window beneath all other windows.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void winpush( )
```

FORTRAN Syntax

```
SUBROUTINE WINPUSH
```

Description

The **winpush** subroutine lowers the current window from anywhere in the stack of windows to the bottom position.

Note: This subroutine cannot be used to add to a display list.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Raising the current window on top of all other windows with the **winpop** subroutine.

Creating and Managing Windows.

---

winset Subroutine

Purpose

Sets the current window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void winset(Int32  windowid)
```

FORTRAN Syntax

```
SUBROUTINE WINSET(windowid)
INTEGER*4 windowid
```

Description

The **winset** subroutine takes the window associated with the *windowid* parameter and makes it the current window.

All drawing (lines, polygons, and NURBS) is done in the current window. Lighting, depth-cueing, and z-buffering all apply to the current window. Every window has an independent set of stacks: matrix stack, name stack, attribute stack, and viewport stack, and all stack manipulation routines such as matrix multiplies are directed at the current window.

The only attributes that are shared across windows are those defined with the **defcursor**, **deflinestyle**, **defpattern**, **defrasterfont**, **lmdef**, **loadXfont**, and **makeobj** subroutines.

The **winset** subroutine is the only subroutine that switches graphics servers.

The **winset** subroutine does *not* raise the indicated window to the top position. Use the **winpop** subroutine to raise a window to the top.

Note: This subroutine cannot be used to add to a display list.

Parameter

windowid Specifies which window to set as current.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the identifier of the current window with the **winget** subroutine.

Creating a new window with the **winopen** subroutine.

Creating and Managing Windows.

---

wintitle Subroutine

Purpose

Adds a title bar to the current window.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void wintitle(Char8 * name)
```

FORTRAN Syntax

```
SUBROUTINE WINTIT( name,  length)
CHARACTER*(*) name
INTEGER*4 length
```

Description

The **wintitle** subroutine adds a title to the current window. Use `wintitle("")` to clear the title.

Note: This subroutine cannot be used to add to a display list.

Parameters

name	Specifies title to display in the title bar of the current window.
length	Specifies the number of characters in the name string.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the title of a window icon with the **icontitle** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows.

---

winX Subroutine

Purpose

Converts an Enhanced X-Windows window into a GL window.

Library

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
short winX (Display * dpy, Window  xid)
```

FORTRAN Syntax

```
INTEGER *2 FUNCTION WINX(dpy, xid)
INTEGER *4 dpy, xid
```

Description

The **winX** subroutine converts the specified Enhanced X-Windows window into a GL window. The Enhanced X-Windows window must be created with the **XCreateWindow** subroutine from the Enhanced X-Windows library. The Enhanced X-Windows window can be mapped before or after conversion to a GL window, and can subsequently be unmapped or remapped.

Not all possible combinations of GL and Enhanced X-Windows calls result in defined behavior, however. In particular, avoid the following usages:

- Do not use Enhanced X-Windows drawing routines to draw into a GL window. The screen results are undefined. Use Enhanced X-Windows drawing routines to draw into Enhanced X-Windows only.
- Use only the GL **loadXfont** subroutine to load X fonts for use with the GL **charstr** subroutine. Use the **XLoadFont** or **XLoadQueryFont** subroutine only if you plan to use X text-rendering routines to draw into an Enhanced X-Windows window.
- Do not change, modify, or delete GL internal properties associated with GL windows. Doing so causes unpredictable behavior.
- Do not grab or ungrab pointers, keyboards, or the server while in GL fullscreen mode. Doing so causes unpredictable behavior.
- Events associated with GL windows can be obtained through the Enhanced X-Windows event queue. However, if you choose to obtain events in this manner, do not also use the GL event queue. Doing so causes unpredictable results. An application can use both the GL and Enhanced X-Windows event queues within the same executable if the GL queue is used only for GL events and the Enhanced X-Windows event queue is used only for Enhanced X-Windows events.

The previous brief restrictions are explained more fully in Using Enhanced X-Windows Calls with GL Subroutines in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

To use the **winX** subroutine correctly, the Enhanced X-Windows window must have been created with the **XCreateWindow** subroutine. The following conditions must also be met within the **XCreateWindow** attributes:

- The visual specified with this subroutine *must* be the largest available PseudoColor visual. Failure to use the largest possible PseudoColor visual results in unpredictable behavior.
- The window depth must match the depth of the visual. If it does not, a BadMatch event error is generated.
- The window attributes must include a color map that was created with this large visual. Specifying a color map of the wrong size causes a BadMatch event error to be generated. Failure to specify a color map that has been created with the large visual results in unpredictable behavior.
- Do not use the default attributes for the border_pixel and the back_pixel attributes, because the default CopyFromParent does not have the correct visual for the border and background pixel maps. Using the incorrect border and back attributes generates a BadMatch event error. It is acceptable to specify `None` for these attributes. Please refer to example programs in the **/usr/lpp/GL/examples** directory for usage.

If only the Enhanced X-Windows ID of a GL window is needed, use the **getXdpy** and **getXwid** subroutines.

A window *must* be converted to a DirectAccess window (with either the **winX** or **winopen** subroutine) before any GL drawing can occur in it. DirectAccess allows the GL drawing library to write rendering commands directly, without intervening layers, to the MicroChannel address space of the graphics adapter. Thus, a Direct Access client *must* execute locally; that is, a GL program must execute on the machine containing the display adapter. Currently, a GL program cannot be operated over a network.

Parameters

dpy Specifies the Enhanced X-Windows connection (as returned by the **XOpenDisplay** subroutine).

xid Specifies the Enhanced X-Windows window ID (as returned by the **XCreateWindow** subroutine) of the window to be converted into a GL window.

Return Value

The GL window identifier.

Example

An example program showing the usage of the **winX** subroutine can be found in the **/usr/lpp/GL/examples** directory.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Because the windowing design on the GXT1000 has changed from that supported on previous graphics adapters, the rules for using **winX** have changed. The **gconfig** subroutine cannot be used to change the mode (color index mode to RGB mode) of a window created with **winX** on the GXT1000. The X11 window must be created in advance with the appropriate visual type that is desired by the application. Choose a PseudoColor visual if color-index mode GL rendering is desired; choose a DirectColor visual if RGB mode operation is desired.

The GLC_CREATE_OVERLAY token on the **glcompat** subroutine modifies the operation of the **winX** routine.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.
/usr/include/X11/Xlib.h	Contains C language constant and variable type definitions for version X11 of Enhanced X-Windows.

Related Information

The **charstr** subroutine, **getXdpy** subroutine, **getXwid** subroutine, **loadXfont** subroutine, **winopen** subroutine.

The **XOpenDisplay** subroutine, **XCreateWindow** subroutine, **XLoadFont** subroutine, **XLoadQueryFont** subroutine.

Using Enhanced X-Windows Calls with GL Subroutines in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

wmpack Subroutine

Purpose

Specifies RGBA writemask with a single packed integer.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void wmpack(Uint32  pack)
```

FORTRAN Syntax

```
SUBROUTINE WMPACK(pack)
INTEGER*4 pack
```

Description

The **wmpack** subroutine changes the current RGBA writemask. Bytes 0, 1, 2, and 3 are alpha, blue, green, and red, respectively (red is the least significant byte in the packed integer, then green, blue, and alpha). Components must range from 0 through 255.

For example,

```
wmpack(0xFF004080);
```

sets red mask to 0x80, green mask to 0x40, blue mask to 0, and alpha mask to 0xFF.

> **Note:** You can load alpha values only if the graphics adapter has alpha bitplanes.

You can also use the **wmpack** subroutine in color map mode to specify writemasks of more than 12 bits. This is useful for certain z-buffer and smoothline applications.

Parameter

pack Specifies a packed integer containing the RGBA writemask.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER GXT1000 adapter supports only on/off control of individual R, G, and B bitplanes. Individual bits cannot be masked in the color planes.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the current writemask with the **getwritemask** subroutine.

Returning the current RGB writemask with the **gRGBmask** subroutine.

Granting write access to a subset of available bitplanes with the **RGBwritemask** subroutine.

Granting write permission to a subset of available bitplanes with the **writemask** subroutine.

Configuring the Frame Buffer, Removing Hidden Surfaces, and Smoothing Jagged Lines with Antialiasing.

Writemasks and Logical Operations and Working in Color Map and RGB Modes.

writemask Subroutine

Purpose
Grants write permission to available bitplanes.

Libraries
Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax
```
void writemask(Colorindex  writem)
```

FORTRAN Syntax
```
SUBROUTINE WRITEM(writem)
INTEGER*4 writem
```

Description
The **writemask** subroutine sets the bitplane writemask in color map mode. Writemasks are used to shield portions of the frame buffer from being written into. A writemask is a small set of bits (8 bits or 12 bits, depending on the frame buffer configuration), one bit for each bitplane of the frame buffer.

If a bit is set (1), the corresponding bitplane is enabled for writing, and any routine that draws into the frame buffer will be able to write into that bitplane. If a bit is reset (0), the corresponding bitplane is marked as read only, and the values stored in that bitplane are not changed.

Note that the writemask protects planes in the color frame buffer. Thus, writemasks essentially prevent certain colors from being written into the frame buffer. Colors that are drawn while a writemask is enabled appear different, depending on the color, the writemask, and the color value currently stored in the frame buffer (at a given pixel).

Writemasks are useful for emulating overlay/underlay planes.

> **Note:** This subroutine is intended for use only in color map mode. To set the writemask in RGB mode, use the **RGBwritemask** subroutine.

Parameter
writem Specifies the mask that controls which bitplanes are available for drawing and which are read only. The mask contains one bit per available bitplane.

Example
The example C language program **vlsi.c** uses the **writemask** subroutine to draw a rectangle in one of four colors by enabling writing into only one of four bitplanes.

The example C language program **circuit.c** uses the **writemask** subroutine to draw power circuitry into several bitplanes.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying the target frame buffer for drawing subroutines with the **drawmode** subroutine.

Returning the current writemask with the **getwritemask** subroutine.

Granting write access to a subset of available bitplanes with the **RGBwritemask** subroutine.

Specifying the RGBA writemask with a single packed integer with the **wmpack** subroutine.

Configuring the Frame Buffer.

Writemasks and Logical Operations, and Working in Color Map and RGB Modes.

---

writepixels Subroutine

Purpose

Paints a row of pixels on the screen in color map mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void writepixels(Int16  number, Colorindex  colors[ ])
```

FORTRAN Syntax

```
SUBROUTINE WRITEP(number, colors)
INTEGER*4 number
INTEGER*2 colors(number)
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **writepixels** subroutine paints a row of pixels on the screen in color map mode.The system reads elements from the *colors* array and writes a pixel of the appropriate color for each.

The starting location for the row of pixels 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. The current character position becomes undefined if the new position is outside the viewport.

The **writepixels** subroutine does not automatically wrap from one line to the next. It can be used in both single and double buffer modes.

The **rectwrite** subroutine provides significantly better performance for pixel block transfers. Even when only one row of pixels needs to be read, use the **rectwrite** subroutine.

1. This subroutine cannot be used to add to a display list.
2. The use of this subroutine is deprecated. Do not use the **writepixels** subroutine in new development.

Parameters

number	Specifies the number of pixels to paint.
colors	Specifies an array of color indexes.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the value of specific pixels in color map mode with the **readpixels** subroutine.

Copying a rectangle of pixels with an optional zoom with the **rectcopy** subroutine.

Reading a rectangular array of pixels into host memory with the **rectread** subroutine.

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** subroutine.

Painting a row of pixels on the screen in RGB mode with the **writeRGB** subroutine.

Graphics Library Overview, Configuring the Frame Buffer, Creating Animated Scenes, Reading and Writing Pixels, Using Viewports and Screenmasks, and Working in Color Map and RGB Modes.

---

writeRGB Subroutine

Purpose

Paints a row of pixels on the screen in RGB mode.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void writeRGB
(Int16  number,
RGBvalue  red[ ], RGBvalue  green[ ], RGBvalue  blue[ ])
```

FORTRAN Syntax

```
SUBROUTINE WRITER(number, red, green, blue)
INTEGER*4 number
CHARACTER*(*) red, green, blue
```

> **Note:** For FORTRAN users, this subroutine accepts long integer parameters (INTEGER*4) when invoked from a FORTRAN program, although it accepts short integers when invoked from a C program. The C and FORTRAN syntax shown here reflect this difference.

Description

The **writeRGB** subroutine paints a row of pixels on the screen in RGB mode.The system reads elements from the arrays specified in the *red*, *green*, and *blue* parameters and writes a pixel of the appropriate color for each.

The starting location for the row of pixels 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. The current character position becomes undefined if the new position is outside the viewport.

The **writeRGB** subroutine does not automatically wrap from one line to the next. It supplies a 24-bit RGB value (8 bits for each color) for each pixel. This value is written directly into the bitplanes.

> **Note:** When there are only 12 color bitplanes available, the lower 4 bits of each color are ignored.

The **rectwrite** subroutine provides significantly better performance for pixel block transfers. Even when only one row of pixels needs to be read, use the **rectwrite** subroutine.

1. This subroutine is available only in RGB mode.
2. This subroutine cannot be used to add to a display list.
3. The use of this subroutine is deprecated. Do not use the **writeRGB** subroutine in new development.

Parameters

number	Specifies the number of pixels to paint.
red	Specifies an array containing red values for each pixel to paint.
green	Specifies an array containing green values for pixel to paint.
blue	Specifies an array containing blue values for each pixel to paint.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Returning the value of specific pixels in RGB mode with the **readRGB** subroutine.

Copying a rectangle of pixels with an optional zoom with the **rectcopy** subroutine.

Reading a rectangular array of pixels into host memory with the **rectread** subroutine.

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** subroutine.

Painting a row of pixels on the screen in color map mode with the **writepixels** subroutine.

Graphics Library Overview, Reading and Writing Pixels, Using Viewports and Screenmasks, and Working in Color Map and RGB Modes.

---

zbuffer Subroutine

Purpose

Enables or disables the z-buffer for storing depth information.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void zbuffer(Int32  bool )
```

FORTRAN Syntax

```
SUBROUTINE ZBUFFE(bool)
LOGICAL bool
```

Description

The **zbuffer** subroutine enables or disables the z-buffer for storing depth information. Each pixel has an associated *z* value. To draw a pixel, the system compares the new *z* value with the existing one. If the new value is less than or equal to the existing value, the **zbuffer** subroutine stores a new color and *z* value in the bitplanes.

Drawing to the z-buffer with the **zdraw** subroutine must be disabled in order for the z-buffer to be enabled.

Parameter

bool Specifies a value for the state of drawing to the z-buffer. The settings for the *bool* parameter are:
True = drawing to the z-buffer is enabled.
False = drawing to the z-buffer is disabled.

Example

The example C language programs **zbuffer1.c** and **zbuffer2.c** enable z-buffering with the **zbuffer** subroutine to disable the display of hidden surfaces.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

The POWER Gt4, and POWER Gt4x, and POWER GXT1000 adapters do not support hidden-line and hidden-surface removal when rendering into the overlay planes. The z-buffer is automatically disabled when drawmode (OVERDRAW) is specified. The z-buffer is automatically enabled when drawmode (NORMALDRAW) is specified.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Determining whether z-buffering is on or off with the **getzbuffer** subroutine.

Setting a depth range with the **lsetdepth** subroutine.

Clearing the z-buffer with the **zclear** subroutine.

Enabling drawing to the z-buffer with the **zdraw** subroutine.

Specifying the function used for depth comparison with the **zfunction** subroutine.

Selecting depth or color as the source for *z* comparisons with the **zsource** subroutine.

Specifying which bits of the z-buffer are written during normal z-buffer operation with the **zwritemask** subroutine.

Graphics Library Overview, Configuring the Frame Buffer, and Removing Hidden Surfaces.

---

zclear Subroutine

Purpose

Initializes the z-buffer.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void zclear( )
```

FORTRAN Syntax

```
SUBROUTINE ZCLEAR
```

Description

The **zclear** subroutine loads the z-buffer with the largest possible positive integer. If the default value of the z comparison function is used (set by the **zfunction** subroutine), the zbuffer, when cleared to the largest possible value, can be used for basic z-buffering.

Only the z-buffer behind the area inside the current screenmask is cleared. The screenmask cannot be made larger than the window. Note that by default the screenmask is exactly the same size as the window.

The **scrmsk** subroutine can be used to change the size of the screenmask. Note also that the **viewport** subroutine resets the screenmask to be precisely the same size as the viewport.

> **Note:** The operation of this subroutine for the Model 730 Supergraphics Processor is modified. (See ″Hardware Considerations″.)

Example

The example C language programs **zbuffer1.c** and **zbuffer2.c** use the **zclear** subroutine to initialize the z-buffer.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Clearing the z-buffer and the color bitplanes simultaneously with the **czclear** subroutine.

Removing Hidden Surfaces.

Understanding the Hardware Used by GL, Clearing, Resetting, and Initializing GL, and Configuring the Frame Buffer.

---

zdraw Subroutine

Purpose

Enables or prohibits drawing to the z-buffer.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void zdraw(Int32 *bool***)**

FORTRAN Syntax

SUBROUTINE ZDRAW(*bool***)**
LOGICAL *bool*

Description

The **zdraw** subroutine allows or prohibits drawing into the z-buffer. The current color mode (either color map or RGB) applies. All current drawing attributes apply as well (color or RGBcolor, writemask or RGBwritemask, pattern, linestyle). However, if you enable drawing to the z-buffer with the **zdraw** subroutine, you must turn z-buffer mode off with the **zbuffer** subroutine. By default, drawing into the z-buffer is prohibited.

Calling the **gconfig** subroutine disables drawing to the z-buffer.

Parameter

bool Returns the state of drawing into the z-buffer. The settings for the *bool* parameter are:
True = drawing into the z-buffer is enabled.
False = drawing into the z-buffer is disabled.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

On the 3-D Color Graphics Processor and on the POWER Gt4, POWER Gt4x, and POWERgraphics GTO, and POWER GXT1000 adapters, calling the **zdraw** subroutine with a value of True affects only the **writepixels**, **writeRGB**, **rectwrite**, **lrectwrite**, and **rectcopy** pixel writing subroutines. If drawing into the z-buffer is enabled when other drawing subroutines (such as those for polygons or lines) are issued, the drawing subroutines draw as normal into the color bitplanes and write depth values into the z-buffer.

BLITs to and from the z-buffer are currently inoperational on the POWER GXT1000 adapter.

On the 3-D Color Graphics Processor, calling the **frontbuffer** subroutine with a value of True also prohibits drawing to the z-buffer.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Enabling drawing to the back buffer with the **backbuffer** subroutine.

Enabling drawing to the front buffer with the **frontbuffer** subroutine.

Graphics Library Overview, Configuring the Frame Buffer, and Removing Hidden Surfaces.

---

zfunction Subroutine

Purpose

Specifies the function used for depth comparison.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void zfunction(Int32  func)
```

FORTRAN Syntax

```
SUBROUTINE ZFUNCT(func)
INTEGER*4 func
```

Description

The **zfunction** subroutine compares the *z* value of the current contents (destination value) of a pixel against the *z* value for the contents that you want to write to that pixel (source, or incoming, value).

For example, if the *func* parameter is ZF_LESS and if the source *z* is less than the destination *z*, the system overwrites the destination pixel value with the source pixel value.

> **Note:** The operation of this subroutine for the Supergraphics Processor Subsystem is modified. (See ″Hardware Considerations″.)

Parameter

func Expects one of eight possible flags used when comparing *z* values. The available flags are:

C	FORTRAN	Description
ZF_NEVER	ZFNEVE	Never overwrite the destination pixel value.
ZF_LESS	ZFLESS	Overwrite the destination pixel value if the *z* of the source pixel value is less than the z of destination value.
ZF_EQUAL	ZFEQUA	Overwrite the destination pixel value if the *z* of the source pixel value is equal to the z of destination value.
ZF_LEQUAL	ZFLEQU	Overwrite the destination pixel value if the z of the source pixel value is less than or equal to the z of destination value. (This is the default value.)
ZF_GREATER	ZFGREA	Overwrite the destination pixel value if the z of the source pixel value is greater than the z of destination value.
ZF_NOTEQUAL	ZFNOTE	Overwrite the destination pixel value if the z of the source pixel value is not equal to the z of destination value.
ZF_GEQUAL	ZFGEQU	Overwrite the destination pixel value if the z of the source pixel value is greater than or equal to the z of destination value.

ZF_ALWAYS	ZFALWA	Always overwrite the destination pixel value.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Clearing the color bitplanes and the z-buffer simultaneously with the **czclear** subroutine.

Initiating z-buffer mode with the **zbuffer** subroutine.

Selecting depth or color as the source for **z** comparisons with the **zsource** subroutine.

Graphics Library Overview, Writemasks and Logical Operations, and Removing Hidden Surfaces.

---

zsource Subroutine

Purpose

Selects either depth or color as the source for *z* comparisons.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

```
void zsource(Int32  source)
```

FORTRAN Syntax

```
SUBROUTINE ZSOURC(source)
INTEGER*4 source
```

Description

The **zsource** subroutine selects either depth or color as the source for *z* comparisons . After a call to the **gbegin**, **ginit**, **greset**, or **winopen** subroutine, the default z-buffering is done with depth (z) values. In certain cases, it is desirable to buffer with respect to color values, especially the color index values generated by the smoothline hardware.

When the *source* parameter is ZSRC_DEPTH, the z-buffer operation is normal. When the *source* parameter is ZSRC_COLOR, however, source and destination color values are compared to determine which pixels the system draws.

Notes:

1. The Supergraphics Processor and the POWER GXT1000 adapter support only the ZSRC_DEPTH setting. The ZSRC_COLOR setting has no effect and is ignored.
2. This subroutine is not supported on the POWER Gt4 and POWER Gt4x adapters. On these adapters, the function provided by this subroutine is superseded by the LO_MIN and LO_MAX values of the **logicops** subroutine.
3. The POWERgraphics GXT1000 does not support the **zsource** subroutine.

The Z-Source Diagram shows the implementation of the z-source function.

Parameter

source
Specifies one of two possible flags:

C	FORTRAN	Description
ZSRC_COLOR	ZSRCCO	Buffering done by color value.
ZSRC_DEPTH	ZSRCDE	Buffering done by *z* value.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

Files

/usr/include/gl/gl.h Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h Contains FORTRAN constant and variable type definitions for GL.

Related Information

Specifying logical, arithmetic, or comparison operations for pixel updates with the **logicops** subroutine.

Initiating z-buffer mode with the **zbuffer** subroutine.

Enabling or disabling drawing to the z-buffer with the **zdraw** subroutine.

Specifying the function used for depth comparison with the **zfunction** subroutine.

Graphics Library Overview, Writemasks and Logical Operations, and Removing Hidden Surfaces.

---

zwritemask Subroutine

Purpose

Specifies the z-buffer writemask.

Libraries

Graphics Library

C (**libgl.a**)

FORTRAN (**libfgl.a**)

C Syntax

void zwritemask(Int32 *mask***)**

FORTRAN Syntax

SUBROUTINE ZWRITE(*mask***)**
INTEGER*4 *mask*

Description

The **zwritemask** subroutine specifies a write-enable mask on the z-buffer. If a bit in the mask is set (1), the corresponding bitplane is enabled for reading and writing, and any subroutine that updates the z-buffer can update that bitplane. If a bit in the mask is reset (0), the corresponding bitplane is marked as read-only, and the values stored in that bitplane are not updated.

Parameter

mask Specifies the mask indicating which z-buffer bits are read only and which can be overwritten. The following rules apply:
If a bit of the mask is set to 0, the corresponding bits are read-only.
If a bit of the mask is set to 1, the corresponding z-buffer bits can be overwritten.

Implementation Specifics

This subroutine is part of GL in the AIXwindows Environment/6000 Version 1, Release 2 with AIXwindows/3D Feature.

On the 3-D Color Graphics Processor, the z-buffer writemask is ignored when drawing is performed directly into the z-buffer; that is, when the value of the **zdraw** subroutine is True. Instead, the color-planes writemask is applied.

The 3-D Color Graphics Processor, and the Supergraphics Processor Subsystem, and the POWER GXT1000 adapter support only two values for the z-buffer writemask: 0x000000 and 0xffffff. All other values are ignored.

The POWER Gt4 and POWER Gt4x adapters support a z-buffer writemask in banks of 8 bitplanes. The eight supported values for the writemask are 0x000000, 0x0000ff, 0x00ff00, 0xff0000, 0x00ffff, 0xffff00, 0xff00ff, and 0xffffff. All other values are ignored.

If an unsupported writemask is specified with this subroutine, it is ignored, and the current writemask is not changed.

Files

/usr/include/gl/gl.h	Contains C language constant and variable type definitions for GL.
/usr/include/gl/fgl.h	Contains FORTRAN constant and variable type definitions for GL.

Related Information

Granting write access to a subset of available bitplanes with the **RGBwritemask** subroutine.

Enabling drawing to the z-buffer with the **zdraw** subroutine.

Specifying an RGBA writemask with a single, packed integer with the **wmpack** subroutine.

Granting write permission to available bitplanes with the **writemask** subroutine.

Graphics Library Overview, Writemasks and Logical Operations, and Removing Hidden Surfaces.

Chapter 2. GL Example Programs

Functional List of GL Example Programs

Select an example program from the following list:

Example programs for drawing curves and surfaces

curve1.c Example C Language program draws three curve segments

curve2.c Example C Language program draws joined curve segments

curve3.c Example C Language program draws a Bezier curve segment

curved.c Example C Language program is a simple curve editor

patch1.c Example C Language program draws three surface patches

Example programs demonstrating hidden surface removal

backface.c Example C Language program draws a cube

octahedron.c Example C Language program draws meshes

Example Lighting Programs

cylinder1.c Example C Language program demonstrates simple lighting

cylinder2.c Example C Language program illustrates multiple surface materials and multiple lights

localatten.c Example C Language program illustrates the effect of light attenuation

platelocal.c Example C Language program illustrates the use of local lights.

Example programs for fonts

font3.c Example C Language program defines a font with three characters

xfonts.c Example C Language program demonstrates how to use Enhanced X-Windows fonts in a GL application.

Example programs demonstrating antialiasing

alias_back.c Example C Language program illustrates antialiased lines of multiple foreground colors on a monochrome background

alias_fore.c Example C Language program illustrates antialiased lines a single foreground color on a multicolored background

alias.c Example C Language program illustrates antialiased lines

Other Example Programs

boxcirc.c Example C Language program draws a 2D box and circle

circuit.c Example C Language program draws power circuitry using writemasks.

colored.c Example C Language program edits a color map

doublebuff.c Example C Language program illustrates double-buffered animation

depthcue.c Example C Language program draws a depth-cued 3D wire frame cube

doily.c Example C Language program draws a doily based on given point count

draw.c Example C Language program is a minimal line drawing program

gamma.c Example C Language program shows the use of gamma ramps to correct the appearance of photographs and antialiased lines and to implement color-map tricks

iobounce.c Example C Language program draws a ball bouncing on a 2D surface

overlay.c Example C Language program demonstrates overlay bitplanes

paint.c Example C Language program demonstrates minimal object space painting

pick1.c Example C Language program demonstrates picking

popup.c Example C Language program demonstrates writing pop-up menu routines

prompt.c Example C Language program demonstrates standard and user-defined prompts

scrn_rotate.c Example C Language program illustrates object rotation

select1.c Example C Language program demonstrates selecting

setshade.c Example C Language program moves a smooth-shaded polygon in and out of the graphics port

sunflower.c Example C Language program draws a sunflower pattern from circles

text.c Example C Language program demonstrates text drawing

tpbig.c Example C Language program illustrates arcs, polygons, character strings, and textport

vlsi.c Example C Language program shows a simple vlsi graphical editor

worms.c Example C Language program

zbuffer1.c Example C Language program draws two intersecting planes and requires the z-buffer option

zbuffer2.c Example C Language program draws three tumbling cubical objects.

zoing.c Example C Language program draws a spiral from circles

---

alias.c Example C Language Program

```
#include <gl/gl.h>
#include <gl/device.h>
#include <stdlib.h>
#include <math.h>

/*
* This program illustrates the drawing of antialiased lines, of
* multiple foreground colors, on an arbitrary multi-colored
* background.
*
* Note that this style of drawing is limited as to the number
* of foreground and background colors. The total number of
* colors is one-sixteenth of the total number of color map
* entries.
*
* The basic idea behind the algorithm is to divide the
* available bitplanes into three groups: those containing
* the background image, those containing the foreground line
* colors and those containing the antialiasing coverage
* information. Color ramps are loaded in the color map ranges
* corresponding to the coverage information. The color ramps
* must blend between all possible foreground and all possible
* background colors.
*
* Note that proper gamma correction is absolutely vital to
* getting antialiased lines that look truly smooth to the user.
* The gamma exponent depends on the monitor (specifically, on
```

```
* the type of phosphors) and therefore needs to be tuned to the
* model of the monitor.
*
* Antialiased lines are supported on the 8 and 24 bit 3D adapters.
* Colormode antialiased lines is not supported on the
* High Speed Graphics Adapter.
*/
draw_fan()
{
  /* draw fan */

  int i;
  for (i=0; i<= 90; i+=10) {
    pushmatrix();
    rot (-(float) i, 'z');
    move (0.0, 0.0, 0.0);
    draw (0.0, 300.0, 0.0);
    popmatrix();
  }
}
/* experimentally determined gamma factor */
double gammy = 2.4;
/* color map data area */
short rramp[256], gramp[256], bramp[256];
/* a utility macro used to build the color ramp */
#define ADD_TO_RAMP(COL,back_r,back_g,back_b)             \
{                                                        \
  for (j=0; j<16; j++){                                  \
     col_idx = COL + j;                                  \
     rramp[col_idx] = (j*fore_r + (16-j)*back_r) / 16;    \
     gramp[col_idx] = (j*fore_g + (16-j)*back_g) / 16;    \
     bramp[col_idx] = (j*fore_b + (16-j)*back_b) / 16;    \
     corr = 255.0 * pow (((double) rramp[col_idx])/255.0, \
       1.0/gammy);                                       \
     rramp[col_idx] = (int) corr;                        \
     corr = 255.0 * pow (((double) gramp[col_idx])/255.0, \
       1.0/gammy);                                       \
     gramp[col_idx] = (int) corr;                        \
     corr = 255.0 * pow (((double) bramp[col_idx])/255.0, \
       1.0/gammy);                                       \
     bramp[col_idx] = (int) corr;                        \
  }                                                      \
}
/* a utility that blends the supplied foreground color
 * to several backgrounds */
void blend_background_cmap (int offset, short fore_r, short
                            fore_g, short fore_b)
{
  int j, col_idx;
  double corr;
    /* blend to background of green */
  ADD_TO_RAMP (offset, 0, 255, 0);
    /* blend to background of purple */
  ADD_TO_RAMP (offset+16, 155, 0, 255);
    /* blend to background of black */
  ADD_TO_RAMP (offset+32, 0, 0, 0);
    /* blend to background of cyan */
  ADD_TO_RAMP (offset+48, 20, 150, 150);
}
```

```
/* the main color map routine, builds cross-ramps between various
 * front and back colors */
void load_cross_ramp ()
{
    /* blend to foreground of white */
  blend_background_cmap (0, 255, 255, 255);
    /* blend to foreground of yellow */
  blend_background_cmap (64, 255, 255, 0);
    /* blend to foreground of red */
  blend_background_cmap (128, 255, 0, 0);
    /* blend to foreground of blue */
  blend_background_cmap (192, 0, 0, 255);
  mapcolors (0, 255, rramp, gramp, bramp);
}
#define RRRANDO ( (float) (400*rand()/32767));
/* This routine draws a background of random polygons.
 * One of four colors are selected randomly.  */
void draw_background()
{
  float vv[2];
  int i, j;
  srand (24515);
  color (0);
  clear();
    /* the writemask guarantees that drawing is occurring only
     * into the "background" bitplanes */
  writemask (0x30);
  for (j=0; j<12; j++) {
    color (16*((4*rand())/32767));
    bgnpolygon();
    for (i=0; i<6; i++) {
      vv[0] = RRRANDO;
      vv[1] = RRRANDO;
      v2f (vv);
    }
    endpolygon();
  }
}
main (argc, argv )
int argc;
char **argv;
{
  long l_lop;
  prefsize (400.0, 400.0);
  winopen ("antialiased lines");
  cmode();
  doublebuffer();
  gconfig();
#if 0
    /* zsource (ZSRC_COLOR) only High Performance 3-D adapter */
    /* enable update comparison to improve line intersections */
  zbuffer (TRUE);
  zclear();
    /* use color, not depth values, to determine if pixel is
     * written */
  zsource (ZSRC_COLOR);
    /* foreground colors always have greater values than
     * background */
  zfunction (ZF_GEQUAL);
    /* create the color ramp */
#endif
```

```
  save_cmap();
  load_cross_ramp();
    /* draw the random background */
  draw_background();
  swapbuffers ();
  draw_background();
    /* the writemask write-protects the background while lines
     * are being drawn */
  writemask (0xcf);
  if (argc==2) {
      l_lop=atol(argv[1]);
      logicop (l_lop);
      printf ("The logic op is: %ld\n",l_lop);
   }
    /* loop until the right mouse button is depressed */
  while (!getbutton(RIGHTMOUSE) && !getbutton(ESCKEY)) {
    color (0);
    clear ();
    pushmatrix();        /* draw normal, jaggy lines */
    linesmooth(FALSE);
    color (15); /* maximum shade of white */
    pushmatrix();
    translate (10.0, 10.0, 0.0);
    rotate( 500-getvaluator (MOUSEY), 'z');
    draw_fan();
    popmatrix();         /* draw smooth, antialiased lines -- lower right*/
    linesmooth(TRUE);
    color (64);  /* yellow */
    pushmatrix();
    translate (390.0, 30.0, 0.0);
    rotate( getvaluator (MOUSEY) + 400, 'z');
    draw_fan();
    popmatrix();            /* draw smooth, antialiased lines -- upper right */
    color (128);   /* red */
    pushmatrix();
    translate (390.0, 390.0, 0.0);
    rotate( getvaluator (MOUSEX) -2300, 'z');
    draw_fan();
    popmatrix();         /* draw smooth, antialiased lines -- upper left */
    color (192);   /* blue */
    pushmatrix();
    translate (30.0, 360.0, 0.0);
    rotate( -400 - getvaluator (MOUSEX), 'z');
    draw_fan();
    popmatrix();
    popmatrix();
    swapbuffers();
  }
  restore_cmap();
}
/*This saves the colormap*/
#define lo_end 0
#define hi_end 255
short *CarrayR, *CarrayG, *CarrayB;
save_cmap()
{
  CarrayR = calloc (lo_end+hi_end,sizeof(short));
  CarrayG = calloc (lo_end+hi_end,sizeof(short));
  CarrayB = calloc (lo_end+hi_end,sizeof(short));
  getmcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}
```

```
/*This restores the colormap*/
restore_cmap()
{
  mapcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}
/*
  Changes:
    -  Added the restoring of the colormap
      -  The program now reads the first parameter passed to it for the value
       to serve as the current logic operation
    -  The function calls that set up the z buffer for the z source color
       have been removed. (#if 0 / #endif)
    -  Added escape key and right mouse for clean exit
*/
```

Related Information

The **linesmooth** subroutine .

Pixel Coverage in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

alias_back.c Example C Language Program

```
#include <gl/gl.h>
#include <gl/device.h>
#include <stdlib.h>
#include <math.h>

/*
* This program illustrates the drawing of anti-aliased lines of
* multiple foreground colors on a monochrome background.
*
* Note that this style of drawing is limited to precisely one
* background color. The number of foreground colors is also
* limited;
* the maximum number of foreground colors is
* one-sixteenth of the total number of colormap entries.
*
* The basic idea behind the algorithm is to divide the available
* bitplanes into two groups: those containing the foreground line
* colors, and those containing the antialiasing coverage
* information.  A color-ramp is loaded in the color map ranges
* corresponding to the coverage information.  The foreground line
* drawing colors can be arbitrary.
*
* Note that proper gamma correction is absolutely vital to
* getting antialiased lines that look truly smooth to the user.
* The gamma exponent depends on the monitor (specifically, on the
* type of phosphors) and therefore needs to be tuned to the model
* of the monitor.  */

draw_fan()
{
  int i; /* draw fan */

  for (i=0; i<= 90; i+=10) {
    pushmatrix();
    rot (-(float) i, 'z');
    move (0.0, 0.0, 0.0);
    draw (0.0, 200.0, 0.0);
    popmatrix();
  }

}
```

```
#define AABLACK  0
#define AAGREEN  16
#define AARED    32
#define AAPURPLE 48
#define AAWHITE  64

/* experimentally determined gamma factor */
double gammy = 2.4;

/* a utility macro used to build the color ramp */
#define ADD_TO_RAMP(COL)                                                \
{                                                                       \
   for (j=0; j<16; j++) {                                               \
      col_idx = COL + j;                                                \
      rramp[col_idx] = (j*foreground_r + (16-j)*background_r) / 16;     \
      gramp[col_idx] = (j*foreground_g + (16-j)*background_g) / 16;     \
      bramp[col_idx] = (j*foreground_b + (16-j)*background_b) / 16;     \
      rramp[col_idx] = (int)                                            \
        (255.0 * pow (((double) rramp[col_idx])/255.0 , 1.0 / gammy));  \
      gramp[col_idx] = (int)                                             \
        (255.0 * pow (((double) gramp[col_idx])/255.0 , 1.0 / gammy));  \
      bramp[col_idx] = (int)                                            \
        (255.0 * pow (((double) bramp[col_idx])/255.0 , 1.0 / gammy));  \
   }                                                                    \
}

/* This routine creates an "random" color ramp and loads it */
void load_background_cmap (short background_r,
short background_g,
short background_b)
{
  int i, j, col_idx;
  int foreground_r, foreground_g, foreground_b;
  short rramp[256], gramp[256], bramp[256];
    /* update gamma correction factor */
  if (getbutton (WKEY)) gammy +=0.02;
  if (getbutton (SKEY)) gammy -=0.02;

   /* create color ramp */

  /* AABLACK */
  foreground_r = 0;
  foreground_g = 0;
  foreground_b = 0;
  ADD_TO_RAMP (AABLACK); /* AAGREEN */
  foreground_r = 0;
  foreground_g = 255;
  foreground_b = 0;
  ADD_TO_RAMP (AAGREEN); /* AARED */
  foreground_r = 255;
  foreground_g = 127;
  foreground_b = 0;
  ADD_TO_RAMP (AARED); /* AAPURPLE */
  foreground_r = 150;
  foreground_g = 25;
  foreground_b = 120;
  ADD_TO_RAMP (AAPURPLE); /* AAWHITE */
  foreground_r = 225;
  foreground_g = 235;
  foreground_b = 255;
  ADD_TO_RAMP (AAWHITE);
  mapcolors (0, 127, rramp, gramp, bramp);
}

#define RRRANDO ( (float) (400*rand()/32767))

/* This routine draws a background of random polygons.
 * One of five colors are selected randomly.  */

void draw_foreground()
{
  int i, j;
```

```
  color (AABLACK);
  clear();
  linesmooth(TRUE);
  translate (-100.0, -100.0, 0.0);
  for (j=0; j<9; j++) {
    color (16*(5*rand()/32767));
    pushmatrix();
    translate ( RRRANDO, RRRANDO, 0.0);
    draw_fan();
    popmatrix();
  }
}
main ()
{
  int i, j, k, iinc=1, jinc=3, kinc=7;
  prefsize (400.0, 400.0);
  winopen ("antialiased lines");
  cmode();
  singlebuffer();
  gconfig();
  save_cmap();
  /* create the color ramp--the background color will be black */
  load_background_cmap (0, 0, 0);
  /* draw multicolored line drawing */
  draw_foreground();
  /* loop until the right mouse button is depressed */
  while (!getbutton(RIGHTMOUSE) && !getbutton(ESCKEY)) {
    /* cycle through a random set of background colors */
    i+=iinc;
    j+=jinc;
    k+=kinc;
    if ((i+iinc>5*255) || (i+iinc<0)) {
      iinc = -iinc;
      i+=iinc;
    }
    if ((j+jinc>5*255) || (j+jinc<0)) {
      jinc = -jinc;
      j+=jinc;
    }
    if ((k+kinc>5*255) || (k+kinc<0)) {
      kinc = -kinc;
      k+=kinc;
    }
    if ( i%5 == 0) {
      load_background_cmap ( (i/5)%255, (j/5)%255,(k/5)%255);
    }
  }
  restore_cmap();
}
/*This saves the colormap*/
#define lo_end 0
#define hi_end 255
short *CarrayR, *CarrayG, *CarrayB;
save_cmap()
{
  CarrayR = calloc (lo_end+hi_end,sizeof(short));
  CarrayG = calloc (lo_end+hi_end,sizeof(short));
  CarrayB = calloc (lo_end+hi_end,sizeof(short));
  getmcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}
```

```
/*This restores the colormap*/
restore_cmap()
{
  mapcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}

/*
  Changes:
    -  Added the restoring of the colormap
    -  Added escape key and right mouse exiting cleanly
*/
```

Related Information

The **linesmooth** subroutine .

Pixel Coverage in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

alias_fore.c Example C Language Program

```
#include <gl/gl.h>
#include <gl/device.h>
#include <stdlib.h>
#include <math.h>

/*
 * This program illustrates the drawing of antialiased lines, of
 * a single (foreground) color,  on an arbitrary multicolored
 * background.
 *
 * Note that this style of drawing is limited to precisely one
 * foreground color.
 * The number of background colors is also limited; the maximum
 * number of background colors is one-sixteenth of the total
 * number of colormap entries.
 *
 * The basic idea behind the algorithm is to divide the available
 * bitplanes into two groups: those containing the background
 * image, and those containing the anti-aliasing coverage
 * information.  A color-ramp is loaded in the color map ranges
 * corresponding to the coverage information.  The background
 * image colors can be arbitrary.
 *
 * Note that proper gamma correction is absolutely vital to
 * getting antialiased lines that look truly smooth to the user.
 * The gamma exponent depends on the monitor (specifically, on the
 * type of phosphors) and therefore needs to be tuned to the model
 * of the monitor.
 */

draw_fan()
{
  int i;   /* draw fan */

  for (i=0; i<= 90; i+=10) {
    pushmatrix();
    rot (-(float) i, 'z');
    move (0.0, 0.0, 0.0);
    draw (0.0, 300.0, 0.0);
    popmatrix();
  }
}

#define AABLACK  0
#define AAGREEN  16
#define AARED    32
#define AAPURPLE 48
#define AAWHITE  64
```

```
/* experimentally determined gamma factor */
double gammy = 2.4;

/* a utility macro used to build the color ramp */
#define ADD_TO_RAMP(COL)                                                    \
{                                                                           \
   for (j=0; j<16; j++) {                                                   \
      col_idx = COL + j;                                                    \
      rramp[col_idx] = (j*foreground_r + (16-j)*background_r) / 16;         \
      gramp[col_idx] = (j*foreground_g + (16-j)*background_g) / 16;         \
      bramp[col_idx] = (j*foreground_b + (16-j)*background_b) / 16;         \
      rramp[col_idx] = (int)                                                \
         (255.0 * pow (((double) rramp[col_idx])/255.0 , 1.0 / gammy));     \
      gramp[col_idx] = (int)                                                \
         (255.0 * pow (((double) gramp[col_idx])/255.0 , 1.0 / gammy));     \
      bramp[col_idx] = (int)                                                \
         (255.0 * pow (((double) bramp[col_idx])/255.0 , 1.0 / gammy));     \
   }                                                                        \
}

/* this routine creates an "random" color ramp and loads it */
void load_foreground_cmap (short foreground_r,
short foreground_g,
short foreground_b)
{
  int i, j, col_idx;
  int background_r, background_g, background_b;
  short rramp[256], gramp[256], bramp[256];

    /* update gamma correction factor */
  if (getbutton (WKEY)) gammy +=0.02;
  if (getbutton (SKEY)) gammy -=0.02;

    /* create color ramp */
  /* AABLACK */
  background_r = 0;
  background_g = 0;
  background_b = 0;
  ADD_TO_RAMP (AABLACK);   /* AAGREEN */
  background_r = 0;
  background_g = 255;
  background_b = 0;
  ADD_TO_RAMP (AAGREEN);   /* AARED */
  background_r = 255;
  background_g = 127;
  background_b = 0;
  ADD_TO_RAMP (AARED);   /* AAPURPLE */
  background_r = 150;
  background_g = 25;
  background_b = 120;
  ADD_TO_RAMP (AAPURPLE);   /* AAWHITE */
  background_r = 225;
  background_g = 235;
  background_b = 255;
  ADD_TO_RAMP (AAWHITE);
  mapcolors (0, 127, rramp, gramp, bramp);
}

#define RRRANDO ( (float) (400*rand()/32767));

/* This routine draws a background of random polygons.
 * One of five colors are selected randomly.  */
void draw_background() {
  float vv[2];
  int i, j;
  srand (234515);
  color (AABLACK);
  clear();
```

```
    /* the writemask guarentees that drawing is occuring only
     * into the "background" bitplanes */
  writemask (0xf0);
  for (j=0; j<12; j++) {
    color (16*(5*rand()/32767));
    bgnpolygon();
    for (i=0; i<6; i++) {
      vv[0] = RRRANDO;
      vv[1] = RRRANDO;
      v2f (vv);
    }
    endpolygon();
  }
}
main ()
{
  int i, j, k, iinc=1, jinc=3, kinc=7;
  prefsize (400.0, 400.0);
  winopen ("antialiased lines");
  cmode();
  doublebuffer();
  gconfig();
  save_cmap();
  /* create the color ramp --- the foreground color will be white */
  load_foreground_cmap (255, 255, 255);
    /* draw the random background */
  draw_background();
  swapbuffers ();
  draw_background();
    /* the writemask write-protects the background while the lines
     * are being drawn */
  writemask (0x0f);
    /* loop until the right mouse button is depressed */
  while (!getbutton(RIGHTMOUSE) && !getbutton(ESCKEY)) {
    color (0);
    clear ();
    pushmatrix();
      /* draw normal, jaggy lines */
    linesmooth(FALSE);
    color (15);
    pushmatrix();
    translate (10.0, 10.0, 0.0);
    rotate( 500-getvaluator (MOUSEY), 'z');
    draw_fan();
    popmatrix();
      /* draw smooth, anti-aliased lines */
    linesmooth(TRUE);
    pushmatrix();
    translate (390.0, 30.0, 0.0);
    rotate( getvaluator (MOUSEY) + 400, 'z');
    draw_fan();
    popmatrix();
    popmatrix();
    swapbuffers();
      /* cycle through a random set of foreground colors */
    i+=iinc;
    j+=jinc;
    k+=kinc;
    if ((i+iinc>5*255) || (i+iinc<0)) {
      iinc = -iinc;
      i+=iinc;
    }
```

```
    if ((j+jinc>5*255) || (j+jinc<0)) {
      jinc = -jinc;
      j+=jinc;
    }
    if ((k+kinc>5*255) || (k+kinc<0)) {
      kinc = -kinc;
      k+=kinc;
    }
    if ( i%5 == 0) {
      load_foreground_cmap ( (i/5)%255, (j/5)%255,  (k/5)%255);
    }
  }
  restore_cmap();
}
/*This saves the colormap*/
#define lo_end 0
#define hi_end 255
short *CarrayR, *CarrayG, *CarrayB;
save_cmap()
{
  CarrayR = calloc (lo_end+hi_end,sizeof(short));
  CarrayG = calloc (lo_end+hi_end,sizeof(short));
  CarrayB = calloc (lo_end+hi_end,sizeof(short));
  getmcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}
/*This restores the colormap*/
restore_cmap()
{
  mapcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}
/*
  Changes:
    -  Added the restoring of the colormap
    -  Added escape key and right mouse for clean exit
*/
```

Related Information

The **linesmooth** subroutine .

Pixel Coverage in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

backface.c Example C Language Program

```
/*
backface.c:
Draw a cube that can run with backface() turned on or
off.  Turn backface() on with the F key.
Turn backface() off with the B key.  Cube is moved
when LEFTMOUSE is pressed and mouse itself is moved.
*/
#include <gl/gl.h>
#include <gl/device.h>
#define CUBE_SIZE 200
#define CUBE_OBJ 1
main ()
{
```

```
  Device  dev;
  int      moveit;
  short    val, x = 30,y = 30;

  initialize();

  while (TRUE) {

    while (qtest()) {

      dev = qread(&val);

      if (dev == ESCKEY) {
        backface(FALSE);
        gexit();
        exit();
      }
      else if (dev == REDRAW) {
        reshapeviewport();
        drawcube(x,y);
      }
      else if (dev == LEFTMOUSE)
        moveit = val;
        /* left mouse is down */
      else if (dev == BKEY) {
        backface(TRUE);
        /* turn back facing off */
        drawcube(x,y);
      }
      else if (dev == FKEY) {
        backface(FALSE);
        /* turn back facing on */
        drawcube(x,y);
      }

    }

    if (moveit) {
      x = getvaluator(MOUSEX);
      y = getvaluator(MOUSEY);
      drawcube(x,y);
    }

  }
}

initialize()
{
  int gid;

  prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4,
      YMAXSCREEN/4, YMAXSCREEN*3/4);
  gid = winopen("backface");

  doublebuffer();
  gconfig();
  shademodel(FLAT);

  ortho((float)-XMAXSCREEN, (float)XMAXSCREEN,
      (float)-YMAXSCREEN, (float)YMAXSCREEN,
      (float)-YMAXSCREEN, (float)YMAXSCREEN);

  qdevice(ESCKEY);
  qdevice(REDRAW);
  qdevice(LEFTMOUSE);
  qdevice(BKEY);
  qdevice(FKEY);
  qenter(REDRAW,gid);

  backface(TRUE);
  /* turn on back facing polygon removal */

}
```

```
/* define a cube */
cube()
{
  /* front face */
  pushmatrix();
  translate(0.0,0.0,CUBE_SIZE);
  color(RED);
  rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
  popmatrix();  /* right face */
  pushmatrix();
  translate(CUBE_SIZE, 0.0, 0.0);
  rotate(900, 'y');
  color(GREEN);
  rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
  popmatrix();  /* back face */
  pushmatrix();
  translate(0.0, 0.0, -CUBE_SIZE);
  rotate(1800, 'y');
  color(BLUE);
  rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
  popmatrix();  /* left face */
  pushmatrix();
  translate(-CUBE_SIZE, 0.0, 0.0);
  rotate(-900, 'y');
  color(CYAN);
  rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
  popmatrix();  /* top face */
  pushmatrix();
  translate(0.0, CUBE_SIZE, 0.0);
  rotate(-900, 'x');
  color(MAGENTA);
  rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
  popmatrix();  /* bottom face */
  pushmatrix();
  translate(0.0, -CUBE_SIZE, 0.0);
  rotate(900, 'x');
  color(YELLOW);
  rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
  popmatrix();
}
drawcube(x,y)
short x,y;
{
  pushmatrix();
  rotate(2*x, 'x');
  rotate(2*y, 'y');
  color(BLACK);
  clear();
  cube();
  popmatrix();
  swapbuffers();
}
/*Changes
   -The constant CUBE_SIZE had a .0 append to the end of it
*/
```

Related Information

The **backface** subroutine, **ortho** or **ortho2** subroutine, **popmatrix** subroutine, **pushmatrix** subroutine, **rotate** subroutine, **shademodel** subroutine, **swapbuffers** subroutine, **translate** subroutine .

Removing Hidden Surfaces in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

boxcirc.c Example C Language Program

```
/*
boxcirc.c:

A simple example which draw a 2d box and circle, press ESCape key
to exit.
*/

#include <gl/gl.h>
#include <gl/device.h>

main()
{
   int   dev,val;

   initialize();

   while (TRUE) {
      if (qtest()) {
         dev = qread(&val);
         if (dev == ESCKEY) {
            gexit();
            exit();
         }
         else if (dev == REDRAW) {
            reshapeviewport();
            drawboxcirc();
         }
      }
   }
}

initialize()
{
   int gid;

   prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4, YMAXSCREEN/4,
       YMAXSCREEN*3/4);

   gid = winopen("boxcirc");
   qdevice(ESCKEY);
   qdevice(REDRAW);
   qenter(REDRAW,gid);

}

drawboxcirc()
{
   pushmatrix();
   translate(200.0, 200.0, 0.0);
   color(BLACK);
   clear();
   color(BLUE);
   recti(0, 0, 100, 100);
   color(RED);
   circi(50, 50, 50);
   popmatrix();

}
```

Related Information

The **circ** subroutine, **clear** subroutine, **color** or **colorf** subroutine .

circuit.c Example C Language Program

Here is a very simple complete program that draws
power and ground circuitry. The interface consists of
the keys P draw power rectangles, G draw ground
rectangles, C clear the window, and Q quit. To draw a
rectangle, press down the left mouse button at one
corner, hold it down and slide to the other corner,
and release it.

When you use the program, be sure to exit by typing Q.
This resets the four lowest entries in the color map
(which are used by all the windows) back to the default
values.
*/

```
#include <gl/gl.h>
#include <gl/device.h>

#define BACKGROUND      0       /* = 00 */
#define POWER           1       /* = 01 */
#define GROUND          2       /* = 10 */
#define SHORT           3       /* = 11 */

long    xorg, yorg, xsize, ysize;
main()
{
  short val, drawtype;
  short x1, y1, x2, y2;
  drawtype = GROUND;
  prefposition (0,800,0,800);
  winopen("circuit");
  winconstraints();
  getorigin(&xorg, &yorg);
  getsize (&xsize,&ysize);
  ortho2 (-0.5,(Coord)xsize,-0.5,(Coord)ysize);
  /*Allow any size to work*/
  gconfig();
  mapcolor(0, 255, 255, 255);
  mapcolor(1, 0, 0, 255);
  mapcolor(2, 0, 0, 0);
  mapcolor(3, 255, 0, 0);
  qdevice(PKEY);                /* draw power rectangles */
  qdevice(GKEY);                /* draw ground rectangles */
  qdevice(CKEY);                /* clear screen */
  qdevice(QKEY);                /* quit */
  qdevice(REDRAW);
  qdevice(LEFTMOUSE); /* mark rectangle corners */
  tie(LEFTMOUSE, MOUSEX, MOUSEY);
  color(0);
  clear();
  while (1)
    switch (qread(&val)) {
    case PKEY:
      drawtype = POWER;
      break;
    case GKEY:
      drawtype = GROUND;
      break;
    case REDRAW:
      reshapeviewport();
      getorigin(&xorg, &yorg);
      getsize (&xsize,&ysize);
      ortho2 (-0.5,(Coord)xsize,-0.5,(Coord)ysize);
      clearscreen();
      gconfig();
    case CKEY:
      clearscreen();
      break;
```

```
    case QKEY:                /* restore default colors */
      mapcolor(0, 0, 0, 0);
      mapcolor(1, 255, 0, 0);
      mapcolor(2, 0, 255, 0);
      mapcolor(3, 255, 255, 0);
      return;
    case LEFTMOUSE:
      qread(&x1);
      qread(&y1);
      qread(&val);
      qread(&x2);
      qread(&y2);
      if (drawtype == POWER)
        powerrect(x1-xorg, y1-yorg,
            x2-xorg, y2-yorg);
      else
        groundrect(x1-xorg, y1-yorg,
            x2-xorg, y2-yorg);
      break;
    }
}
clearscreen()
{
  writemask(3);                /* clear window before drawing. */
  color(BACKGROUND);
  clear();
}
powerrect(x1, y1, x2, y2)
short x1, y1, x2, y2;
{
  writemask(1);
  color(1);
  rectfs(x1, y1, x2, y2);
}
groundrect(x1, y1, x2, y2)
short x1, y1, x2, y2;
{
  writemask(2);
  color(2);
  rectfs(x1, y1, x2, y2);
}
/*
   Changes:
      -The program now starts from the default size window
         prefposition (0,800,0,800);
         getsize (&xsize,&ysize);
         qdevice(REDRAW);
      -The program now handles resize/redraw
            case REDRAW:
               reshapeviewport();
               getorigin(&xorg, &yorg);
               getsize (&xsize,&ysize);
               ortho2 (-0.5,(Coord)xsize,-0.5,(Coord)ysize);
               clearscreen();
               gconfig();
*/
```

Related Information

The **writemask** subroutine .

Writemasks and Logical Operations in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

colored.c Example C Language Program

```
/*Example colored.c

Edit the colormap and display the results in the graphics window.

This program works on any 3d adapter.
*/

#include <gl/gl.h>
#include <gl/device.h>

#define START 64
#define CURRENTCOLOR 63
#define BARWIDTH 67
#define REDBAR   934
#define GREENBAR 800
#define BLUEBAR  666
#define STARTBAR 250
#define ENDBAR   1082
#define indextovalue(index) (4*index + 3)

short redindex = 0, greenindex = 0, blueindex = 0;
short whichbar();
long xorg,yorg,xsize,ysize;
long redbar,greenbar,bluebar;
long startbar,endbar;

main()
{
  short index, val;
  Device xpos, ypos;
  initialize();
  while (TRUE) {
    switch (qread(&val)) {
    case ESCKEY:
      greset();
      gexit();
      exit(0);
    case REDRAW:
      reshapeviewport();
      getwindowsize();
      buildmap();
      displaymap();
      break;
    case LEFTMOUSE:
      if (val){
        qread(&xpos);
        qread(&ypos);
        qread(&val);
        qread(&val);
        qread(&val);
        if (insideport(xpos,ypos)) {
          index = -1;
          switch (whichbar(xpos,ypos,&index)) {
          case 0:
            redindex = index;
            break;
          case 1:
            greenindex = index;
            break;
          case 2:
            blueindex = index;
            break;
          default:
            break;
          }

          if (index != -1) {
            buildmap();
            displaymap();
```

```
          }
        }
      }
      break;
    }
  }
}
initialize()
{
  int gid;

  prefposition(10, XMAXSCREEN-10, 10, YMAXSCREEN-20);
  keepaspect(5,4);
  gid = winopen("colored");
  ortho2(-0.5, (float)XMAXSCREEN-0.5, -0.5, (float)YMAXSCREEN-0.5);
  color(0);
  clear();
  mapcolor(CURRENTCOLOR, 0, 0, 0);
  qdevice(LEFTMOUSE);
  tie(LEFTMOUSE, MOUSEX, MOUSEY);
  qdevice(ESCKEY);
  qdevice(REDRAW);
  qenter(REDRAW,gid);
}
getwindowsize()
{
  getorigin(&xorg,&yorg);
  getsize(&xsize,&ysize);
  redbar = ((REDBAR * ysize) / YMAXSCREEN) + yorg;
  greenbar = ((GREENBAR * ysize) / YMAXSCREEN) + yorg;
  bluebar = ((BLUEBAR * ysize) / YMAXSCREEN) + yorg;
  startbar = ((STARTBAR * xsize) / XMAXSCREEN) + xorg;
  endbar = ((ENDBAR * xsize) / XMAXSCREEN) + xorg;
}
buildmap()
{
  register i, j;
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 64; j++) {
      switch (i) {
      case 0:                /* red */
        mapcolor(START+i*64+j, indextovalue(j),
            indextovalue(greenindex),
            indextovalue(blueindex));
        break;
      case 1:                /* green */
        mapcolor(START+i*64+j, indextovalue(redindex),
            indextovalue(j),
            indextovalue(blueindex));
        break;
      case 2:                /* blue */
        mapcolor(START+i*64+j, indextovalue(redindex),
            indextovalue(greenindex),
            indextovalue(j));
        break;
      }
    }
  }
  mapcolor(CURRENTCOLOR, indextovalue(redindex),
      indextovalue(greenindex),
      indextovalue(blueindex));
}
displaymap()
{
  register i, j;
```

```
  char redstr[10], greenstr[10], bluestr[10];
  color(BLACK);
  clear();
  for (i = 0; i < 3; i++)
    for (j = 0; j < 64; j++) {
      color(START+i*64 + j);
      rectfi(250 + 13*j, 934 - i*133, 263 + 13*j,
          867 - i*133);
      color(WHITE);
      recti(250 + 13*j, 934 - i*133, 263 + 13*j,
          867 - i*133);
    }
  color(CURRENTCOLOR);
  rectfi(500, 267, 750, 400);
  color(WHITE);
  recti(500, 267, 750, 400);
  cmov2i(186, 894);
  charstr("RED");
  cmov2i(186, 760);
  charstr("GREEN");
  cmov2i(186, 627);
  charstr("BLUE");
  cmov2i(343, 327);
  charstr("CURRENT COLOR");
  cmov2i(475, 133);
  charstr("Left mouse button: choose a color");
  cmov2i(475, 112);
  charstr("Escape key : exit");
  move2i(startbar + 13*redindex, 934);
  draw2i(startbar + 13*redindex, 960);
  cmov2i(startbar + 6 + 13*redindex, 940);
  sprintf(redstr, "%d", indextovalue(redindex));
  charstr(redstr);
  move2i(startbar + 13*greenindex, 800);
  draw2i(startbar + 13*greenindex, 827);
  cmov2i(startbar + 6 + 13*greenindex, 806);
  sprintf(greenstr, "%d", indextovalue(greenindex));
  charstr(greenstr);
  move2i(startbar + 13*blueindex, 666);
  draw2i(startbar + 13*blueindex, 694);
  cmov2i(startbar + 6 + 13*blueindex, 674);
  sprintf(bluestr, "%d", indextovalue(blueindex));
  charstr(bluestr);
  cmov2i(563, 414);
  charstr("(");
  charstr(redstr);
  charstr(", ");
  charstr(greenstr);
  charstr(", ");
  charstr(bluestr);
  charstr(")");
}
/* return 1 if the position of the cursor is within the window */
insideport(x,y)
int x, y;
{
  if(x<xorg)
    return 0;
  if(x>(xorg+xsize))
    return 0;
  if(y<yorg)
    return 0;
  if(y>(yorg+ysize))
    return 0;
  return 1;
}
```

```
/*
returns 0 if in the redbar, 1 if in the greenbar and 2 if in the
blue bar
*/
short whichbar(xpos,ypos,index)
long xpos,ypos;
short *index;
{
  short i;
  i = -1;
  if (redbar - BARWIDTH <= ypos && ypos <= redbar)  /* red color bar */
    i = 0;
  else if (greenbar-BARWIDTH <= ypos &&
      ypos <= greenbar)                   /* green color bar */
    i = 1;
  else if (bluebar - BARWIDTH <= ypos &&
      ypos <= bluebar) /* blue color bar */
    i = 2;
  if (i != -1) {
    if (startbar <= xpos && xpos < endbar) {
      *index = (xpos - startbar)/13;
      return(i);
    }
  }
  return(-1);
}
/*
Changes:
  - Added the define of start.  Before this addition this program
    did not work on a gl adapter with an 8-bit color table
    #define START 512
    change to:
    #define START 64
  - Changed the CURRENTCOLOR to 63
    Keep in mind that this program will alter the colormap and it
    does not put it back the way it was before
  - The comments at the beginning of the documentation are incorrect.
    The way that the program was before this program only worked on a
    machine with at least a 10-bit color look up table.
*/
```

Related Information

The **keepaspect** subroutine, **mapcolor** subroutine, **prefposition** subroutine, **setlinestyle** subroutine, **wincontraints** subroutine, **winopen** subroutine .

---

curve1.c Example C Language Program

```
/* Example C Language Program curve1.c */

/*
This program draws 3 curve segments. The "horizontal" one is drawn
with a Bezier basis matrix, the "vertical" one with a Cardinal
basis matrix, and the "diagonal" one with a B-spline basis matrix.
All use the same set of 4 control points, contained in the array
geom1.

Before crv (rcrv) is called, a basis and precision matrix must be
defined.
*/

#include <gl/gl.h>
#include <gl/device.h>

#define BEZIER 1
#define CARDINAL 2
#define BSPLINE 3
```

```
Matrix beziermatrix = {
  { -1, 3 ,-3, 1 },
  {  3,-6 , 3, 0 },
  { -3, 3 , 0, 0 },
  {  1, 0 , 0, 0 }
};
Matrix cardinalmatrix = {
  { -0.5,  1.5, -1.5,  0.5 },
  {  1.0, -2.5,  2.0, -0.5 },
  { -0.5,  0.0,  0.5,  0.0 },
  {  0.0,  1.0,  0.0,  0.0 }
};
Matrix bsplinematrix = {
  { -1.0/6.0,  3.0/6.0, -3.0/6.0, 1.0/6.0 },
  {  3.0/6.0, -6.0/6.0,  3.0/6.0, 0.0     },
  { -3.0/6.0,  0.0    ,  3.0/6.0, 0.0     },
  {  1.0/6.0,  4.0/6.0,  1.0/6.0, 0.0     }
};
Coord geom1[4][3] = {
  { 100.0, 200.0, 0.0},
  { 200.0, 300.0, 0.0},
  { 200.0, 100.0, 0.0},
  { 300.0, 200.0, 0.0}
};
main()
{
  int dev,val;
  initialize();
  while (TRUE) {
    if (qtest()) {
      dev = qread(&val);
      if (dev == ESCKEY) {
        gexit();
        exit(1);
      }
      else if (dev == REDRAW) {
        reshapeviewport();
        drawcurve();
      }
    }
  }
}
initialize()
{
  int gid;
  prefposition(200, 600, 100, 500);
  gid = winopen("curve1");
  qdevice(ESCKEY);
  qdevice(REDRAW);
  qenter(REDRAW,gid);
}
drawcurve()
{
  int i,xx,yy,x,y;
  color(BLACK);
  clear();
  defbasis(BEZIER,beziermatrix);
    /* define a basis matrix called BEZIER */
  curvebasis(BEZIER);
    /* identify the BEZIER matrix as the current basis matrix */
  curveprecision(20);
    /* set the current precisionto 20 (the curve segment will be
```

```
    drawn using 20 line segments) */
  color(RED);
  crv(geom1);
    /* a new curve segment is drawn */
  defbasis(CARDINAL,cardinalmatrix);
    /* a new basis is defined */
  curvebasis(CARDINAL);
    /* the current basis is reset.  note that the curveprecision
    does not have to be restated unless it is to be changed */
  color(BLUE);
  crv(geom1);
    /* a new curve segment is drawn */
  defbasis(BSPLINE,bsplinematrix);
    /* a new basis is defined */
  curvebasis(BSPLINE);
    /* the current basis is reset */
  color(GREEN);
  crv(geom1);
    /* a new curve segment is drawn */
    /* show the control points */
   color(WHITE);
  for ( i = 0 ; i < 4 ; i++ )  {
    for ( xx = -2 ; xx < 2 ; xx++) {
      for ( yy = -2 ; yy < 2 ; yy++) {
        pnt2( geom1[i][0] + (Coord) xx , geom1[i][1] + (Coord) yy);
      }
    }
  }
}
/*Changes:
  - showing the control points (added variables i,xx,yy)
  - the size of the window with
      from prefposition (200,500,100,400); to
      prefposition(200, 600, 100, 500);
   - remove the translate(150.0, 150.0, 0.0); because when the
      window redraws it successively moves the curves away
   - change the colors of the curves from all red to each
      of the primary colors of light
   - added 100.0 to each of the y components of the curve's geomerty
*/
```

Related Information

The **crv** subroutine, **curvebasis** subroutine, **curveprecision** subroutine, **defbasis** subroutine .

Drawing Curves in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

curve2.c Example C Language Program

```
/*
curve2.c

This program demonstrates the use of joined curve segments. It
draws three curves. One with a Bezier basis, one with a Cardinal
spline basis, and one with a B-spline basis.

The array geom2 contains 6 control points.  With the Bezier basis
matrix, 3 sets of overlapping control points result in 3 separate
curve segments.  With the Cardinal spline and B-spline matrices,
the same overlapping sets of control points result in 3 joined
curve segments.
*/

#include <gl/gl.h>
#include <gl/device.h>
```

```
Matrix beziermatrix = {
  { -1,  3, -3,  1 } ,
  {  3, -6,  3,  0 },
  { -3,  3,  0,  0 },
  {  1,  0,  0,  0 }};
Matrix cardinalmatrix = {
  { -0.5,  1.5, -1.5,  0.5 },
  {  1.0, -2.5,  2.0, -0.5 },
  { -0.5,  0.0,  0.5,  0.0 },
  {  0.0,  1.0,  0.0,  0.0 }};
Matrix bsplinematrix = {
  { -1.0/6.0,  3.0/6.0, -3.0/6.0, 1.0/6.0 },
  {  3.0/6.0, -6.0/6.0,  3.0/6.0,    0.0 },
  { -3.0/6.0,      0.0,  3.0/6.0,    0.0 },
  {  1.0/6.0,  4.0/6.0,  1.0/6.0,    0.0 }};
#define BEZIER 1
#define CARDINAL 2
#define BSPLINE 3
Coord geom2[6][3] = {
  { 150.0, 400.0, 0.0},
  { 350.0, 100.0, 0.0},
  { 200.0, 350.0, 0.0},
  {  50.0,   0.0, 0.0},
  {   0.0, 200.0, 0.0},
  { 100.0, 300.0, 0.0}};
main()
{
  int dev,val;
  initialize();
  while (TRUE) {
    if (qtest()) {
      dev = qread(&val);
      if (dev == ESCKEY) {
        gexit();
        exit();
      }
      else if (dev == REDRAW) {
        reshapeviewport();
        drawcurve();
      }
    }
  }
}
initialize()
{
  int gid;
  prefposition(200, 650, 200, 800);
  gid = winopen("curve2");
  winconstraints();                  /* Allows resizing */
  qdevice(ESCKEY);
  qdevice(REDRAW);
  qenter(REDRAW,gid);
}
drawcurve()
{
  short i,xx,yy;
  color(BLACK);
  clear();
  pushmatrix();                      /* Copy the top matrix */
  translate (50.0,100.0,0.0);        /* Center the splines */
  defbasis(BEZIER,beziermatrix);     /* define a basis matrix
```

```
                                    called BEZIER */
  defbasis(CARDINAL,cardinalmatrix); /* a new basis is
                                    defined */
  defbasis(BSPLINE,bsplinematrix);   /* a new basis is
                                    defined */
  curvebasis(BEZIER);      /* the Bezier matrix becomes the
                              current basis */
  curveprecision(20);      /* the precision is set to 20 */
  color(RED);
  crvn(6, geom2);          /* the curvs command called with a
                              Bezier basis causes 3 separate
                              curve segments to be drawn */
  curvebasis(CARDINAL);    /* the Cardinal basis becomes the
                              current basis */
  color(GREEN);
  crvn(6, geom2);          /* the crvs command called with a
                              Cardinal spline basis causes a
                              smooth curve to be drawn */
  curvebasis(BSPLINE);     /* the B-spline basis becomes the
                              current basis */
  color(BLUE);
  crvn(6, geom2);          /* the curvs command called with a
                              B-spline basis causes the
                              smoothest curve to be drawn */

    /* show the control points */
  color(WHITE);
  for ( i = 0 ; i < 6 ; i++ )  {
    for ( xx = -2 ; xx < 2 ; xx++) {
      for ( yy = -2 ; yy < 2 ; yy++) {
        pnt2( geom2[i][0] + (Coord) xx , geom2[i][1] + (Coord) yy);
      }
    }
  }
  popmatrix();
}
/*
  Changes:
      - Now the control points are displayed
    - The window can be resized
*/
```

Related Information

The **crvn** subroutine .

Drawing Curves in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

curve3.c Example C Language Program

```
/*
Example: curve3.c

This program draw a Bezier curve segment using curveit().  The
Cardinal spline and B-spline curve segments could be drawn in a
similar manner (only the basis matrix would be different).
*/

#include <gl/gl.h>
#include <gl/device.h>

#define BEZIER 1

Matrix beziermatrix = {
  { -1.0,  3.0, -3.0,  1.0 } ,
  {  3.0, -6.0,  3.0,  0.0 },
  { -3.0,  3.0,  0.0,  0.0 },
  {  1.0,  0.0,  0.0,  0.0 }};
```

```
Matrix geom1 = {
  { 50.0, 150.0, 0.0, 1.0},
  { 150.0, 225.0, 0.0, 1.0},
  { 150.0, 075.0, 0.0, 1.0},
  { 250.0, 150.0, 0.0, 1.0}};
Matrix precisionmatrix = {
  { 6.0/8000.0,       0.0,    0.0, 0.0},
  { 6.0/8000.0, 2.0/400.0,    0.0, 0.0},
  { 1.0/8000.0, 1.0/400.0, 1/20.0, 0.0},
  {        0.0,       0.0,    0.0, 1.0}};
main()
{
  int dev,val;
  initialize();
  while (TRUE) {
    if (qtest()) {
      dev = qread(&val);
      if (dev == ESCKEY) {
        gexit();
        exit();
      }
      else if (dev == REDRAW) {
        reshapeviewport();
        drawcurve();
      }
    }
  }
}
initialize()
{
  int gid;
  prefposition(200, 500, 100, 400);
  gid = winopen("curve3");
   winconstraints();
  qdevice(ESCKEY);
  qdevice(REDRAW);
  qenter(REDRAW,gid);
}
drawcurve()
{
  int i,xx,yy;
  color(BLACK);
  clear();
  pushmatrix();                /* the current transformation
                                * matrix on the matrix stack is
                                * saved */
  multmatrix(geom1);           /* the product of the current
                                * transformation matrix and the
                                * matrix containing the
                                * control points becomes the new
                                * current transformation matrix */
  multmatrix(beziermatrix);    /* the product of the basis
                                * matrix and the current
                                * transformation matrix becomes
                                * the new current transformation
                                * matrix */
  multmatrix(precisionmatrix); /* the product of the precision
                                * matrix and the current
                                * transformation matrix
                                * becomes the new current
                                * transformation matrix */
```

```
  move(0.0,0.0,0.0);           /* this command must be issued
                                * so that the correct first
                                * point is generated by the
                                * curveit command */
  color(RED);
  curveit(20);                 /* a curve consisting of 20 line
                                * segments is drawn */
  popmatrix();                 /* the original transformation
                                * matrix is restored */
    /* show the control points */
  color(WHITE);
  for ( i = 0 ; i < 4 ; i++ )  {
    for ( xx = -2 ; xx < 2 ; xx++) {
      for ( yy = -2 ; yy < 2 ; yy++) {
        pnt2( geom1[i][0] + (Coord) xx , geom1[i][1] + (Coord) yy);
      }
    }
  }
}
/*
  Changes:
    - the control points are drawn on the screen
    - added winconstraints() so the window could be resized
    - changes the geometry so it would be centered
      from:
         Matrix geom1 = {
            { 100.0, 100.0, 0.0, 1.0},
            { 200.0, 200.0, 0.0, 1.0},
            { 200.0,   0.0, 0.0, 1.0},
            { 300.0, 100.0, 0.0, 1.0}};
      to:
         Matrix geom1 = {
            { 100.0, 150.0, 0.0, 1.0},
            { 200.0, 225.0, 0.0, 1.0},
            { 200.0, 075.0, 0.0, 1.0},
            { 300.0, 150.0, 0.0, 1.0}};
*/
```

Related Information

The **curveit** subroutine, the **multmatrix** subroutine .

Drawing Curves in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

curved.c Example C Language Program

```
/*
curved.c -
A minimal curve editor.*
           Paul Haeerli - 1985
*/
#include <gl/gl.h>
#include <gl/device.h>
float endgeom[4][3];
float geom[100][3];
int pt[100];
#define ADDPOINT        1
#define MOVEPOINT       2
#define INSERTPOINT     3
#define DELETEPOINT     4
#define CHANGEPOINT     5
#define BACKGROUND      7
```

```
#define LINE            0
#define ROUND           1
#define SQUARE          2
#define MOUSEXMAP(x)    ( (100.0*((x)-xorg))/xsize )
#define MOUSEYMAP(y)    ( (100.0*((y)-yorg))/ysize )

short raster[] = {
  0xf800, 0x8800, 0x8800, 0x8800, 0xf800,
  0x7000, 0x8800, 0x8800, 0x8800, 0x7000, };

Fontchar chars[] = {
  {0,0,0, 0, 0,0},
  {0,5,5,-2,-2,5},
  {5,5,5,-2,-2,5}
};

Matrix b_spline = {
  {-1.0/6.0, 1.0/2.0,  -1.0/2.0, 1.0/6.0},
  { 1.0/2.0,    -1.0,   1.0/2.0,     0.0},
  {-1.0/2.0,     0.0,   1.0/2.0,     0.0},
  { 1.0/6.0, 2.0/3.0,   1.0/6.0,     0.0}
};

Matrix lob_spline = {
  {     0.0,       0.0,       0.0,  0.0},
  { 1.0/3.0,  -2.0/3.0,   1.0/3.0,  0.0},
  {-7.0/6.0,   4.0/3.0,  -1.0/6.0,  0.0},
  {     1.0,       0.0,       0.0,  0.0}, };

Matrix hib_spline = {
  {0.0,       0.0,       0.0,      0.0},
  {0.0,   1.0/3.0,  -2.0/3.0,  1.0/3.0},
  {0.0,  -1.0/2.0,       0.0,  1.0/2.0},
  {0.0,   1.0/6.0,   2.0/3.0,  1.0/6.0}, };

Matrix c_spline =  {
  {-0.5,   1.5,  -1.5,   0.5},
  { 1.0,  -2.5,   2.0,  -0.5},
  {-0.5,   0.0,   0.5,   0.0},
  { 0.0,   1.0,   0.0,   0.0}
};

Matrix loc_spline = {
  { 0.0,   0.0,   0.0,  0.0},
  { 0.5,  -1.0,   0.5,  0.0},
  {-1.5,   2.0,  -0.5,  0.0},
  { 1.0,   0.0,   0.0,  0.0}
};

Matrix hic_spline = {
  {0.0,  0.0,  0.0,  0.0},
  {0.0,  0.5, -1.0,  0.5},
  {0.0, -0.5,  0.0,  0.5},
  {0.0,  0.0,  1.0,  0.0}
};

#define BSPLINE          100
#define LOBSPLINE        101
#define HIBSPLINE        102

int xsize, ysize;
int xorg, yorg;
float mx, my;
int curmode = ADDPOINT;
int points;
int menu;
main(argc,argv)
int argc;
char **argv;
{
  prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
      YMAXSCREEN*3/4);
  winopen("curved");
```

```
  menu = defpup("curved %t|add|move|insert|delete|change");
  defrasterfont(1,7,3,chars,10,raster);
  font(1);
  deflinestyle(1,0xf0f0);
  if (argc == 1) {
    defbasis(BSPLINE,b_spline);
    defbasis(LOBSPLINE,lob_spline);
    defbasis(HIBSPLINE,hib_spline);
  }
  else {
    defbasis(BSPLINE,c_spline);
    defbasis(LOBSPLINE,loc_spline);
    defbasis(HIBSPLINE,hic_spline);
  }
  curveprecision(6);
  makeframe();
  initdevices();
  points = 0;
  while (1)
    getinput();
}
initdevices()
{
  qdevice(MOUSEX);
  qdevice(MOUSEY);
  qdevice(LEFTMOUSE);
  qdevice(MENUBUTTON);
  qdevice(KEYBD);
}
getinput()
{
  Device dev;
  short val;
  int sel;
  do {
    if (!qtest())
      mouseevent(2,mx,my);
    dev = qread(&val);
    switch (dev) {
    case MENUBUTTON:
      if (val) {
        sel = dopup(menu);
        if (sel>0)
          curmode = sel;
      }
      font(1);
      break;
    case LEFTMOUSE:
      mouseevent(val,mx,my);
      break;
    case MOUSEX:
      mx = MOUSEXMAP(val);
      break;
    case MOUSEY:
      my = MOUSEYMAP(val);
      break;
    case KEYBD:
      switch (val) {
      case 'a':
        break;
      case 'i':
        curmode = INSERTPOINT;
        break;
      case 'm':
        curmode = MOVEPOINT;
        break;
```

```
      case 'd':
        curmode = DELETEPOINT;
        break;
      case 'c':
        curmode = CHANGEPOINT;
        break;
      }
      break;
    case REDRAW:
      reshapeviewport();
      makeframe();
      break;
    }
  } while (qtest());
}
int curpoint;
int moving;
mouseevent(state,x,y)
int state;
float x, y;
{
  int nextpoint;
  switch (curmode) {
  case ADDPOINT:
    if (state == 1) {
      curpoint = duppoint(points);
      geom[curpoint][0] = x;
      geom[curpoint][1] = y;
      pt[curpoint] = SQUARE;
      drawline(LINE);
    }
    break;
  case MOVEPOINT:
    if (state == 1) {
      curpoint = findpoint(x,y);
      moving = 1;
    }
    else if (state == 2) {
      if (moving) {
        drawline(BACKGROUND);
        geom[curpoint][0] = x;
        geom[curpoint][1] = y;
        drawline(LINE);
      }
    }
    else if (state == 0)
      moving = 0;
    break;
  case INSERTPOINT:
    if (state == 1) {
      curpoint = findpoint(x,y);
      if (curpoint < 0)
        curpoint = duppoint(points);
      else
        curpoint = duppoint(curpoint);
      drawline(BACKGROUND);
      geom[curpoint][0] = x;
      geom[curpoint][1] = y;
      pt[curpoint] = SQUARE;
      drawline(LINE);
    }
    break;
  case DELETEPOINT:
    if (state == 1) {
      curpoint = findpoint(x,y);
```

```
      if (curpoint >= 0) {
        drawline(BACKGROUND);
        delpoint(curpoint);
        drawline(LINE);
      }
    }
    break;
  case CHANGEPOINT:
    if (state == 1) {
      curpoint = findpoint(x,y);
      if (curpoint >= 0) {
        drawline(BACKGROUND);
        if (pt[curpoint] == ROUND)
          pt[curpoint] = SQUARE;
        else
          pt[curpoint] = ROUND;
        drawline(LINE);
      }
    }
    break;
  }
}
makeframe()
{
  getorigin(&xorg,&yorg);
  getsize(&xsize,&ysize);
  ortho2(0.0,100.0,0.0,100.0);
  color(BACKGROUND);
  clear();
  drawline(LINE);
}
float ppdist(x1,y1,x2,y2)
float x1,y1,x2,y2;
{
  register float dx, dy;
  dx = x2-x1;
  if (dx<0) dx = -dx;
  dy = y2-y1;
  if (dy<0) dy = -dy;
  return dx+dy;
}
float pldist(x,y,x1,y1,x2,y2)
float x, y, x1, y1, x2, y2;
{
  register float dx, dy, c;
  dx = x2-x1;
  dy = y2-y1;
  c = dy*x1-dx*y1;
}
findpoint(x,y)
float x, y;
{
  register float mindist;
  register int minpnt;
  register int i;
  float dist;
  mindist = 100000.0;
  minpnt = -1;
  for (i=0; i<points; i++) {
    dist = ppdist(geom[i][0],geom[i][1],x,y);
```

```
    if (dist<mindist) {
      mindist = dist;
      minpnt = i;
    }
  }
  return minpnt;
}
findedge(x,y)
float x, y;
{
  register float mindist;
  register int minpnt;
  register int i;
  float dist;
  mindist = 100000.0;
  minpnt = -1;
  for (i=0; i<points; i++) {
    dist = ppdist(geom[i][0],geom[i][1],x,y);
    if (dist<mindist) {
      mindist = dist;
      minpnt = i;
    }
  }
  return minpnt;
}
duppoint(pnt)
int pnt;
{
  register int i;
  for (i=points; i>pnt; i--) {
    geom[i][0] = geom[i-1][0];
    geom[i][1] = geom[i-1][1];
    pt[i] = pt[i-1];
  }
  points++;
  return pnt;
}
delpoint(pnt)
int pnt;
{
  register int i;
  for (i=pnt; i<points; i++) {
    geom[i][0] = geom[i+1][0];
    geom[i][1] = geom[i+1][1];
    pt[i] = pt[i+1];
  }
  points--;
}
drawline(c)
int c;
{
  register int i, j;
  pt[0] = SQUARE;
  pt[points-1] = SQUARE;
  color(c);
  if (c == BACKGROUND)
    clear();
  else {
    setlinestyle(1);
    move2(geom[0][0],geom[0][1]);
```

```
    for (i=0; i<points; i++) {
      draw2(geom[i][0],geom[i][1]);
      putsym(i);
    }
    setlinestyle(0);
    move2(geom[0][0],geom[0][1]);
    for (i=1; i<points; i++) {
      if(pt[i] == SQUARE) {
        if(pt[i-1] == ROUND) {
          for (j=0; j<4; j++) {
            endgeom[j][0] = geom[i-2+j][0];
            endgeom[j][1] = geom[i-2+j][1];
          }
          endgeom[3][0] = endgeom[2][0];
          endgeom[3][1] = endgeom[2][1];
          curvebasis(BSPLINE);
          crv(endgeom);
          draw2(geom[i][0],geom[i][1]);
        }
        else {
          draw2(geom[i][0],geom[i][1]);
        }
      }
      else {
        if (pt[i-1] == SQUARE) {
          for (j=0; j<4; j++) {
            endgeom[j][0] = geom[i-2+j][0];
            endgeom[j][1] = geom[i-2+j][1];
          }
          endgeom[0][0] = endgeom[1][0];
          endgeom[0][1] = endgeom[1][1];
          curvebasis(BSPLINE);
          crv(endgeom);
        }
        else {
          curvebasis(BSPLINE);
          crv(&geom[i-2][0]);
        }
      }
    }
  }
}
putsym(i)
register int i;
{
  char buf[2];
  cmov2(geom[i][0],geom[i][1]);
  if (pt[i] == SQUARE)
    buf[0] = 1;
  else
    buf[0] = 2;
  buf[1] = 0;
  charstr(buf);
}
```

Related Information

The **charstr** subroutine, **cmov** subroutine, **deflinestyle** subroutine, **defpup** subroutine, **defrasterfont** subroutine, **dopup** subroutine, **font** subroutine .

cylinder1.c Example C Language Program

```
/* Example C Languange Program cylinder1.c*/
```

```
/*
The following program illustrates how to use the Graphics
Library to perform lighting.  It draws a cylinder and rotates
it.

This program requires a z buffer.
*/

#include <gl/gl.h>
#include <math.h>
#include <stdio.h>

#define RADIUS 0.9
#define TWOPI 6.28318530
#define PI 3.14159265

/* define black RGB color */
float  blackvec[3] = {
  0.0, 0.0, 0.0};

Matrix idmat = {
  1.0,0.0,0.0,0.0, /* identity matrix */
  0.0,1.0,0.0,0.0,
  0.0,0.0,1.0,0.0,
  0.0,0.0,0.0,1.0};

/*define a polygon with some structures
 * -- for code readability*/
typedef struct { /* structure for a 3-D vertex */
  Coord x;
  Coord y;
  Coord z;
} POINT;

typedef struct { /* 4 vertex lighted polygon struct */
  POINT vertex[4];
  POINT normal[4];
} POLYGON;

int number_of_polys;        /* cylinder polygon count */
POLYGON *polygon;           /* pointer to polygon list */

/*
** def_simple_light_calc()
** Tell the Graphics Library to DEFINE a simple
** lighting calculation that accounts for diffuse
** and ambient reflection.  This simple
** lighting calculation happens to use the default
** lighting parameters in the Graphics Library.
*/
def_simple_light_calc()
{
  lmdef(DEFMATERIAL, 1, 0, NULL);
  lmdef(DEFLIGHT, 1, 0, NULL);
  lmdef(DEFLMODEL, 1, 0, NULL);
}

/*
** use_simple_light_calc()
** Tell the Graphics Library to USE the
** simple lighting calculation that we
** defined earlier.
*/

use_simple_light_calc()
{
  lmbind(MATERIAL, 1);
  lmbind(LIGHT0, 1);
  lmbind(LMODEL, 1);
}

/*
** make_cylinder()
** Draw a cylinder using (2 * n) polygons
```

```
** to approximate the curvature and n
** polygons to describe the length.
** This requires (2 * n^2) polygons to
** describe the cylinder. Compute the
** surface normal at each vertex so we
** can use the Graphics Library to perform
** lighting calculations.
*/
make_cylinder(n)
int n;
{
  POLYGON *p;                 /* pointer into polygon list */
  float theta, dtheta,        /* current angle and angle */
        x, dx;                /* increment around section */
                              /* current position and */
                              /* increment along cylinder side */
  int vertex_i;               /* vertex counter */
                              /* allocate and point to enough */
                              /* memory for all the polygons */
  number_of_polys = 2 * n * n;
  p = polygon = (POLYGON *)
      malloc(number_of_polys * sizeof(POLYGON));
  dx = 3.0/n;                 /* n polygons for 3.0 units of length */
  dtheta = PI/n;              /* length of polygon along curvature */
                              /* for each layer of polygons along */
                              /* length of cylinder ... */
  for (x = -1.5; x < 1.5; x = x+dx) {
                              /* ... and for each polygon describing */
                              /* the circumference */
    for (theta = 0.0; theta < TWOPI; theta += dtheta) {
                              /* calculate the four points */
                              /* describing the polygon */
      p->vertex[0].x =
      p->vertex[1].x = x;
      p->vertex[0].y = p->vertex[3].y =
          RADIUS * cos(theta);
      p->vertex[0].z = p->vertex[3].z =
          RADIUS * sin(theta);
      p->vertex[1].y = p->vertex[2].y =
          RADIUS * cos(theta + dtheta);
      p->vertex[1].z = p->vertex[2].z =
          RADIUS * sin(theta + dtheta);
      p->vertex[2].x = p->vertex[3].x = x + dx;
                               /* calculate the four normals of unit length */
      for (vertex_i = 0; vertex_i < 4; vertex_i++) {
        p->normal[vertex_i].x = 0;
        p->normal[vertex_i].y =
            p->vertex[vertex_i].y / RADIUS;
        p->normal[vertex_i].z =
            p->vertex[vertex_i].z / RADIUS;
      }
      p++;
    }
  }
}
/*
** draw_cylinder()
** This subroutine increments through the 4
** vertices describing each polygon of
** the cylinder defined in make_cylinder.
** Note how a normal is sent down the
** graphics pipeline before each vertex
** so that the Graphics Library will
```

```
** compute the color for each vertex
** based on the lighting parameters that we
** are using.
*/
draw_cylinder()
{
  POLYGON *p;        /* pointer into polygon list */
  int poly_i;        /* polygon counter */
                     /* start at first polygon and */
                     /* increment through all of them */
  p = polygon;
  for (poly_i = 0; poly_i < number_of_polys; poly_i++) {
    bgnpolygon();        /* describe the polygon */
    n3f(&p->normal[0]);
    v3f(&p->vertex[0]);
    n3f(&p->normal[1]);
    v3f(&p->vertex[1]);
    n3f(&p->normal[2]);
    v3f(&p->vertex[2]);
    n3f(&p->normal[3]);
    v3f(&p->vertex[3]);
    endpolygon();
    p++;               /* go to the next polygon */
  }
}
/*
** main()
*/
main()
{
  int i;               /* set up graphics environment */
  prefposition(100, 600, 100, 600);
  winopen("cylinder");
  RGBmode();
  doublebuffer();
  gconfig();
  lsetdepth(0, 0x7FFFFF);
  zbuffer(TRUE);       /* Use mmode() to set up projection */
                       /* and viewing matrices for lighting */
  mmode(MVIEWING);
  perspective(400, 1.0, 4.0, 12.0);
  loadmatrix(idmat);
  lookat(0.0,0.0,8.0,0.0,0.0,0.0,0); /* let there be light !!!! */
  def_simple_light_calc();
  use_simple_light_calc();           /* Rotate cylinder in 2 deg. increments */
                                     /* about Y and Z axis for 180 frames */
  make_cylinder(25);
  for (i = 0; i < 180; i++) {
    c3f(blackvec);                   /* clear the frame */
    clear();
    zclear();
    pushmatrix();                    /* make a frame */
    rot(i * 2.0, 'Z');
    rot(i * 2.0, 'Y');
    draw_cylinder();
    popmatrix();
    swapbuffers();
  }
  sleep(3);
}
```

Related Information

The **c** subroutine, the **mmode** subroutine, the **rot** subroutine .

Lighting Basics in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

cylinder2.c Example C Language Program

```
/*
 *  cylinder2.c:
 *
 *  This program displays two intersecting cylinders, using a
 *  different surface material for each cylinder.  In addition, each
 *  cylinder is lit by two light source.
 *
 *  This program requires the z-buffer option.
 *
 */

#include <gl.h>
#include <math.h>
#include <stdio.h>

#define RADIUS 0.9
#define TWOPI 6.28318530
#define PI 3.14159265

/* define black RGB color */
float  blackvec[3] = {0.0, 0.0, 0.0};
Matrix idmat = {1.0,0.0,0.0,0.0,  /* identity matrix */
                0.0,1.0,0.0,0.0,
      0.0,0.0,1.0,0.0,
      0.0,0.0,0.0,1.0};

/* define a polygon with some structures */
typedef struct {   /* 3-D vertex structure */
    Coord x;
    Coord y;
    Coord z;
} POINT;

typedef struct {   /* lighted polygon struct */
    POINT vertex[4];
    POINT normal[4];
} POLYGON;

int number_of_polys;   /* cylinder polygon count */
POLYGON *polygon;   /* polygon list pointer */

/* define property arrays */
float shiny_material[] =
    {SPECULAR,  0.8, 0.8, 0.8, /* light gray reflectance */
     DIFFUSE,   0.4, 0.4, 0.4, /* gray reflectance */
     SHININESS, 30.0,          /* focused highlight */
     LMNULL};

float purple_material[] =
    {SPECULAR,  0.3, 0.3, 0.3, /* gray reflectance */
     DIFFUSE,   0.8, 0.0, 0.8, /* purple reflectance */
     SHININESS, 3.0,           /* unfocused highlight */
     AMBIENT,   0.2,0.0,0.2,   /* purple reflectance */
     LMNULL};

float blue_light[] =
    {LCOLOR,    0.0,0.0,0.6,   /* blue light */
     POSITION,  0.0,0.1,0.0,0.0,  /* Y axis at infinity */
     LMNULL};

/*
** def_light_calc()
** Tell the Graphics Library to DEFINE a
** lighting calculation that accounts for
```

```
** ambient, diffuse, and specular reflection.
** This lighting calculation defines a second
** material and light source.
*/
def_light_calc() {
    lmdef(DEFMATERIAL, 1, 11, shiny_material);
    lmdef(DEFMATERIAL, 2, 15, purple_material);
    lmdef(DEFLIGHT, 1, 0, NULL);
    lmdef(DEFLIGHT, 2, 10, blue_light);
    lmdef(DEFLMODEL, 1, 0, NULL);
}

/*
** use_light_calc()
** Tell the Graphics Library to USE
** the lighting calculation that we
** defined earlier.
*/
use_light_calc()
{
    lmbind(LIGHT0, 1);   /* use light source description 1 */
    lmbind(LIGHT1, 2);   /* use light source description 2 */
    lmbind(LMODEL, 1);   /* use lighting model 1 */
}

/*
** make_cylinder()
** Draw a cylinder using (2 * n) polygons
** to approximate the curvature and n polygons
** to describe the length. This requires (2 * n^2)
** polygons to describe the cylinder. Compute
** the surface normal at each vertex so we can
** use the hardware lighting facility to perform
** lighting calculations.
*/
make_cylinder(n)
int n;
{
    POLYGON *p;      /* polygon list pointer */
    float theta, dtheta,/* current angle and angle */
              /* increment around section */
     x, dx;   /* current position and */
         /* increment along cylinder side */
    int vertex_i;   /* vertex counter */

    /* allocate and point to enough */
    /* memory for all the polygons */
    number_of_polys = 2 * n * n;
    p = polygon = (POLYGON *)
   malloc(number_of_polys * sizeof(POLYGON));

    dx = 3.0/n;   /* n polygons for 3.0 units of length */
    dtheta = PI/n; /* length of polygon along curvature */

    /* for each layer of polygons */
    /* along length of cylinder ... */
    for (x = -1.5; x < 1.5; x = x+dx) {
   /* ... and for each polygon */
   /* describing the circumference */
        for (theta = 0.0; theta < TWOPI; theta += dtheta) {
       /* calculate the four points */
       /* describing the polygon */
       p->vertex[0].x = p->vertex[1].x = x;
       p->vertex[0].y = p->vertex[3].y =
      RADIUS * cos(theta);
       p->vertex[0].z = p->vertex[3].z =
      RADIUS * sin(theta);
       p->vertex[1].y = p->vertex[2].y =
```

```
      RADIUS * cos(theta + dtheta);
       p->vertex[1].z = p->vertex[2].z =
      RADIUS * sin(theta + dtheta);
       p->vertex[2].x = p->vertex[3].x = x + dx;

       /* calculate the four normals of unit length */
       for (vertex_i = 0; vertex_i < 4; vertex_i++) {
      p->normal[vertex_i].x = 0;
      p->normal[vertex_i].y =
          p->vertex[vertex_i].y / RADIUS;
      p->normal[vertex_i].z =
          p->vertex[vertex_i].z / RADIUS;
       }
       p++;
   }
    }
}

/*
** draw_cylinder()
** This subroutine increments through the 4
** vertices describing each polygon of the
** cylinder defined in make_cylinder.  Note
** how a normal is sent to the graphics
** hardware before each vertex so that the
** lighting facility will compute the color
** for each vertex based on the lighting
** parameters that we are using.
*/
draw_cylinder()
{
    POLYGON *p;    /* pointer into polygon list */
    int poly_i;    /* polygon counter */

    /* start at first polygon and */
    /* increment through all of them */
    p = polygon;
    for (poly_i = 0; poly_i < number_of_polys; poly_i++) {

   bgnpolygon();   /* describe the polygon */
            n3f(&p->normal[0]);
            v3f(&p->vertex[0]);
            n3f(&p->normal[1]);
            v3f(&p->vertex[1]);
            n3f(&p->normal[2]);
            v3f(&p->vertex[2]);
            n3f(&p->normal[3]);
            v3f(&p->vertex[3]);
   endpolygon();
   p++;      /* go to the next polygon */
    }
}

/*
** Main Program
*/
main()
{
    int i;

    /* set up graphics environment */
    prefposition(100, 600, 100, 600);
    winopen("cylinder");
    RGBmode();
    doublebuffer();
    gconfig();
    lsetdepth(0, 0x7FFFFF);
    zbuffer(TRUE);
```

```
    /* Use mmode() to set up projection */
    /* and viewing matrices for lighting */
    mmode(MVIEWING);
    perspective(400, 1.0, 4.0, 12.0);
    loadmatrix(idmat);
    lookat(0.0,0.0,8.0,0.0,0.0,0.0,0);
    /* let there be light !!!! */
    def_light_calc();
    use_light_calc();
    /* Rotate cylinders in 2 deg. increments */
    /* about Y and Z axis for 180 frames */
    make_cylinder(25);
    for (i = 0; i < 180; i++) {
        c3f(blackvec);
        clear();
   zclear();
   pushmatrix();
           rot(i * 2.0, 'Z');
       rot(i * 2.0, 'Y');
       /* use white shiny material for cyl 1*/
       lmbind(MATERIAL, 1);
            draw_cylinder();
       pushmatrix();
           rot(90.0, 'Y');
           scale(0.9,0.9,0.9);
      /* use purple rough material for cyl 2 */
           lmbind(MATERIAL, 2);
                draw_cylinder();
       popmatrix();
   popmatrix();
   swapbuffers();
    }
    sleep(3);
}
```

Related Information

The **bgnpolygon** or **endpolygon** subroutine, **lmbind** subroutine, **lmdef** subroutine, **n3f** subroutine, **scale** subroutine, **v** subroutine .

Lighting Basics in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

depthcue.c Example C Language Program

```
/* depthcue.c Example C Language Program
 *
 * Draws a depthcue'd 3-d wireframe cube with lots of
 * little points inside. Moving the mouse rotates the
 * cube.
 * NEAR and FAR (Z) clipplanes map to z values of
 * getgdesc(GD_ZMIN) and getgdesc(GD_ZMAX) respectively.
 * Points near NEAR are the brightest, those near FAR are
 * the dimmest. Give two hexadecimal values on the command
 * line to specify which zvalues are mapped to bright
 * and dark.
 *
 * Press the middle mouse to quit.
 */
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define CUBE 9531
float hrand();
```

```
/*Hold the colormap values*/
static short *CarrayR, *CarrayG, *CarrayB;

main (argc,argv)
int argc;
char **argv;
{
  int val, i;
  long near, far,           /*Z buffer and depth cueing*/
     nearest, farthest;     /*information*/
  float inc, c;
  short lo_end=128,         /*Low/High index of the colormap*/
     hi_end=255;            /*Used for the depth cueing*/

  prefposition(200, 600, 200, 600);
  winopen("depthcue");
  winconstraints();
  doublebuffer();
  gconfig();

  /*Find out the nearest and farthest values of the z mapping*/
  nearest   = getgdesc(GD_ZMIN);
  farthest  = getgdesc(GD_ZMAX);

  if (argc == 5) {
    near = strtol(argv[1], (char **) NULL, 0);
    far = strtol(argv[2], (char **) NULL, 0);
    lo_end = strtol(argv[3], (char **) NULL, 0);
    hi_end = strtol(argv[4], (char **) NULL, 0);

  }
  else if (argc == 3)  {
    near = strtol(argv[1], (char **) NULL, 0);
    far = strtol(argv[2], (char **) NULL, 0);
  }
  else {
    near = nearest;
    far  = farthest;
  }

  if (near < nearest)  near=nearest; /*Clamp to adapters constraints*/
  if (far  > farthest) far=farthest;

  reshapeviewport();
  perspective(600, 1.0, 350.0, 1400.0);
  lookat(0.0, 0.0, 700.0, 0.0, 0.0, 0.0, 0);
  qdevice(KEYBD);
  makeobj(CUBE);              /* generate a bunch of random points */
  for (i = 0; i < 100; i++)
    pnt(hrand(-200.0,200.0), hrand(-200.0,200.0),
        hrand(-200.0,200.0)); /* and a cube */
  movei(-200, -200, -200);
  drawi( 200, -200, -200);
  drawi( 200,  200, -200);
  drawi(-200,  200, -200);
  drawi(-200, -200, -200);
  drawi(-200, -200,  200);
  drawi(-200,  200,  200);
  drawi(-200,  200, -200);
  movei(-200,  200,  200);
  drawi( 200,  200,  200);
  drawi( 200, -200,  200);
  drawi(-200, -200,  200);
  movei( 200,  200,  200);
  drawi( 200,  200, -200);
  movei( 200, -200, -200);
  drawi( 200, -200,  200);
  closeobj();

  /*Save the current colormap so it can be reset when the program ends*/
  save_cmap(lo_end,hi_end);
```

```
  /* load the color map with a cyan ramp */
  /* starting at lo_end and going to hi_end inclusively */
  inc = 255.0/((float)(hi_end-lo_end));
  c = 0.0;
  for (i = lo_end; i <= hi_end; i++)
    {
      c += inc;
      mapcolor(i, 0,(Int16) c, (Int16) c);
    }
  /*Print out the information about the color and z buffer*/
  printf (" near    = %ld, far      = %ld \n", near, far);
  printf (" nearest = %ld, farthest = %ld \n", nearest, farthest);
  printf (" low colormap = %d, high colormap = %d\n", lo_end, hi_end);
  printf (" number of colors in ramp = %d \n", 1+hi_end-lo_end);
  /* Set the range of z values that will be stored in the  */
  /* bitplanes.  The percentage takes care of the error    */
  /* accumulation that occurs when the farthest limits are */
  /* used.                                                 */
  lsetdepth((long)(near*0.98), (long)(far*0.98));
  /* set up the mapping of z values to color map           */
  /* indexes: z value nearest is mapped to index lo_end    */
  /* and z value farthest is mapped to index hi_end        */
  lshaderange(lo_end,hi_end,nearest,farthest);
  /* turn on depthcue mode:  the color index of each       */
  /* pixel in points and lines is determined from the      */
  /* z value of the pixel                                  */
  depthcue(TRUE);
  /* until a key is pressed, rotate cube according to      */
  /* movement of the mouse                                 */
  while (!getbutton(MIDDLEMOUSE)) {
    pushmatrix();
    rotate(3*getvaluator(MOUSEY), 'x');
    rotate(3*getvaluator(MOUSEX), 'y');
    color(BLACK);
    clear();
    callobj(CUBE);
    popmatrix();
    swapbuffers();
  }
  /*Restore the colormap*/
  restore_cmap(lo_end,hi_end);
  /*Exit gracefully*/
  gexit();
  exit(1);
}
/* this routine returns random numbers in the            */
/* specified range                                       */
float hrand(low,high)
float low,high;
{
  float val;
  val = ((float)( (short)rand() & 0xffff)) / ((float)0xffff);
  return( (2.0 * val * (high-low)) + low);
}
/*This saves the colormap*/
save_cmap(lo_end,hi_end)
short lo_end,hi_end;
{
  CarrayR = calloc (lo_end+hi_end,sizeof(short));
  CarrayG = calloc (lo_end+hi_end,sizeof(short));
  CarrayB = calloc (lo_end+hi_end,sizeof(short));
```

```
  getmcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}

/*This restores the colormap*/
restore_cmap(lo_end,hi_end)
short lo_end,hi_end;
{
  mapcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}

/*Changes
    -  Put constants in the program for the colormap creation
    -  This program accept 2 arguments that are the nearest and
       farthest z values which are used for the depth cueing.
       This program called: lshaderange(128,255,-0x40000,0x3FFFFF);
       now:    lshaderange(lo_end,hi_end,nearest,farthest);
    -  Used a define for the object number (called CUBE)
    -  Save/restore the colormap
    -  The defaults of the near and far z mappings are no longer
       hard coded  ( near = -0x400000; far = 0x3fffff;), but are
       initialized with getgdesc.  Also the near and far mappings
       are clamped to there min and max values respectively.
*/
```

Related Information

---

doily.c Example C Language Program

```
/*
doily.c:

Draws a doily depending on how many points you give
it (range is
currently set between 3..100).  point count is equivale
nt to how
many line segments make up the circle's edge.
*/

#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>

#define PI 3.1415926535

float points[100][2];

main(argc, argv)
int argc;
char *argv[];
{
  int val,dev;
  long numpts;            /* First figure out how many points there are. */

  if (argc != 2) {
    printf("Usage: %s <point_count>\n", argv[0]);
    exit(0);
  }

  numpts = atoi(argv[1]); /* convert argument to internal format */
  if (numpts > 100) {
    printf("Too many points\n");
    exit(0);
  }
  else if (numpts < 3) {
    printf("Too few points\n"); exit(0);
  }
```

```
  initialize(numpts);
  while (TRUE) {
    if (qtest()) {
      dev = qread(&val);
      if (dev == ESCKEY) {
        gexit();
        exit(0);
      }
      else if (dev == REDRAW) {
        reshapeviewport();
        drawdoily(numpts);
      }
    }
  }
}
initialize(numpts)
long numpts;
{
  int gid;
  long i;    /* Now get the x and y coordinates of numpts equally-
              * spaced points around the unit circle.
              */
  for (i = 0; i < numpts; i++) {
    points[i][0] = cos((i*2.0*PI)/numpts);
    points[i][1] = sin((i*2.0*PI)/numpts);
  }
  keepaspect(1,1);
  prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
      YMAXSCREEN*3/4);
  gid = winopen("doily");
  qdevice(ESCKEY);
  qdevice(REDRAW);
  qenter(REDRAW,gid);
  ortho2(-1.2, 1.2, -1.2, 1.2);
}
drawdoily(numpts)
long numpts;
{
  long i,j;
  color(BLACK);
  clear();
  color(RED);
  for (i = 0; i < numpts; i++)
    for (j = i+1; j < numpts; j++) {
      move2(points[i][0], points[i][1]);
      draw2(points[j][0], points[j][1]);
    }
}
/*
  Changes:
  - exit(); to exit(1);
*/
```

Related Information

The **keepaspect** subroutine, **prefposition** subroutine .

doublebuff.c Example C Language Program

```
/* doublebuff.c:
 *
 * This program demonstrates double buffering.
 * Double buffering is a method for providing smooth,
 * flicker free animation of moving images. This example
 * draws two wireframe" cubes, one inside the other.
 * The cubes are connected to the mouse, and move as the
 * mouse is moved. Notice how the cubes move smoothly as
 * the mouse is moved.
 *
 * Holding down the shift key puts the system into
 * singlebuffer mode. Notice how the image flickers.
 * This flickering occurs because you are watching the
 * window being cleared and the cube being redrawn.
 * Letting go of the shift key puts the system back into
 * doublebuffer mode. The motion is smooth because all
 * drawing is occurring in the back buffer, which is
 * invisible. When the drawing is completed, the front
 * and back buffers are * exchanged.
 */

#include <gl/gl.h>
#include <gl/device.h>

#define DOUBLE_BUFFER 1
#define SINGLE_BUFFER 0

int buffer_mode=DOUBLE_BUFFER;         /* DOUBLE_BUFFER=doublebuffer, * 1=singlebuffer */
main()
{
  int x, y;                /*  current position of object */
  Device dev;
  short val;
  long event=0;
  x = 0;
  y = 0;
  initialize();
  while(TRUE) {
    if (qtest())           /*If there are any events in the queue*/
      dev = qread(&val);   /*Read the queue*/
      /*This allows the program always move the cube*/

    switch(dev) {
    case ESCKEY:           /*  exit program with ESC */
      gexit();
      exit(0);
      break;
    case REDRAW:           /* window was exposed. Redraw it. */
      reshapeviewport();
      drawcube(x,y);
      break;
    case LEFTMOUSE:        /* LEFTMOUSE turns on double buffer*/
      if (buffer_mode == SINGLE_BUFFER){
        buffer_mode = DOUBLE_BUFFER;
        doublebuffer();    /* put system in doublebuffer mode */
        gconfig();
      }
      break;
    case RIGHTMOUSE:       /* RIGHTMOUSE turns on single buffer*/
      if (buffer_mode == DOUBLE_BUFFER) {
        buffer_mode = SINGLE_BUFFER;
        singlebuffer();    /* put system in singlebuffer mode */
        gconfig();
      }
      break;
    }                      /*End of the switch statement*/
```

```
    y = getvaluator(MOUSEX); /*read the x,y value of the mouse*/
    x = getvaluator(MOUSEY); /*works outside of the window*/
    drawcube(x,y);
  }
}
initialize()
{
  /* open a window, 500 by 500 pixels */
  prefsize (500,500);
  winopen("doublebuffer"); /* put system into double buffer mode */
  winconstraints();
  doublebuffer();
  gconfig();               /* queue up the shift keys and the mouse */
  qdevice (ESCKEY);
  qdevice (LEFTMOUSE);
  qdevice (RIGHTMOUSE);

  /* use perspective projection */
  perspective (400, 3.0/2.0, 0.001, 100000.0);

  /* look at the origin from a distance of three units */
  translate (0.0, 0.0, -3.0);
  /* draw the first time, so that we don't stare
      * at a blank screen */
  drawcube();
}
drawcube(rotx,roty)
int rotx, roty;
{
  /* clear to background color */
  color(BLACK);
  clear();
  /* set the drawing color */
  color(WHITE);
  pushmatrix();
  /* rotate into the desired viewing position */
  rotate(rotx,'x');
  rotate(roty,'y');
  /* draw the big cube */
  cube();
  /* draw the small cube */
  scale(0.3,0.3,0.3);
  cube();
  popmatrix();

  /* put up a message, instructions */
  cmov2 (-0.4,-1.0);
  charstr ("Left: double buffer |  Right: single buffer");

  /* put up a message, showing buffering mode */
  cmov2 (-0.95, 0.95);
  if (buffer_mode == SINGLE_BUFFER) {
    charstr ("single-buffered, flickering image");
  }
  else {
    charstr ("double-buffered, smoothly animated image");
  }

  /*If it is double buffered, exchange the front and back buffers */
  /*Otherwise if it is single buffered the library ignores*/
  /*this call*/
  swapbuffers();

}
cube()
{
  /* make a cube out of 4 squares */
  pushmatrix();
  side();
  rotate(900,'x');
```

```
  side();
  rotate(900,'x');
  side();
  rotate(900,'x');
  side();
  popmatrix();
}

side()
{
  /* make a square translated 0.5 in the z direction */
  pushmatrix();
  translate(0.0,0.0,0.5);
  rect(-0.5,-0.5,0.5,0.5);
  popmatrix();
}

/*Changes:
  - The line once read buffer_mode = ; and now it reads:
        buffer_mode = 1; (Actually now it is SINGLE_BUFFER)
  - The qread and qdevice of MOUSEX and MOUSEY was removed.
    It is too choppy and it cannot read the mouse once the
    pointer's hot spot is outside of the window.  Now it is
    the getvaluator (MOUSE[X|Y]).
  - Put if (qtest) before the qread so the program does
    not stay in the qread command when the mouse's hot
    spot is outside of the window.
  - Added the defines for better programming style
      #define DOUBLE_BUFFER 1
      #define SINGLE_BUFFER 0
  - Made the left and right mouse buttons represent
      double and single buffer animation respectively
  - Added the button instructions in the window (above)
  - Added winconstraints so the window can be made larger
*/
```

Related Information

The **doublebuffer** subroutine .

Creating Animated Scenes in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)*.

---

draw.c Example C Language Program

```
/*
draw.c

An absolutely minimal line drawing program.
*/

#include <gl/gl.h>
#include <gl/device.h>

main()
{
   Device dev;
   short val;
   unsigned short xpos, ypos;

   initialize();
   while(TRUE) {
      dev = qread(&val);
      switch(dev) {               /* wait for mouse down */
      case ESCKEY:                /* quit */
         gexit();
         exit(0);
      case REDRAW:
         reshapeviewport();
```

```
         color(BLACK);
         clear();
         color(RED);
         break;
      case MIDDLEMOUSE:           /* move */
         qread(&xpos);
         qread(&ypos);
         move2i(xpos, ypos);
         qread(&val);              /* these three reads clear out */
         qread(&val);              /* the queue */
         qread(&val);
         break;
      case LEFTMOUSE:             /* draw */
         qread(&xpos);
         qread(&ypos);
         draw2i(xpos, ypos);
         qread(&val);
         qread(&val);
         qread(&val);
         break;
      }
   }
}
initialize()
{
   int gid;
   float xmax,ymax;
   prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4, YMAXSCREEN/4,
       YMAXSCREEN*3/4);
   gid = winopen("draw");
   xmax = .5 + (float) XMAXSCREEN;
   ymax = .5 + (float) YMAXSCREEN;
   ortho2(xmax/4.0,xmax*3.0/4.0,ymax/4.0,ymax*3.0/4.0);
   qdevice(ESCKEY);
   qdevice(LEFTMOUSE);
   qdevice(MIDDLEMOUSE);
   tie(LEFTMOUSE, MOUSEX, MOUSEY);
   tie(MIDDLEMOUSE, MOUSEX, MOUSEY);
   color(BLACK);
   clear();
   color(RED);
}
```

Related Information

The **getdrawmode** subroutine .

Understanding the Frame Buffer in GL in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* explains the use of different buffers.

---

font3.c Example C Language Program

```
/* Examples: font3.c*/
/* Define a font with three characters -- a lower-case j,
 * an arrow, and a Greek sigma. Use ASCII values 1 and 2
 * (\001 and \002) for the arrow and sigma. Use the
 * ASCII value of j (= \152) for the j character.
 */
#include <gl/gl.h>
#define EXAMPLEFONT        1
#define efont_ht        16
#define efont_nc        127
```

```
unsigned short efont_bits[] = {
   /* lower-case j */
   0x7000, 0xd800, 0x8c00, 0x0c00, 0x0c00, 0x0c00, 0x0c00,
   0x0c00, 0x0c00, 0x1c00, 0x0000, 0x0000, 0x0c00, 0x0c00,
  /* arrow */
   0x0200, 0x0300, 0x0380, 0xafc0, 0xafe0, 0xaff0, 0xafe0,
   0xafc0, 0x0380, 0x0300, 0x0200,
  /* sigma */
   0xffc0, 0xc0c0, 0x6000, 0x3000, 0x1800, 0x0c00, 0x0600,
   0x0c00, 0x1800, 0x3000, 0x6000, 0xc180, 0xff80,
  };
#define efont_nr (sizeof efont_bits)
#define ASSIGN(fontch, of, wi, he, xof, yof, wid) \
                fontch.offset = of; \
                fontch.w = wi; \
                fontch.h = he; \
                fontch.xoff = xof; \
                fontch.yoff = yof; \
                fontch.width = wid
Fontchar efont_chars[127];
main ()
{
   ASSIGN(efont_chars['j'],        0, 6, 14, 0, -2,  8);
   ASSIGN(efont_chars['\001'],14, 12, 11, 0,  0, 14);
   ASSIGN(efont_chars['\002'],25, 10, 13, 0,  0, 12);
   prefsize (200,200);
   winopen("font");
   color(BLACK);
   clear();
   defrasterfont (EXAMPLEFONT, efont_ht, efont_nc, efont_chars,
       efont_nr, efont_bits);
   font (EXAMPLEFONT);
   color(RED);
   cmov2i(100, 100);
   charstr ("j\001\002\001jj\002");
   cmov2i(100, 84);
   charstr("ajb\001c\002d");
   sleep(10);
}
```

Related Information

The **charstr** subroutine, **defrasterfont** subroutine, **font** subroutine .

Fonts in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* describes font assignment in GL, and discusses relevant subroutines.

---

gamma.c Example C Language Program for GL

```
#include <stdio.h>
#include <math.h>
#include <gl/gl.h>
#include <gl/device.h>
/* Example:  gammaramp.c */
/*
 * This example program shows the use of gamma ramps.  Gamma ramps
 * are useful for correcting the appeareance of photographs
 * (images) and antialiased lines (to correct for monitor
 * phosphors), and for implementing color-map tricks.
 *
```

```
 * Pressing PF7 or PF8 changes the gamma correction factor and
 * loads the new gamma ramp.  A window shows the current gamma
 * factor and a black-to-white scale.
 */
/* ------------------------------------------------------------ */
/*
 * This routine prints out the current value of gamma in a window.
 * At the bottom of the window is a color bar.
 */
void show_val (float gam)
{
   char gamstr[32];
   cpack (0x0);         /* black */
   clear();
   cpack (0xff);        /* red */
   cmov2 (10, 80);
   charstr ("Currently, the gamma correction factor is ");
   sprintf (gamstr, "%3.2f", gam);
   charstr (gamstr);
   cmov2 (10, 60);
   charstr ("To lower gamma, press PF7");
   cmov2 (10, 40);
   charstr ("To raise gamma, press PF8");   /* draw a sample color bar */
   cpack (0x0);         /* black */
   pmv2 (10, 10);
   pdr2 (10, 25);
   cpack (0xffffff);    /* white */
   pdr2 (390, 25);
   pdr2 (390, 10);
   pclos ();
}
/* ------------------------------------------------------------ */
/*
 * The following routine updates the adapter gamma ramps
 * based on the value of gamma passed to it.  Note that the
 * entire screen is affected (because there is only one set
 * of ramps for the entire screen).
 */
void update_ramps (float gam) {
   short rramp[256], gramp[256], bramp[256];
   int i;
   double corr, og;   /* compute the new gamma ramp */
   og = 1.0 / (gam + 0.00001);
   corr = 255.0 * pow ((double) (1.0 / 255.0), og);
   for (i=0; i<256; i++) {
      rramp[i] = gramp[i] = bramp[i] = (short) (corr * pow((float) (i), og));
   }
   /* load the new gamma ramp */
   gammaramp (rramp, gramp, bramp);
}
/* ------------------------------------------------------------ */
main() {
   float gamma = 1.0;
   float delta = 0.05;
   short dev, val;
   prefsize (400, 100);
   minsize (400, 100);
   maxsize (400, 100);         /* disable resizing of window */
   winopen ("gamma-ramp demo");
   RGBmode ();                 /* gamma ramps work in color map mode too! */
   gconfig ();
   update_ramps (gamma);       /* initialize the gamma ramps */
   show_val (gamma);           /* print current value of gamma */
   qdevice (RIGHTMOUSE);       /* queue up desired devices */
```

```
   qdevice (ESCKEY);
   qdevice (F7KEY);
   qdevice (F8KEY);
   while (TRUE) {
      dev = qread (&val);
      switch (dev) {
      case ESCKEY:
      case RIGHTMOUSE:
         exit(0);
      case F7KEY:
         if (val == FALSE) break;  /* do nothing for key release */
         gamma += delta;
         update_ramps (gamma);
         show_val (gamma);
         break;
      case F8KEY:
         if (val == FALSE) break;  /* do nothing for key release */
         gamma -= delta;
         update_ramps (gamma);
         show_val (gamma);
         break;
      case REDRAW:
         show_val (gamma);
      default:
         break;
      }
   }
}
```

Related Information

Pixel Coverage in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* describes aspects of antialiasing (smoothing lines for screen display) related to color variation in the pixels.

---

iobounce.c Example C Language Program

```
/*
iobounce.c:

  A "pool" ball that "bounces" around a 2-d "surface".
  RIGHTMOUSE stops ball
  MIDDLEMOUSE increases y velocity
  LEFTMOUSE increases x velocity
*/

#include <gl/gl.h>
#include <gl/device.h>

#define XMIN 100
#define YMIN 100
#define XMAX 900
#define YMAX 700

long xvelocity = 0, yvelocity = 0;

main()
{

   Device dev;
   short val;

   initialize();
   while (TRUE) {
      while (qtest()) {
         dev = qread(&val);
         switch (dev) {
         case LEFTMOUSE:          /* increase xvelocity */
            if (xvelocity >= 0)
               xvelocity++;
            else
```

```
                xvelocity--;
             break;
          case MIDDLEMOUSE:        /* increase yvelocity */
             if (yvelocity >= 0)
                yvelocity++;
             else
                yvelocity--;
             break;
          case RIGHTMOUSE:         /* stop ball */
             xvelocity = yvelocity = 0;
             break;
          case ESCKEY:
             gexit();
             exit(0);
          }
       }
       drawball();
    }
}
initialize()
{
    int gid;

    prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4, YMAXSCREEN/4,
        YMAXSCREEN*3/4);
    gid = winopen("iobounce");
    doublebuffer();
    gconfig();
    shademodel(FLAT);
    ortho2(XMIN - 0.5, XMAX + 0.5, YMIN - 0.5, YMAX + 0.5);
    qdevice(ESCKEY);
    qdevice(REDRAW);
    qdevice(LEFTMOUSE);
    qdevice(MIDDLEMOUSE);
    qdevice(RIGHTMOUSE);
    qenter(REDRAW,gid);
}
drawball()
{
    static xpos = 500,ypos = 500;
    long radius = 10;

    color(BLACK);
    clear();
    xpos += xvelocity;
    ypos += yvelocity;
    if (xpos > XMAX - radius ||
        xpos < XMIN + radius) {
       xpos -= xvelocity;
       xvelocity = -xvelocity;
    }
    if (ypos > YMAX - radius ||
        ypos < YMIN + radius) {
       ypos -= yvelocity;
       yvelocity = -yvelocity;
    }
    color(YELLOW);
    circfi(xpos, ypos, radius);
    swapbuffers();
}
```

Related Information

localatten.c Example C Language Program

```
/*
This program demonstrates the effect of light attenuation
by continuously moving a local light toward a flat plate.

It draws a flat green plate at z = 0; -1.0 > x, y > 1.0.
The eye is 6 units above, looking down.  A
light bounces up and down in the range 0.1 > z > 1.5, and
x = y = 0.  The lighting model attenuates intensity with
distance, so the center of the plate gets brighter as
the light moves closer.  The character string printed at
the lower left of the plate shows the height of the
light.
Note that the color is set after the cmov()
command -- the cmov() actually sends a vertex through the
transformation, and it will set the current color.
If you move the cpack() command just above the cmov()
command, the character string will be lighted and will
appear in varying shades of green.
*/

#include <gl/gl.h>
#include <stdio.h>

Matrix idmat = {
   1.0,0.0,0.0,0.0,
   0.0,1.0,0.0,0.0,
   0.0,0.0,1.0,0.0,
   0.0,0.0,0.0,1.0};

float  green_material[] = {
   DIFFUSE,  0.0, 1.0, 0.0,
   LMNULL};

float  local_white_light[] = {
   LCOLOR, 1.0, 1.0, 1.0,
   POSITION, 0.0, 0.0, 1.0, 1.0,
   LMNULL};

float  light_model[] = {
   AMBIENT, 0.0, 0.0, 0.0,
   LOCALVIEWER, 0.0,
   ATTENUATION, 1.0, 1.0,
   LMNULL};

/*
** draw_plate draws a flat plate covering the
** range -1.0 <= x, y <= 1.0 and z = 0. using
** n^2 rectangles.  All the normal vectors are
** perpendicular to the plate.
*/
draw_plate(n)
long n;
{
   long i, j;
   float p0[3], p1[3], p2[3], p3[3];
   float n0[3];

   n0[0] = n0[1] = 0.0;
   n0[2] = 1.0;
   p0[2] = p1[2] = p2[2] = p3[2] = 0.0;
   for (i = 0; i < n; i++) {
      p0[0] = p1[0] = -1.0 + 2.0*i/n;
      p2[0] = p3[0] = -1.0 + 2.0*(i+1)/n;
      for (j = 0; j < n; j++) {
         p0[1] = p3[1] = -1.0 + 2.0*j/n;
         p1[1] = p2[1] = -1.0 + 2.0*(j+1)/n;
         bgnpolygon();
```

```
         n3f(n0);
         v3f(p0);
         n3f(n0);
         v3f(p1);
         n3f(n0);
         v3f(p2);
         n3f(n0);
         v3f(p3);
         endpolygon();
      }
   }
}
/*
** Tell the Graphics Library to DEFINE a
** lighting calculation that accounts for
** diffuse and ambient reflection.  In
** addition, this lighting calculation
** includes a local light whose emitted
** light is attenuated as a function of
** distance from the object.
*/
def_light_calc()
{
   lmdef(DEFLMODEL, 1, 10, light_model);
   lmdef(DEFMATERIAL, 1, 5, green_material);
   lmdef(DEFLIGHT, 1, 10, local_white_light);
}
/*
** Tell the Graphics Library to USE the lighting
** calculation that we defined earlier.
*/
use_light_calc()
{
   lmbind(LMODEL, 1);
   lmbind(LIGHT1, 1);
   lmbind(MATERIAL, 1);
}
main()
{
   float dist;
   long flag = 1;
   char str[32];
   keepaspect(1, 1);
   prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
       YMAXSCREEN*3/4);
   winopen("local");
   RGBmode();
   doublebuffer();
   gconfig();
   /*
    ** Use mmode() to set up projection and
    ** viewing matrices for lighting.
     */
   mmode(MVIEWING);
   perspective(400, 1.0, 0.5, 10.0);
   loadmatrix(idmat);
   lookat(0.0,0.0,6.0,0.0,0.0,0.0,0);
   def_light_calc();
   use_light_calc();
   dist = 1.0;
   while (TRUE) {
      if (flag) {
         dist += .01;
         if (dist > 1.5) flag = 1 - flag;
      }
```

```
      else {
         dist -= .01;
         if (dist < 0.1) flag = 1 - flag;
      }
      cpack(0);
      clear();
      sprintf(str, "Light Distance: %1.2f", dist);
      cmov2(-1.5, -1.5);
      cpack(0xffffff);
      charstr(str);
      pushmatrix();
            /*
           ** Change the position of the local light
           ** by REDEFINING and REBINDING the light.
           ** Repositioning the light changes the
           ** illumination of the plate for two reasons:
           ** 1) the affect of attenuation, and
           ** 2) the light direction vector from a
           ** vertex on the plate to the repositioned
           ** light source has changed.
           */
      local_white_light[7] = dist;
      lmdef(DEFLIGHT, 1, 10, local_white_light);
      lmbind(LIGHT1, 1);
      draw_plate(20);
      popmatrix();
      swapbuffers();
   }
}
```

Related Information

The **cpack** subroutine, **loadmatrix** subroutine, **RGBmode** subroutine .

---

makefile Example C Language Program

```
all: aliasback aliasfore antialias backface boxcirc circuit
colored curve1 curve2 curve3 curved cylinder1 cylinder2 depthcue
doily doublebuff draw font3 gammaramp iobounce local octahedron
overlay paint patch1 pick1 plate popup prompt screenrotate
select1 setshade sunflower text tpbig vlsi worms xfonts zbuffer1
zbuffer2 zoing

aliasback.o:   aliasback.c
   cc -g -c aliasback.c
aliasback:   aliasback.o
   cc -g -o aliasback aliasback.o -lm -lgl

aliasfore.o:   aliasfore.c
   cc -g -c aliasfore.c
aliasfore:   aliasfore.o
   cc -g -o aliasfore aliasfore.o -lm -lgl

antialias.o:   antialias.c
   cc -g -c antialias.c
antialias:   antialias.o
   cc -g -o antialias antialias.o -lm -lgl

backface.o:   backface.c
   cc -g -c backface.c
backface:   backface.o
   cc -g -o backface backface.o -lm -lgl

boxcirc.o:   boxcirc.c
   cc -g -c boxcirc.c
boxcirc:   boxcirc.o
   cc -g -o boxcirc boxcirc.o -lm -lgl
```

```
circuit.o:   circuit.c
   cc -g -c circuit.c
circuit:   circuit.o
   cc -g -o circuit circuit.o -lm -lgl

colored.o:   colored.c
   cc -g -c colored.c
colored:   colored.o
   cc -g -o colored colored.o -lm -lgl

curve1.o:   curve1.c
   cc -g -c curve1.c
curve1:   curve1.o
   cc -g -o curve1 curve1.o -lm -lgl

curve2.o:   curve2.c
   cc -g -c curve2.c
curve2:   curve2.o
   cc -g -o curve2 curve2.o -lm -lgl

curve3.o:   curve3.c
   cc -g -c curve3.c
curve3:   curve3.o
   cc -g -o curve3 curve3.o -lm -lgl

curved.o:   curved.c
   cc -g -c curved.c
curved:   curved.o
   cc -g -o curved curved.o -lm -lgl

cylinder1.o:   cylinder1.c
   cc -g -c cylinder1.c
cylinder1:   cylinder1.o
   cc -g -o cylinder1 cylinder1.o -lm -lgl

cylinder2.o:   cylinder2.c
   cc -g -c cylinder2.c
cylinder2:   cylinder2.o
   cc -g -o cylinder2 cylinder2.o -lm -lgl

depthcue.o:   depthcue.c
   cc -g -c depthcue.c
depthcue:   depthcue.o
   cc -g -o depthcue depthcue.o -lm -lgl

doily.o:   doily.c
   cc -g -c doily.c
doily:   doily.o
   cc -g -o doily doily.o -lm -lgl

doublebuff.o:   doublebuff.c
   cc -g -c doublebuff.c
doublebuff:   doublebuff.o
   cc -g -o doublebuff doublebuff.o -lm -lgl

draw.o:   draw.c
   cc -g -c draw.c
draw:   draw.o
   cc -g -o draw draw.o -lm -lgl

font3.o:   font3.c
   cc -g -c font3.c
font3:   font3.o
   cc -g -o font3 font3.o -lm -lgl

gammaramp.o:   gammaramp.c
   cc -g -c gammaramp.c
gammaramp:   gammaramp.o
   cc -g -o gammaramp gammaramp.o -lm -lgl

iobounce.o:   iobounce.c
   cc -g -c iobounce.c
iobounce:   iobounce.o
   cc -g -o iobounce iobounce.o -lm -lgl
```

```
local.o:   local.c
   cc -g -c local.c
local:   local.o
   cc -g -o local local.o -lm -lgl

octahedron.o:   octahedron.c
   cc -g -c octahedron.c
octahedron:   octahedron.o
   cc -g -o octahedron octahedron.o -lm -lgl

overlay.o:   overlay.c
   cc -g -c overlay.c
overlay:   overlay.o
   cc -g -o overlay overlay.o -lm -lgl

paint.o:   paint.c
   cc -g -c paint.c
paint:   paint.o
   cc -g -o paint paint.o -lm -lgl

patch1.o:   patch1.c
   cc -g -c patch1.c
patch1:   patch1.o
   cc -g -o patch1 patch1.o -lm -lgl

pick1.o:   pick1.c
   cc -g -c pick1.c
pick1:   pick1.o
   cc -g -o pick1 pick1.o -lm -lgl

plate.o:   plate.c
   cc -g -c plate.c
plate:   plate.o
   cc -g -o plate plate.o -lm -lgl

popup.o:   popup.c
   cc -g -c popup.c
popup:   popup.o
   cc -g -o popup popup.o -lm -lgl

prompt.o:   prompt.c
   cc -g -c prompt.c
prompt:   prompt.o
   cc -g -o prompt prompt.o -lm -lgl

screenrotate.o:   screenrotate.c
   cc -g -c screenrotate.c
screenrotate:   screenrotate.o
   cc -g -o screenrotate screenrotate.o -lm -lgl

select1.o:   select1.c
   cc -g -c select1.c
select1:   select1.o
   cc -g -o select1 select1.o -lm -lgl

setshade.o:   setshade.c
   cc -g -c setshade.c
setshade:   setshade.o
   cc -g -o setshade setshade.o -lm -lgl

sunflower.o:   sunflower.c
   cc -g -c sunflower.c
sunflower:   sunflower.o
   cc -g -o sunflower sunflower.o -lm -lgl

text.o:   text.c
   cc -g -c text.c
text:   text.o
   cc -g -o text text.o -lm -lgl

tpbig.o:   tpbig.c
   cc -g -c tpbig.c
tpbig:   tpbig.o
   cc -g -o tpbig tpbig.o -lm -lgl
```

```
vlsi.o:   vlsi.c
   cc -g -c vlsi.c
vlsi:   vlsi.o
   cc -g -o vlsi vlsi.o -lm -lgl

worms.o:   worms.c
   cc -g -c worms.c
worms:   worms.o
   cc -g -o worms worms.o -lm -lgl

xfonts.o:   xfonts.c
   cc -g -c xfonts.c
xfonts:   xfonts.o
   cc -g -o xfonts xfonts.o -lm -lgl -lX11

zbuffer1.o:   zbuffer1.c
   cc -g -c zbuffer1.c
zbuffer1:   zbuffer1.o
   cc -g -o zbuffer1 zbuffer1.o -lm -lgl

zbuffer2.o:   zbuffer2.c
   cc -g -c zbuffer2.c
zbuffer2:   zbuffer2.o
   cc -g -o zbuffer2 zbuffer2.o -lm -lgl

zoing.o:   zoing.c
   cc -g -c zoing.c
zoing:   zoing.o
   cc -g -o zoing zoing.o -lm -lgl
```

Related Information

---

octahedron.c Example C Language Program

```
/* Example: octahedron.c */

/* This example program draws a multicolored octahedron spinning
 * like a top. The drawoctahedron subroutine demonstrates how
 * triangle meshes, including the swaptmesh() subroutine,
 * are used.
 * The c3f() subroutine sets vertex colors; you can ignore these
 * calls if you are studying the logic of mesh drawing.
 * All rotation and hidden surface removal are handled in the
 * main() routine.  The calculations of the rotation angles are
 * based on Euler's equations for a spinning top.
 * On adapters without a z-buffer, this program will not display
 * the octahedron correctly, since it employs the z-buffer to
 * remove hidden surfaces.
 */

#include <gl/gl.h>

float octdata[6][3] = {   /* positions of the six vertices of octahedron */
{
   1.0, 0.0, 0.0},
   {0.0, 1.0, 0.0},
   {0.0, 0.0, 1.0},
   {-1.0, 0.0, 0.0},
   {0.0, -1.0, 0.0},
   {0.0, 0.0, -1.0}
};

float colordata[6][3] = {  /* colors of the vertices */
   {1.0, 0.0, 0.0},         /* red */
   {0.0, 1.0, 0.0},         /* green */
   {0.0, 0.0, 1.0},         /* blue */
   {0.0, 1.0, 1.0},         /* cyan */
   {1.0, 0.0, 1.0},         /* magenta */
   {1.0, 1.0, 0.0}
};          /* yellow */
```

```
void drawoctahedron()
{
   bgntmesh();              /* use triangle mesh to draw the
                               octahedron */
   c3f (colordata[0]);
   v3f (octdata [0]);
   c3f (colordata[1]);
   v3f (octdata[1]);
   c3f (colordata[2]);
   v3f (octdata[2]);        /* vertices in first triangle are
                                  0, 1, 2 */
   c3f (colordata[3]);
   v3f (octdata[3]);        /* second triangle - 1, 2, 3 */
   swaptmesh();             /* swap vertex order (2,3) -> (3,2) */
        c3f (colordata[4]);
   v3f (octdata[4]);        /* third triangle - 3, 2, 4 */
   c3f (colordata[0]);
   v3f (octdata[0]);        /* fourth triangle - 2, 4, 0 */
   swaptmesh();             /* swap vertex order (4,0) -> (0,4) */
   c3f (colordata[5]);
   v3f (octdata[5]);        /* fifth triangle - 0, 4, 5 */
   c3f (colordata[3]);
   v3f (octdata[3]);        /* sixth triangle - 4, 5, 3 */
   swaptmesh();             /* swap vertex order (5,3) -> (3,5) */
   c3f (colordata[1]);
   v3f (octdata[1]);        /* seventh trinagle - 3, 5, 1 */
   c3f (colordata[0]);
   v3f (octdata [0]);       /* eigth triangle - 5, 1, 0 */
   endtmesh();
}
main()
{
   long prec=0, spin=0;
   long i;
   prefposition(100, 500, 100, 500);   /* location of window to
                                          be opened */
   winopen("octahedron");
   ortho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);
   zbuffer(TRUE);      /* hidden surfaces removed with z buffer */
   doublebuffer();     /* for smooth motion */
   RGBmode();          /* direct color mode */
   gconfig();          /* reconfigure for RGBmode and
                          doublebuffer */
   for (i=0; i<1500; i++) {
      pushmatrix();                /* save viewing transformation */
      translate (0.0, -1.0, 0.0);  /* make origin the
                                      bottom of the oct */
      rotate (prec, 'y');          /* precession of axis */
      rotate (323, 'x');           /* inclination of 32.3
                                      degrees */
      rotate (spin, 'y');          /* spin around axis */
      translate (0.0, 1.0, 0.0);   /* center in the middle of
                                      the window */
      prec += 15;
      spin += 45;
      if (prec > 3600) prec -= 3600;
      if (spin > 3600) spin -= 3600;
      cpack(0);                    /* color black */
      clear();
      zclear();                    /* clear the z buffer */
      drawoctahedron();
      swapbuffers();               /* show completed drawing */
      popmatrix();                 /* restore viewing
                                      transformation */
   }
```

```
}
```

Related Information

Triangular Meshes in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* explains how to specify three-dimensional objects that are composed of triangular faces.

---

overlay.c Example C Language Program

```
/*
overlay.c

This program demonstrates how to use overlay bitplanes.
*/

#include <gl/gl.h>
#include <gl/device.h>

main ()
{
   Colorindex heat;
   int i, xpos = 1013, ypos = 1013, rampup();
   float xspd = 0.0, yspd = 0.0, yaccel = -1.0;
   float yacc = -.4,  yreflect = -0.6;

   prefposition(0, XMAXSCREEN, 0, YMAXSCREEN);
   winopen ("overlay");
   doublebuffer ();
   overlay (2);
   gconfig ();
   get_cmap();
   drawmode (OVERDRAW);             /* Get into the overlay bitplanes */
   mapcolor (1, 255, 0, 0);
   mapcolor (2, 0, 255, 0);
   mapcolor (3, 0, 255, 255);
   color (1);
   rectfi (200, 200, 300, 300);     /* Draw some rectangles for a ball to roll
   under. */
   color (2);
   rectfi (500, 500, 600, 600);
   color (3);
   rectfi (800, 800, 900, 900);
   rectfi (850, 500, 950, 600);
   rectfi (750, 400, 850, 500);
   rectfi (650, 300, 750, 400);
   drawmode (NORMALDRAW);
   mapcolor (0, 0, 0, 0);
   mapcolor (1, 255, 255, 0);
   mapcolor (10, 255, 255, 255);
   rampup(12, 82, 255, 255, 0, 255, 0, 0);
   setbell(1);
   while (!getbutton (MIDDLEMOUSE)) {
      for (i = 0; i < 1013 && !getbutton(MIDDLEMOUSE); i = i + 3)
      {
         color (BLUE);                /* Roll the ball up and
                  to the right */
         clear ();
         if ((i==210) || (i==510) || (i==810))
            ringbell();                /* Ring the bell everytime */
         if ((i>250) && (i<550))      /* the ball gets to the    */
            color (2);                /* next rectangle.         */
         else if ((i>550) && (i<850))
            color (5);                /* change the ball's color */
         else if (i>850)
            color (10);
         else
```

```
            color (1);
         circfi (i, i, 10);
         swapbuffers ();
      }
      yspd = 0.0;
      for (heat=82, ypos=1013; ypos>=5 && !getbutton(MIDDLEMOUSE);
          ypos+=yspd)
      {
         color (BLUE);        /* drop the ball back to the bottom */
         clear ();
         color (heat--);      /* change the ball's color          */
         yspd += yacc;        /* as it falls.                     */
         circfi (1013, ypos, 10);
         swapbuffers ();
      }
      yspd = -60.0;
      for (xpos=1013, ypos=10; xpos>=-0&& !getbutton(MIDDLEMOUSE);
          xpos -= 5)
      {
         if (ypos <= 10)      /* roll the ball back to the beginning, */
            yspd *= yreflect;  /* and keep updating its'   */
         color (BLUE);        /* bounce-ability per frame */
         clear ();
         color (1);
         ypos += yspd;
         yspd += yaccel;
         circfi (xpos, ypos, 10);
         swapbuffers ();
      }
   }
   drawmode (OVERDRAW);     /* clean up the overlay bitplanes */
   color (0);
   clear();
   drawmode(NORMALDRAW);
   restore_cmap();          /* restore the color map */
   greset();
   gexit ();
}
/*
Make a color ramp.  Make an interpolated ramp from the 1st
arguement's index to the second one. The 3rd, and 4th are red's
low and hi indexes (5&6 green's, 7&8 are blue's).
*/
#define round(n)     ((int) (n + 0.5))
rampup(first_lutv,last_lutv,minR,maxR,minG,maxG,minB,maxB)
unsigned short first_lutv, last_lutv, /* Start & end ramp values. */
minR, maxR,                           /* Low and high red, */
minG, maxG,                           /* green, */
minB, maxB;                           /* and blue values */
{
   unsigned short len_red, len_green,
      len_blue,                        /* Length of each color */
     i;                                /* Counter for number of steps */
   short red, gre, blu;                /* lut values */
   float rdx, gdx, bdx,                /* Sizes of rgb increments */
     r, g, b,                          /* A position on the ramp */
     steps;                            /* No. of steps along the ramp
                                          at which intensity assignments
                                          will be made */
   /* Determine length of ramp*/
   steps = (float) (last_lutv-first_lutv + 1);
   len_red   = (maxR - minR);          /* determine length of red */
   len_green = (maxG - minG);          /* determine length of green */
   len_blue  = (maxB - minB);          /* determine length of blue */
   rdx = (float) len_red   / steps;    /* compute step   */
```

```
   gdx = (float) len_green / steps;    /* sizes of r, g, */
   bdx = (float) len_blue  / steps;    /* and b values   */
   r = minR;                           /* Assign starting */
   g = minG;                           /* indexes for each */
   b = minB;                           /* color value      */
   for (i = first_lutv; i <= last_lutv; i++) {
      red = (short) round(r);           /* Round off the */
      gre = (short) round(g);           /* given r, g,   */
      blu = (short) round(b);           /* and b values  */
      mapcolor(i, red, gre, blu);       /* assign next color
               map index */
      r += rdx;                         /* Increment the */
      g += gdx;                         /* color indexes */
      b += bdx;
   }
}
#define MAXCOLI 255
static short CarrayR[MAXCOLI+1], CarrayG[MAXCOLI+1], CarrayB[MAXCOLI+1];
unsigned short index;
get_cmap() {
   short rcomp, gcomp, bcomp;
   for (index=0; index<=MAXCOLI; index++) {
      getmcolor(index,&rcomp, &gcomp, &bcomp);
      CarrayR[index] = rcomp;
      CarrayG[index] = gcomp;
      CarrayB[index] = bcomp;
   }
}
restore_cmap() {
   for (index=0; index<=MAXCOLI; index++)
      mapcolor(index,CarrayR[index],
          CarrayG[index], CarrayB[index]);
}
```

Related Information

The **gconfig** subroutine, **getmcolor** subroutine, **overlay** subroutine .

---

paint.c Example C Language Program

```
/*
 *        paint.c -
 *                A minimal object space paint program.
 *
 *                Paul Haeberli - 1985
 */
#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>
#define ABS( a )        (((a) > 0) ? (a) : -(a))
#define MOUSE               12
#define TABLET              13
#define DRAWLINE             2
#define NEWCOLOR             3
#define CLEAR                4
#define NEWSIZE              5
#define MOUSEXMAP(x)        ( (100.0*((x)-xorg))/(xsize) )
#define MOUSEYMAP(y)        ( (100.0*((y)-yorg))/(ysize) )
#define BPSCALE 16.0
struct event {
   struct event *next;
   int func;
```

```
   float arg1;
   float arg2;
   float arg3;
   float arg4;
};
int xsize, ysize;
int xorg, yorg;
int mx, my;
int bpx, bpy;
int mmiddle, mleft;
int curcolor = 7;
int lastcurcolor = 7;
float curx, cury, cursize;
int curdev = MOUSE;
struct event *histstart = 0;
struct event *histend = 0;
float xpos, ypos;
int pendown;
int brushsides;
float brushcoords[30][2];
int menu;
main()
{
   cursize = 1.0;
   prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
       YMAXSCREEN*3/4);
   winopen("paint");
   menu = defpup("paint %t|mouse|tablet");
   makebrush();
   makeframe();
   getinput();
}
getinput()
{
   Device dev;
   short val;
   float x, y;
   while(TRUE) {
      do {
         dev = qread(&val);
         switch (dev) {
         case MOUSEX:
            mx = val;
            if (curdev == MOUSE)
               xpos = MOUSEXMAP(val);
            break;
         case MOUSEY:
            my = val;
            if (curdev == MOUSE)
               ypos = MOUSEYMAP(val);
            break;
         case BPADX:
            bpx = val;
            if (curdev == TABLET)
               xpos = val/BPSCALE;
            break;
         case BPADY:
            bpy = val;
            if (curdev == TABLET)
               ypos = val/BPSCALE;
            break;
         case BPAD0:
            if (curdev == TABLET)
               pendown = val;
```

```
            if (val) {
               curx = xpos = bpx/BPSCALE;
               cury = ypos = bpy/BPSCALE;
            }
            break;
         case MENUBUTTON:
            if(val) {
               switch (dopup(menu)) {
               case 1:
                  curdev = MOUSE;
                  break;
               case 2:
                  curdev = TABLET;
                  break;
               }
            }
            break;
         case MIDDLEMOUSE:
            mmiddle = val;
            if (mmiddle) {
               clearscreen();
               history(CLEAR);
            }
            break;
         case LEFTMOUSE:
            mleft = val;
            if (mleft) {
               if (!inside(mx-xorg, my-yorg,
                    0, xsize, 0, ysize, 0)) {
                  newcolor(getapixel(mx,my));
                  history(NEWCOLOR,(float)curcolor);
               }
            }
            if (curdev == MOUSE) {
               pendown = val;
               curx = xpos = MOUSEXMAP(mx);
               cury = ypos = MOUSEYMAP(my);
            }
            break;
         case REDRAW:
            makeframe();
            replay();
            break;
         case ESCKEY:
            gexit();
            exit(0);
            break;
         }
      } while (qtest());
      if (pendown) {
         x = xpos;
         y = ypos;
         drawbrush(x,y,curx,cury);
         history(DRAWLINE,x,y,curx,cury);
         curx = x;
         cury = y;
      }
   }
}
```

```
clearscreen()
{
   color(curcolor);
   clear();
}
newcolor(c)
int c;
{
   lastcurcolor = curcolor;
   curcolor = c;
   paintport();
}
makeframe()
{
   qdevice(ESCKEY);
   qdevice(MOUSEX);
   qdevice(MOUSEY);
   qdevice(MENUBUTTON);
   qdevice(MIDDLEMOUSE);
   qdevice(LEFTMOUSE);
   qdevice(BPADX);
   qdevice(BPADY);
   qdevice(BPAD0);
   getsize(&xsize,&ysize);
   getorigin(&xorg,&yorg);
   paintport();
   newcolor(0);
   clearscreen();
   newcolor(255);
   newcolor(128+32);
}
paintport()
{
   viewport(0,xsize-1,0,ysize);
   ortho2(-0.5,99.5,-0.5,99.5);
}
inside(x,y,xmin,xmax,ymin,ymax,fudge)
int x, y, xmin, xmax, ymin, ymax, fudge;
{
   if (x>xmin-fudge && x<xmax+fudge &&
       y>ymin-fudge && y<ymax+fudge)
      return 1;
   else
      return 0;
}
makebrush()
{
   int i;
   brushsides = 4;
   brushcoords[0][0] = -0.6;
   brushcoords[0][1] = -0.2;
   brushcoords[1][0] = -0.6;
   brushcoords[1][1] = -0.4;
   brushcoords[2][0] =  0.6;
   brushcoords[2][1] =  0.2;
   brushcoords[3][0] =  0.6;
   brushcoords[3][1] =  0.4;
   for (i=0; i<brushsides; i++) {
      brushcoords[i][0] = 0.5*brushcoords[i][0];
      brushcoords[i][1] = 0.5*brushcoords[i][1];
   }
}
```

```
drawbrush(x,y,ox,oy)
float x, y, ox, oy;
{
   register int i, n;
   register float dx, dy;
   float quad[4][2];
   float delta;
   int c;
   dx = ox-x;
   dy = oy-y;
   if (lastcurcolor != curcolor) {
      delta = sqrt(dx*dx+dy*dy);
      if (delta<0.001)
         return;
      c = (int) (curcolor + (lastcurcolor-curcolor) *
          (ABS(dx)/delta) );
      color(c);
   }
   else
      color(curcolor);
   pushmatrix();
   translate(x,y,0.0);
   for (i=0; i<brushsides; i++) {
      n = (i+1) % brushsides;
      quad[0][0] = brushcoords[i][0];
      quad[0][1] = brushcoords[i][1];
      quad[1][0] = brushcoords[n][0];
      quad[1][1] = brushcoords[n][1];
      quad[2][0] = quad[1][0]+dx;
      quad[2][1] = quad[1][1]+dy;
      quad[3][0] = quad[0][0]+dx;
      quad[3][1] = quad[0][1]+dy;
      polf2(4,quad);
   }
   polf2(brushsides,brushcoords);
   popmatrix();
}
history(func,arg1,arg2,arg3,arg4)
int func;
float arg1, arg2, arg3, arg4;
{
   register struct event *e, *n;
   e = (struct event *)malloc(sizeof(struct event));
   switch (func) {
   case CLEAR:
      zaphistory();
      history(NEWCOLOR,(float)curcolor);
      break;
   case NEWCOLOR:
   case DRAWLINE:
      e->func = func;
      e->arg1 = arg1;
      e->arg2 = arg2;
      e->arg3 = arg3;
      e->arg4 = arg4;
      e->next = 0;
      if (!histstart) {
         histstart = histend = e;
      }
      else {
         histend->next = e;
         histend = e;
      }
      break;
   }
}
```

```
zaphistory()
{
   register struct event *e, *n;
   e = histstart;
   while (e) {
      n = e->next;
      free(e);
      e = n;
   }
   histstart = histend = 0;
}
replay()
{
   register struct event *e;
   register int i;
   i = 0;
   e = histstart;
   while (e) {
      switch (e->func) {
      case NEWCOLOR:
         newcolor((int)e->arg1);
         break;
      case DRAWLINE:
         drawbrush(e->arg1,e->arg2,e->arg3,e->arg4);
         break;
      case CLEAR:
         clearscreen();
         break;
      }
      e = e->next;
      i++;
   }
}
/*
 *        getapixel -
 *                Read a pixel from a specific screen location.
 *
 */
getapixel(mousex, mousey)
short mousex, mousey;
{
   short pixel;
   int   xmin, ymin;    /* Convert position to window relative coordinates */
   getorigin(&xmin, &ymin);
   mousex -= xmin;
   mousey -= ymin;
   rectread(mousex, mousey, mousex+1, mousey+1, &pixel);
   return(pixel);
}
```

Related Information

The **getorigin** subroutine, **getsize** subroutine, **ortho or ortho2** subroutine, **rectread or lrectread** subroutine, **viewport** subroutine .

---

patch1.c Example C Language Program

```
/* Example C Language Program patch1.c */
/*
This program draws three surface patches. First, one based on
Bezier curves, then one based on B-Spline curves, and
finally one based on Cardinal curves.
*/
```

```
#include <gl/gl.h>
#include <gl/device.h>

Matrix beziermatrix = {
  { -1,  3, -3, 1 },
  {  3, -6,  3, 0 },
  { -3,  3,  0, 0 },
  {  1,  0,  0, 0 }};

Matrix cardinalmatrix = {
  { -0.5,  1.5, -1.5,  0.5 },
  {  1.0, -2.5,  2.0, -0.5 },
  { -0.5,  0.0,  0.5,  0.0 },
  {  0.0,  1.0,  0.0,  0.0 }};

Matrix bsplinematrix = {
  { -1.0/6.0, 3.0/6.0, -3.0/6.0, 1.0/6.0 },
  {  3.0/6.0, -6.0/6.0, 3.0/6.0, 0.0     },
  { -3.0/6.0,  0.0,     3.0/6.0, 0.0     },
  {  1.0/6.0,  4.0/6.0, 1.0/6.0, 0.0     }};

#define BEZIER 1

#define CARDINAL 2

#define BSPLINE 3

Coord geomx[4][4] = {
  {   0.0, 100.0, 200.0, 300.0} ,
  {   0.0, 100.0, 200.0, 300.0},
  { 700.0, 600.0, 500.0, 400.0},
  { 700.0, 600.0, 500.0, 400.0}};

Coord geomy[4][4] = {
  { 400.0, 500.0, 600.0, 700.0},
  {   0.0, 100.0, 200.0, 300.0},
  {   0.0, 100.0, 200.0, 300.0},
  { 400.0, 500.0, 600.0, 700.0}};

Coord geomz[4][4] = {
  { 100.0, 200.0, 300.0, 400.0 },
  { 100.0, 200.0, 300.0, 400.0 },
  { 100.0, 200.0, 300.0, 400.0 },
  { 100.0, 200.0, 300.0, 400.0 }};

main()
{
  Device dev;
  short val;

  initialize();
  while (TRUE) {
    if (qtest()) {
      dev = qread(&val);
      if (dev == ESCKEY) {
        gexit();
        exit();
      }
      else if (dev == REDRAW) {
        reshapeviewport();
        drawpatch();
      }
    }
  }
}

initialize()
{
  int gid;
```

```
  prefsize(400,400);
  gid = winopen("patch1");
  qdevice(ESCKEY);
  qdevice(REDRAW);
  qenter(REDRAW,gid);
  ortho(-100.0, 800.0, -100.0, 800.0, -800.0, 100.0);
}
drawpatch()
{
  int i,j,xx,yy;
  color(BLACK);
  clear();
  defbasis(BEZIER, beziermatrix);    /* define a basis matrix
                                        called BEZIER */
  defbasis(CARDINAL,cardinalmatrix); /* define a basis matrix
                                        called CARDINAL */
  defbasis(BSPLINE,bsplinematrix);   /* define a basis matrix
                                        called BSPLINE */
  patchbasis(BEZIER,BEZIER);     /* a Bezier basis will be used
                                    for both directions in the
                                    first patch */
  patchcurves(4,7);              /* seven curve segments will be
                                    drawn in the u direction and
                                    four in the v direction */
  patchprecision(20,20);         /* the curve segments in u
                                    direction will consist of 20
                                    line segments (the lowest
                                    multiple of vcurves greater
                                    than usegments) and the curve
                                    segments in the v direction
                                    will consist of 21 line
                                    segments (the lowest multiple
                                    of ucurves greater than
                                    vsegments) */
  color(RED);
  patch(geomx,geomy,geomz);      /* the patch is drawn based on
                                    the sixteen specified control
                                    points */
  patchbasis(CARDINAL,CARDINAL); /* the bases for both directions
                                    are reset */
  color(GREEN);
  patch(geomx,geomy,geomz);      /* another patch is drawn using
                                    the same control points but a
                                    different basis */
  patchbasis(BSPLINE,BSPLINE);   /* the bases for both directions
                                    are reset again */
  color(BLUE);
  patch(geomx,geomy,geomz);      /* a third patch is drawn */
  color(WHITE);                  /* show the control points */
  for ( i = 0 ; i < 4 ; i++ )  {
    for ( j = 0 ; j < 4 ; j++ )  {
         for ( xx = -2 ; xx < 2 ; xx++) {
            for ( yy = -2 ; yy < 2 ; yy++) {
               pnt( geomx[i][j] + (Coord) xx , geomy[i][j] + (Coord) yy ,
               geomz[i][j] );
        }
      }
    }
  }
}
/*
  Changes:
     - The prefposition changed to a prefsize (400,400);
       Was:  prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4, YMAXSCREEN/4,
```

```
            YMAXSCREEN*3/4);
     - The ortho parameters where changed
         ortho(0.0-20.0, (float)(XMAXSCREEN*3/4),
             0.0-20.0, (float)(YMAXSCREEN*3/4),
             (float)XMAXSCREEN, -(float)XMAXSCREEN);
       Now:
         ortho(-100.0, 800.0, -100.0, 800.0, -800.0, 100.0);
*/
```

Related Information

The **patch** subroutine, **patchbasis** subroutine, **patchcurves** subroutine, **patchprecision** subroutine .

Drawing Surfaces in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* describes the mathematics and programming statements involved in creating three-dimensional surfaces (or patches) with wire frames of curve segments.

---

pick1.c Example C Language Program

```
/*
 *  pick1.c:
 *
 *A sample picking program. Use LEFTMOUSE to "pick" the
 *background, a circle, or the square.
 */

#include <gl/gl.h>
#include <gl/device.h>

#define PICKS 1

main()
{
   short namebuffer[50];
   long numpicked;
   short val, i, j, k;
   Device dev;

   initialize();
   while (TRUE) {
      dev = qread(&val);
      if (val == 0)
         continue;
      switch (dev) {
      case ESCKEY:
         gexit();
         exit(0);
      case REDRAW:
         reshapeviewport();
         color(BLACK);
         clear();
         callobj(PICKS);
         break;
      case LEFTMOUSE:
         pick(namebuffer, 50);
         ortho2(-0.5, XMAXSCREEN + 0.5, -0.5,
             YMAXSCREEN + 0.5);
         callobj(PICKS);
         numpicked = endpick(namebuffer);
         printf("hits: %d; ",numpicked);
         j = 0;
         for (i = 0; i < numpicked; i++) {
            printf(" ");
            k = namebuffer[j++];
            printf("%d ", k);
```

```
            for (;k; k--)
               printf("%d ", namebuffer[j++]);
            printf("|");
         }
         printf("\n");
         break;
      default:
         break;
      }
   }
}
initialize()
{
   int gid;
   prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,
       YMAXSCREEN/4,YMAXSCREEN*3/4);
   gid = winopen("pick1");
   ortho2(-0.5, XMAXSCREEN + 0.5, -0.5, YMAXSCREEN + 0.5);
   qdevice(ESCKEY);
   qdevice(REDRAW);
   qdevice(LEFTMOUSE);
   qdevice(MIDDLEMOUSE);
   qenter(REDRAW,gid);
   initnames();
   makeobj(PICKS);
   color(RED);
   loadname(1);
   rectfi(20,20,100,100);
   loadname(2);
   pushname(3);
   circi(50,500,50);
   loadname(4);
   circi(50,530,60);
   loadname(5);
   move2i(30,30);
   draw2i(32,32);
   closeobj();
}
```

Related Information

The **endpick** subroutine, **initnames** subroutine, **loadname** subroutine, **pick** subroutine, **pushname** subroutine .

---

platelocal.c Example C Language Program

```
/*
This example program uses a blue local light to
illuminate a white flat plate.  By removing the
definition of FIXED_LIGHT in the first line of
the program, the light source will maintain its
position relative to the plate.  This
demonstrates an important concept:  the position
of a light source (or direction if using an
infinite light source) is transformed by the
current transformation matrix at the time it is
bound.

Try changing the local light to an infinite
light and run the program again.  Notice how the
color across the plate is now constant at any
given instant.  Since the plate surface material
```

```
has no specular reflectance, we did not use a
local viewer (remember diffuse reflection is
independent of viewer position).
*/

/* Example C Language Program platelocal.c */
/*
This program draws a flat plate with a simple
local light.  If the line at the top of the file
is left in, the light is fixed, and the plate
moves.  Thus the bright spot on the plate will
appear to move around (on the plate).
Sometimes, the plate gets in front of the light,
and almost disappears, since only the back is
lit.  It does not quite disappear, since there
is a small ambient component for the default
material.

If the statement "#define FIXED_LIGHT" is
deleted, the light is effectively attached to
the moving plate, so the lighted portion of the
plate moves with the plate.
*/

#define FIXED_LIGHT
#include <gl/gl.h>
#include <stdio.h>

Matrix idmat =
   {1.0,0.0,0.0,0.0,
    0.0,1.0,0.0,0.0,
    0.0,0.0,1.0,0.0,
    0.0,0.0,0.0,1.0};

float  white_material[] =   {
  DIFFUSE,  1.0, 1.0, 1.0,
  SPECULAR, 0.0, 0.0, 0.0,
  LMNULL};

float  local_blue_light[] = {
  LCOLOR, 0.0, 0.0, 1.0,
  POSITION, 0.5, 0.5, 0.1, 1.0,
  LMNULL};

/*
** draw_plate draws a flat plate covering
** the range -1.0 <= x, y <= 1.0 and z = 0.0
** using n^2 rectangles.  All the normal vectors are
** perpendicular to the plate.
*/

draw_plate(n)
long n;
{
  long i, j;
  float p0[3], p1[3], p2[3], p3[3];
  float n0[3];
  n0[0] = n0[1] = 0.0;
  n0[2] = 1.0;
  p0[2] = p1[2] = p2[2] = p3[2] = 0.0;
  for (i = 0; i < n; i++) {
    p0[0] = p1[0] = -1.0 + 2.0*i/n;
    p2[0] = p3[0] = -1.0 + 2.0*(i+1)/n;
    for (j = 0; j < n; j++) {
      p0[1] = p3[1] = -1.0 + 2.0*j/n;
      p1[1] = p2[1] = -1.0 + 2.0*(j+1)/n;
      bgnpolygon();
      n3f(n0);
      v3f(p0);
      n3f(n0);
      v3f(p1);
      n3f(n0);
```

```
      v3f(p2);
      n3f(n0);
      v3f(p3);
      endpolygon();
    }
  }
}
/*
** Tell the Graphics Library to DEFINE a
** lighting calculation that accounts for
** diffuse and ambient reflection.  In addition,
** the lighting calculation includes a LOCAL light.
*/
def_light_calc()
{
  lmdef(DEFLMODEL, 1, 0, NULL);
  lmdef(DEFMATERIAL, 1, 9, white_material);
  lmdef(DEFLIGHT, 1, 10, local_blue_light);
}
/*
** Tell the Graphics Library to USE the lighting
** calculation that we defined earlier.
*/
use_light_calc()
{
  lmbind(LMODEL, 1);
  lmbind(LIGHT0, 1);
  lmbind(MATERIAL, 1);
}
main()
{
  int i;
  keepaspect(1, 1);
  prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
      YMAXSCREEN*3/4);
  winopen("local");
  RGBmode();
  doublebuffer();
  gconfig();
    /*
    ** Use mmode() to set up projection and viewing
    ** matrices for lighting.
    */
  mmode(MVIEWING);
  perspective(400, 1.0, 0.5, 10.0);
  loadmatrix(idmat);
  lookat(0.0,0.0,6.0,0.0,0.0,0.0,0);
  def_light_calc();
  use_light_calc();
  for (i = 0; i < 1800; i++) {
    cpack(0);
    clear();
    pushmatrix();
    rot(i*0.5, 'Z');
    rot(i*0.5, 'Y');
    #ifndef FIXED_LIGHT
    lmbind(LIGHT0, 1);
    #endif FIXED_LIGHT
    draw_plate(10);
    popmatrix();
    swapbuffers();
  }
}
```

Related Information

Advanced Lighting Capabilities in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* describes how to manipulate material emission, ambient light, run-time lighting capabilities (Imcolor Subroutine), local viewer, local lights, and light attenuation.

---

popup.c Example C Language Program

```
/*
popup.c:
Demonstrates "how to write your own popup menu" routine
s.
Use LEFTMOUSE instead of RIGHTMOUSE to pop up the me
nus.
*/

#include <gl/gl.h>
#include <gl/device.h>

#define LINE 1

#define POINTS 2

#define CIRCLE 3

#define RECT 4

#define RECTF 5

#define QUIT 6

typedef struct {
  short type;
  char *text;
} popupentry;

popupentry mainmenu[] = {
  {LINE, "Line"},
  {POINTS, "100 points"},
  {CIRCLE, "Filled circle"},
  {RECT, "Outlined rectangle"},
  {RECTF, "Filled rectangle"},
  {QUIT, "Quit"},
  {0, 0}
};
/* mark end of menu */

main()
{
  long win;
  short val, command;

  prefposition(0,XMAXSCREEN,0,YMAXSCREEN);
  win = winopen("popup");
  ortho2(-1.0, 1.0, -1.0, 1.0);
  overlay(2);
  gconfig();

  drawmode(OVERDRAW);
  mapcolor(0, 0, 0, 0);        /* background only */
  mapcolor(1, 120, 120, 120);  /* popup background */
  mapcolor(2, 255, 255, 255);  /* popup text only */
  drawmode(NORMALDRAW);
  qdevice(RIGHTMOUSE);
  qdevice(LEFTMOUSE);
  tie(LEFTMOUSE, MOUSEX, MOUSEY);
  color(0);
  clear();

  while (TRUE) {
    switch(qread(&val)) {
    case REDRAW:
      reshapeviewport();
      drawstuff(command);
```

```
      break;
    case LEFTMOUSE:
      drawstuff(command = popup(mainmenu));
    default:
      break;
    }
  }
}
drawstuff(command)
short command;
{
  register i, j;
  color(0);
  clear();
  color(GREEN);
  switch(command) {
  case LINE:
    move2(-1.0, -1.0);
    draw2(1.0, 1.0);
    break;
  case POINTS:
    for (i =  0; i < 10; i++)
      for (j = 0; j < 10; j++)
        pnt2(i/20.0, j/20.0);
    break;
  case CIRCLE:
    circf(0.0, 0.0, 0.5);
    break;
  case RECT:
    rect(-0.5, -0.5, 0.5, 0.5);
    break;
  case RECTF:
    rectf(-0.5, -0.5, 0.5, 0.5);
    break;
  case QUIT:
    greset();
    gexit();
    exit(0);
  default:
    break;
  }
}
popup(names)
popupentry names[];
{
  register short i, menucount;
  short menutop, menubottom, menuleft, menuright;
  short lasthighlight = -1, highlight;
  short dummy, x, y;
  menucount = 0;
  qread(&x);
  qread(&y);
  pushmatrix();
  drawmode(OVERDRAW);
  ortho2(-0.5, 1279.5, -0.5, 1023.5);
  while (names[menucount].type)
    menucount++;
  menutop = y + menucount*8;
  menubottom = y - menucount*8;
  if (menutop > YMAXSCREEN) {
    menutop = YMAXSCREEN;
    menubottom = menutop - menucount*16;
  }
```

```
if (menubottom < 0) {
  menubottom = 0;
  menutop = menubottom + menucount*16;
}
menuleft = x - 100;
menuright = x + 100;
if (menuleft < 0) {
  menuleft = 0;
  menuright = menuleft + 200;
}
if (menuright > XMAXSCREEN) {
  menuright = XMAXSCREEN;
  menuleft = menuright - 200;
}
color(0);
clear();
color(1);                /* menu background */
rectfi(menuleft, menubottom, menuright, menutop);
color(2);                /* menu text */
move2i(menuleft, menubottom);
draw2i(menuleft, menutop);
draw2i(menuright, menutop);
draw2i(menuright, menubottom);
for (i = 0; i < menucount; i++) {
  move2i(menuleft, menutop - (i+1)*16);
  draw2i(menuright, menutop - (i+1)*16);
  cmov2i(menuleft + 10, menutop - 14 - i*16);
  charstr(names[i].text);
}
while (!qtest()) {
  x = getvaluator(MOUSEX);
  y = getvaluator(MOUSEY);
  if (menuleft < x && x < menuright &&
      menubottom < y && y < menutop)
  {
    highlight = (menutop - y)/16;
    if (lasthighlight != -1 && lasthighlight != highlight) {
      color(1);
      rectfi(menuleft+1,
          menutop - lasthighlight*16 - 15,
          menuright-1, menutop - lasthighlight*16 - 1);
      color(2);
      cmov2i(menuleft + 10,
          menutop - 14 - lasthighlight*16);
      charstr(names[lasthighlight].text);
    }
    if (lasthighlight != highlight) {
      color(2);
      rectfi(menuleft+1, menutop - highlight*16 - 15,
          menuright-1, menutop - highlight*16 - 1);
      color(1);
      cmov2i(menuleft + 10,
          menutop - 14 - highlight*16);
      charstr(names[highlight].text);
    }
    lasthighlight = highlight;
  }
  else /* the cursor is outside the menu */
  {
    if (lasthighlight != -1)
    {
      color(1);
      rectfi(menuleft+1,
          menutop - lasthighlight*16 - 15,
```

```
            menuright-1, menutop - lasthighlight*16 - 1);
        color(2);
        cmov2i(menuleft + 10,
            menutop - 14 - lasthighlight*16);
        charstr(names[lasthighlight].text);
        lasthighlight = -1;
      }
    }
  }
  qread(&dummy);
  qread(&x);
  qread(&y);
  color(0);
  rectfi(menuleft, menubottom, menuright, menutop);
  if (menuleft<x && x<menuright && menubottom<y && y<menutop)
    x = (menutop - y)/16;
  else
    x = 0;
  drawmode(NORMALDRAW);
  popmatrix();
  return names[x].type;
}
```

Related Information

prompt.c Example C Language Program

```
/*
prompt.c:

This program demonstrates a standard GL prompt and a user-defined
prompt. If you choose the user-defined prompt, mouse events are
ignored until you press the Enter key.

Peter Broadwell & dave ratcliffe 1989
*/

#include <stdio.h>
#include <gl/gl.h>
#include <gl/device.h>

#define PROMPT        1
#define EXIT          2

long menu;                /* The user-defined prompt's identifier */
char aString[40];

main()
{
  Device dev;
  short val;
  long menuval;

  init();                 /* process events forever */
  while(TRUE) {
    dev=qread(&val);
    switch(dev) {
    case ESCKEY:
      exit();
      break;
    case REDRAW:
      reshapeviewport();
      color(BLUE);
      clear();
      break;
    case RIGHTMOUSE:
      if(val) {
        menuval = dopup(menu);
        switch (menuval) {
```

```
        case PROMPT:
                          /* prompt to get file name */
          getUserString("File: ", aString,
              sizeof(aString));
          printf("The user entered \"%s\"\n",
              aString);
          break;
        case EXIT:
          exit();
          break;
        default:
          break;
        }
      }
      break;
    default:
      break;
    }
  }
}
init()                    /* do all the basic graphics setup */
{
  long sx, sy;
  ginit();                /* Open a full size window */
  overlay(2);
  drawmode(OVERDRAW);
  mapcolor(BLACK,0,0,0);
  mapcolor(RED,255,0,0);
  drawmode(NORMALDRAW);
  gconfig();
  qdevice(ESCKEY);
  qdevice(RIGHTMOUSE);
  qenter(REDRAW, 1);
  menu = defpup("GL-style prompt %t|My Prompt|Exit");
}
/*
Clear prompt, move to start of prompt box, and output requested
prompt
*/
getUserString(prompt,userStr,maxlen)  /* get name of file */
char *prompt, *userStr;
int maxlen;
{
                                      /* lower left corner of prompt box */
  #define FILEX 5
  #define FILEY 15
  #define FILEYHI (30+FILEY)          /* 30 pixels hi */
  #define TEXTX (FILEX+5)
  #define TEXTY (FILEY+10)
  #define clearprompt(aprmpt)                 \
    color(RED); clear(); color(BLACK); linewidth(2);\
    recti(FILEX+2, FILEY+2, wxsize-8, FILEYHI-1);\
    linewidth(1); cmov2i(TEXTX, TEXTY); charstr(aprmpt);
  int cur_str_len;
  short c;
  Device dev;
  long maxwidth, maxxval;             /* max length of window's width
                                         in pixels */
  char *str;
  char *prmpt = prompt, keyBoardWasQueued;
  long oldmode, xorig, yorig, wxsize, wysize;
  Screencoord mask1, mask2, mask3, mask4; /* Save old state to restore latter */
```

```
  pushmatrix();
  oldmode = getdrawmode();
  getscrmask(&mask1, &mask2, &mask3, &mask4);
  keyBoardWasQueued = isqueued(KEYBD);
  drawmode(OVERDRAW);                       /* Enable overlay */
                                            /* Set viewport to fill window */
  getorigin(&xorig,&yorig);
  getsize(&wxsize,&wysize);
  ortho2(-0.5,(float)wxsize, -0.5, (float)wysize);
  maxxval = wxsize + xorig;
  userStr[0] = '\0';
  maxwidth = (wxsize-11) - (FILEX + strwidth(prompt));
  scrmask(FILEX, (Screencoord)(wxsize-6), FILEY, FILEYHI);
  cur_str_len = strlen(userStr);
  clearprompt(prmpt);                       /* Display my prompt */
  qdevice(KEYBD);
                                            /* read till eof ('\n' or '\r') */
  while(dev = qread(&c)) {
    if(dev != KEYBD)
      continue;         /* don't care */
    switch(c) {
    case '\027':        /* ^W sets cursor back to start */
      cur_str_len = 0;
      clearprompt(prmpt);
      break;
    case '\n':
    case '\r':
      goto done;
    case '\b':
      if(cur_str_len) {
        userStr[--cur_str_len] = '\0';
        clearprompt(prmpt);
        /* display rightmost portion */
        for(str=userStr; *str && strwidth(str) >
            maxwidth; str++);
        charstr(str);
      }
      break;
    default:
      if(cur_str_len < (maxlen -1)) {
        str = &userStr[cur_str_len];
        userStr[cur_str_len++] = c;
        userStr[cur_str_len] = '\0';
        charstr(str);
      }
      else {
        ringbell();
      }
      break;
    }
  }
done:
  if(!keyBoardWasQueued) unqdevice(KEYBD);
  scrmask(mask1, mask2, mask3, mask4);      /* restore old */
  drawmode(OVERDRAW);
  color(0);
  clear();
  drawmode(oldmode);
  popmatrix();
  userStr[maxlen] = '\0';
}
```

Related Information

The **drawmode** subroutine, **getdrawmode** subroutine, **getscrmask** subroutine, **ginit** subroutine, **isqueued** subroutine, **linewidth** subroutine, **scrmask** subroutine, **strwidth** subroutine, **unqdevice** subroutine .

run_all Example C Language Program

```
set +x
typeset temp
for dude in *
do
   typeset temp2=${temp%.c*}
   echo "Executing: $temp2"
   $temp2
done
```

scrn_rotate.c Example C Language Program

```
/*
screenrotate.c
This program illustrates a technique for rotating an object about
a fixed set of axes (screen axes x, y, and z).
 Use the numeric
keypad to rotate the image.
It also demonstrates a technique for doing backface elimination
depending upon the visual relationship between the eye
point and a six-sided cube.
NOTE: If compiled with the "define" flag as "-DBACKFACE" the
graphics library function backface() will replace the code
ensuing from the function norm_dot and beyond.
*/

#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>

Coord ident [4][4] =     {
   1.0, 0.0, 0.0, 0.0,    /* identity matrix */
   0.0, 1.0, 0.0, 0.0,
   0.0, 0.0, 1.0, 0.0,
   0.0, 0.0, 0.0, 1.0};

static Coord cm [4][4] = {
   1.0, 0.0, 0.0, 0.0,    /* cumulative matrix */
   0.0, 1.0, 0.0, 0.0,
   0.0, 0.0, 1.0, 0.0,
   0.0, 0.0, 0.0, 1.0};

/*  Define the sides of the cube in world coordinates. */
static Coord pfrnt[4][3] = {
   {    0.0,    0.0,    0.0},
   {  100.0,    0.0,    0.0},
   {  100.0,  100.0,    0.0},
   {    0.0,  100.0,    0.0}
};

static Coord pback[4][3] = {
   {    0.0,    0.0, -100.0},
   {    0.0,  100.0, -100.0},
   {  100.0,  100.0, -100.0},
   {  100.0,    0.0, -100.0}
};
```

```
static Coord ptop[4][3] =  {
   {    0.0,  100.0,    0.0},
   {  100.0,  100.0,    0.0},
   {  100.0,  100.0, -100.0},
   {    0.0,  100.0, -100.0}
};
static Coord pbot[4][3] =  {
   {    0.0,    0.0,    0.0},
   {    0.0,    0.0, -100.0},
   {  100.0,    0.0, -100.0},
   {  100.0,    0.0,    0.0}
};
static Coord prsid[4][3] = {
   {  100.0,    0.0,    0.0},
   {  100.0,    0.0, -100.0},
   {  100.0,  100.0, -100.0},
   {  100.0,  100.0,    0.0}
};
static Coord plsid[4][3] = {
   {    0.0,    0.0,    0.0},
   {    0.0,  100.0,    0.0},
   {    0.0,  100.0, -100.0},
   {    0.0,    0.0, -100.0}
};
Coord x, y, z;
Angle rx, ry, rz;
float norm_dot();
main()
{
   int i, j;
   long dev;
   short data;
   /*  initialize and set the display to double buffer mode */
   prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
       YMAXSCREEN*3/4);
   #ifdef BACKFACE
   winopen("screeen rotation (backface)");
   #else
   winopen("screeen rotation");
   #endif
   doublebuffer();
   gconfig();
   writemask((1<<getplanes())-1);
   qdevice(PAD1);   /* translate (in Z) toward the eyepoint */
   qdevice(PAD2);   /* rotate about the X axis in a negative direction */
   qdevice(PAD3);   /* translate (in Z) away from the eyepoint */
   qdevice(PAD4);   /* rotate about the Y axis in a positive direction */
   qdevice(PAD5);   /* reset rotations and translations to default */
   qdevice(PAD6);   /* rotate about the Y axis in a negative direction */
   qdevice(PAD7);   /* rotate about the Z axis in a positive direction */
   qdevice(PAD8);   /* rotate about the X axis in a positive direction */
   qdevice(PAD9);   /* rotate about the Z axis in a negative direction */
   qdevice(FKEY);   /* translate (in Z) toward the eyepoint */
   qdevice(BKEY);   /* translate (in Z) away from the eyepoint */
   qdevice(ESCKEY); /* exit program */
   #ifdef BACKFACE
   /* compile with "-DBACKFACE" if you desire to use GL's */
   backface(TRUE);
   #endif
   perspective(470,1.25,1.0,10000.0);
   /*  initialize the modeling transformation values */
   rx =     0;
```

```
ry =     0;
rz =     0;
x = -50.0;
y = -50.0;
z = -400.0;

/*  set up the loop for reading input and drawing the cube */
while(TRUE) {
   color(BLACK);
   clear();
   viewcube();

   /*  read the input for moving the box around the eye point */
   while (qtest()) {
      dev = qread(&data);
      switch (dev) {
      case REDRAW:                /* redraw event */
         reshapeviewport();
         viewcube('t');
         break;
         case(ESCKEY):             /* exit program */
         gexit();
         exit(0);
         break;
         case(FKEY):               /* translate toward the eyepoint */
         case(PAD1):
         while(getbutton(FKEY) || getbutton(PAD1)) {
            z = z + 20.0;
            viewcube('t');
         }

         break;
         case(BKEY):               /* translate away from the eyepoint */
         case(PAD3):
         while(getbutton(BKEY) || getbutton(PAD3)) {
            z = z - 20.0;
            viewcube('t');
         }

         break;

         case(PAD2):
         while(getbutton(PAD2)) {  /* rotate about the X axis */
            rx = rx - 100;
            viewcube('x');
         }

         updatemat('x');           /* incorporate this rotation into */
         rx = 0;                   /* cumulative rotation matrix */
         break;

         case(PAD4):
         while(getbutton(PAD4)) {  /* rotate about the Y axis */
            ry = ry + 100;
            viewcube('y');
         }

         updatemat('y');           /* incorporate this rotation into */
         ry = 0;                   /* cumulative rotation matrix */
         break;

         case(PAD6):
         while(getbutton(PAD6)) {  /* rotate about the Y axis */
            ry = ry - 100;
            viewcube('y');
         }

         updatemat('y');           /* incorporate this rotation into */
         ry = 0;                   /* cumulative rotation matrix */
         break;
```

```
            case(PAD7):
            while(getbutton(PAD7)) {  /* rotate about the Z axis */
               rz = rz + 100;
               viewcube('z');
            }
            updatemat('z');           /* incorporate this rotation into */
            rz = 0;                   /* cumulative rotation matrix */
            break;
            case(PAD8):
            while(getbutton(PAD8)) {  /* rotate about the X axis */
               rx = rx + 100;
               viewcube('x');
            }
            updatemat('x');           /* incorporate this rotation into */
            rx = 0;                   /* cumulative rotation matrix */
            break;
            case(PAD9):
            while(getbutton(PAD9)) {  /* rotate about the Z axis */
               rz = rz - 100;
               viewcube('z');
            }
            updatemat('z');           /* incorporate this rotation into */
            rz = 0;                   /* cumulative rotation matrix */
            break;
            case(PAD5):               /* reset rotations & translations   */
            x  =  -50.0;
            y  =  -50.0;
            z  = -400.0;
            rx = 0;
            ry = 0;
            rz = 0;
            for(i=0;i<4;i++) {
               for(j=0;j<4;j++)
                  cm[i][j] = ident[i][j];
            }
            viewcube('t');
            break;
         } /* end switch */
         qreset();
      } /* end while(qtest()) */
   } /* end while(1) */
} /* end of main */
viewcube(axis)
char axis;
{
   /*  Transform the cube in world space and (if BACKFACE not
      defined, in software, ) check each face for back face
      elimination
  */
   color(BLACK);
   clear();
   pushmatrix();
   translate(x,y,z);
   pushmatrix();
   translate(50.0,50.0,-50.0);   /* apply rotation about a single axis */
   switch(axis) {
      case('x'):
      rotate(rx,'x');
      break;
      case('y'):
      rotate(ry,'y');
      break;
      case('z'):
      rotate(rz,'z');
```

```
      break;
   default:
      break;
   } /* apply all prior rotations */
   multmatrix(cm);
   translate(-50.0,-50.0,50.0);
   #ifdef BACKFACE
   /* compile with "-DBACKFACE" if you desire to use GL's version */
   color(1);
   polf(4,pfrnt);
   color(2);
   polf(4,pback);
   color(3);
   polf(4,ptop);
   color(4);
   polf(4,pbot);
   color(5);
   polf(4,prsid);
   color(6);
   polf(4,plsid);
   #else
   color(1);
   if(norm_dot(pfrnt) >= 0.0) polf(4,pfrnt);
   color(2);
   if(norm_dot(pback) >= 0.0) polf(4,pback);
   color(3);
   if(norm_dot(ptop) >= 0.0) polf(4,ptop);
   color(4);
   if(norm_dot(pbot) >= 0.0) polf(4,pbot);
   color(5);
   if(norm_dot(prsid) >= 0.0) polf(4,prsid);
   color(6);
   if(norm_dot(plsid) >= 0.0) polf(4,plsid);
   #endif
   popmatrix();
   popmatrix();
   swapbuffers();
}
/*
 *  Function to postmultiply cumulative rotations
 *     by rotation about a single axis
 */
updatemat(axis)
char axis;
{
   pushmatrix();
   loadmatrix(ident);
   switch (axis) {
   case ('x'):
      rotate(rx,'x');
      break;
   case ('y'):
      rotate(ry,'y');
      break;
   case ('z'):
      rotate(rz,'z');
      break;
   default:
      break;
   }
   multmatrix(cm);
   getmatrix(cm);
   popmatrix();
}
```

```
/*
      The function norm_dot takes as input an array of points in
      homogeneous coordinates which make up a surface or plane.  The
      unit normal of the surface and the eyepoint to surface unit
      vector are computed and the dot product is calculated.  This
      function returns the dot product floating point value and the
      transformed points for the surface.
*/
float norm_dot(passpoly)
Coord passpoly[][3];
{
   int i;
   float a[3],b[3],c[3],d,abs;
   Coord postrans [4][3];
  /* Apply the current transformation to the surface points. */
   transform(4,passpoly,postrans);
  /* Determine two vectors which lie in the specified plane.
   * The first three points are taken from the surface array.
   * These points are ordered by the right-hand rule in the
   * right-hand coordinate system:  i.e. points ordered counter-
   * clockwise when on the positive side of the plane or surface
   * are visible, not backfacing, surfaces.
   *    a[] gets the xyz coords of row 2
   *    b[] gets the xyz coords of row 0.
    */
   /* Determine two vectors.  Note that this routine assumes they
   * are not in-line */
   for(i = 0; i < 3; i++)
      a[i] = postrans[2][i] - postrans[1][i];
   for(i = 0; i < 3; i++)
      b[i] = postrans[0][i] - postrans[1][i];
  /* Find the cross product of the two vectors */
   c[0] = a[1] * b[2] - a[2] * b[1];
   c[1] = a[2] * b[0] - a[0] * b[2];
   c[2] = a[0] * b[1] - a[1] * b[0];
  /* Calculate the unit normal vector for the plane or poly
   * using the square root of the sum of the squares of x, y,
   * and z to determine length of vector, then dividing each
    * axis by that length (x/l, y/l, z/l).
   */
   abs = 0.0;
   for (i = 0; i < 3; i++)
      abs += (c[i]*c[i]);
   d = sqrt(abs);
   if (fabs(d) > 0.000001)    {
      for (i = 0; i < 3; i++)
         a[i] = c[i]/d;
       /* Calculate the unit vector pointing from the eyepoint to
          * the normal of the plane or poly */
      abs = 0.0;
      for (i = 0; i < 3; i++)
         c[i] = postrans[1][i];
      for (i = 0; i < 3; i++)
         abs = abs + (c[i]*c[i]);
      d = sqrt(abs);
      if (fabs(d) > 0.000001)    {
         for (i = 0; i < 3; i++)
            b[i] = c[i]/d;
       /* Return the dot product between the eye vector and the
            * plane normal */
         for (i = 0, d=0.0; i < 3; i++)
            d = d + a[i]*b[i];
      }
```

```
      else
         printf("\n Magnitude of surface vector is zero!");
   }
   else
      printf("\n Magnitude of eye vector is zero!");
   return(d);
}
/* The function transform() simply multiplies each vertex point
 * with the current transformtion matrix without any clipping,
 * scaling, etc. to derive transformed world coordinate values.
 */
transform(n,passpoly,postrans)
long n;
Coord passpoly[][3], postrans[][3];
{
   Matrix ctm;
   pushmatrix();
   getmatrix(ctm);
   postrans[0][0] = passpoly[0][0]*ctm[0][0] +
       passpoly[0][1]*ctm[1][0] +
       passpoly[0][2]*ctm[2][0] + ctm[3][0];
   postrans[0][1] = passpoly[0][0]*ctm[0][1] +
       passpoly[0][1]*ctm[1][1] +
       passpoly[0][2]*ctm[2][1] + ctm[3][1];
   postrans[0][2] = passpoly[0][0]*ctm[0][2] +
       passpoly[0][1]*ctm[1][2] +
       passpoly[0][2]*ctm[2][2] + ctm[3][2];
   postrans[1][0] = passpoly[1][0]*ctm[0][0] +
       passpoly[1][1]*ctm[1][0] +
       passpoly[1][2]*ctm[2][0] + ctm[3][0];
   postrans[1][1] = passpoly[1][0]*ctm[0][1] +
       passpoly[1][1]*ctm[1][1] +
       passpoly[1][2]*ctm[2][1] + ctm[3][1];
   postrans[1][2] = passpoly[1][0]*ctm[0][2] +
       passpoly[1][1]*ctm[1][2] +
       passpoly[1][2]*ctm[2][2] + ctm[3][2];
   postrans[2][0] = passpoly[2][0]*ctm[0][0] +
       passpoly[2][1]*ctm[1][0] +
       passpoly[2][2]*ctm[2][0] + ctm[3][0];
   postrans[2][1] = passpoly[2][0]*ctm[0][1] +
       passpoly[2][1]*ctm[1][1] +
       passpoly[2][2]*ctm[2][1] + ctm[3][1];
   postrans[2][2] = passpoly[2][0]*ctm[0][2] +
       passpoly[2][1]*ctm[1][2] +
       passpoly[2][2]*ctm[2][2] + ctm[3][2];
   postrans[3][0] = passpoly[3][0]*ctm[0][0] +
       passpoly[3][1]*ctm[1][0] +
       passpoly[3][2]*ctm[2][0] + ctm[3][0];
   postrans[3][1] = passpoly[3][0]*ctm[0][1] +
       passpoly[3][1]*ctm[1][1] +
       passpoly[3][2]*ctm[2][1] + ctm[3][1];
   postrans[3][2] = passpoly[3][0]*ctm[0][2] +
       passpoly[3][1]*ctm[1][2] +
       passpoly[3][2]*ctm[2][2] + ctm[3][2];
   popmatrix();
}
/*
    Changes:
      - changed the name of the window acccording to the
        #ifdef BACKFACE
*/
```

Related Information

The **getbutton** subroutine, **getmatrix** subroutine, **getplanes** subroutine, **gexit** subroutine, **qdevice** subroutine, **qenter** subroutine, **qread** subroutine, **qreset** subroutine, **qtest** subroutine, **reshapeviewport** subroutine .

---

select1.c Example C Language Program

```
/*
select1.c:

Select demo program.  the "ship" is the blue rectangle.  The
"planet" is the red circle. move the ship so it intersects the
planet and the ship will crash.
*/

#include <gl/gl.h>
#include <gl/device.h>

#define PLANET 1

main()
{
  short type, val;
  register short buffer[50], cnt, i;
  float shipx, shipy, shipz;

  for (i = 0; i < 50; i++)
    buffer[i] = 0;
  initialize();
  while (TRUE) {
    type = qread(&val);
    if (val==0)
      continue;
    switch (type) {
    case REDRAW:
      reshapeviewport();
      drawplanet();
      break;
    case ESCKEY:
      gexit();
      exit();
    case LEFTMOUSE:
      /* set ship location to cursor location */
      shipz=0;
      shipx=getvaluator(MOUSEX);
      shipy=getvaluator(MOUSEY);
      /* draw the ship */
      color(BLUE);
      rect(shipx, shipy, shipx+20, shipy+10);
      /* specify the selecting region to be a box
         surrounding the ship */
      pushmatrix();
      ortho(shipx, shipx+.05, shipy, shipy+.05,
          shipz-0.5, shipz+.05);
      /* clear the name stack */
      initnames();
      gselect(buffer, 50);  /* enter selecting mode */
      /* put "1" on the name stack to be saved if
         PLANET draws into the selecting region */
      loadname(1);
      pushname(2);                /* draw the planet */
      callobj(PLANET);            /* exit selecting mode */
      cnt = endselect(buffer);
      popmatrix();                /* check to see if PLANET was selected */
      printf("cnt = %d\n",cnt);
      for (i = 0;i<4;i++)
        printf("buffer[%d] = %d\n",i,buffer[i]);
      if (buffer[1]==1) {
```

```
        printf("CRASH\n");
        gexit();
        exit();
      }
      break;
    default:
      break;
    }
  }
}
initialize()
{
  int gid;
  float xmax,ymax;
  prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4, YMAXSCREEN/4,
      YMAXSCREEN*3/4);
  keepaspect(1,1);
  gid = winopen("select1");
  qdevice(ESCKEY);
  qdevice(REDRAW);
  qdevice(LEFTMOUSE);
  qenter(REDRAW,gid);
  xmax = .5 + (float) XMAXSCREEN;
  ymax = .5 + (float) YMAXSCREEN;
  ortho(xmax/4.0, xmax*3.0/4.0, ymax/4.0, ymax*3.0/4.0, 0.0,
      -xmax/2.0);
  createplanet(PLANET);
}
drawplanet()
{
  color(BLACK);
  clear();
  color(RED);
  /* create the planet object */
  callobj(PLANET);
}
createplanet(x)
{
  makeobj(x);
  circfi(600,600,20);
  closeobj();
}
```

Related Information

The **endselect** subroutine, **getvaluator** subroutine, **gselect** subroutine .

---

setshade.c Example C Language Program

```
/*
Example setshade.c:
Moves a smooth-shaded polygon in and out of the graphics port.
Press LEFTMOUSE to exit.
*/
#include <gl/gl.h>
#include <gl/device.h>
main()
{
  unsigned short i;
  Coord x;
```

```
  prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,YMAXSCREEN*3/4);
  winopen("setshade/clip test");
  doublebuffer();
  gconfig();
  get_cmap();
  for (i=0; i<128; i++)
    mapcolor(128+i, 2*i, 0, 2*i);
  makeobj(1);
  color(128+127);
  pmv2i(100,-100);
  color(128+0);
  pdr2i(100,100);
  color(128+127);
  pdr2i(-100,100);
  color(128+0);
  pdr2i(-100,-100);
  pclos();
  closeobj();
  while (!getbutton(LEFTMOUSE)) {
    for (x = -150.0; x < 150.0; x++) {
      color(CYAN);
      clear();
      pushmatrix();
      translate(x,150.0,0.0);
      rotate(300,'z');
      callobj(1);
      popmatrix();
      swapbuffers();
      if (getbutton(LEFTMOUSE))
        break;
    }
    for (x=150.0; x>-150.0; x--) {
      color(CYAN);
      clear();
      pushmatrix();
      translate(x,150.0,0.0);
      rotate(300,'z');
      callobj(1);
      popmatrix();
      swapbuffers();
      if (getbutton(LEFTMOUSE))
        break;
    }
  }
  restore_cmap();
  gexit();
}
#define lo_end 128
#define hi_end 255
static Int16 CarrayR[hi_end+1], CarrayG[hi_end+1], CarrayB[hi_end+1];
get_cmap()
{
  getmcolors ((Int16)lo_end,(Int16)hi_end, CarrayR, CarrayG, CarrayB);
}
restore_cmap()
{
  mapcolors ((Int16)lo_end,(Int16)hi_end, CarrayR, CarrayG, CarrayB);
}
/*
  Changes:
  - This needed the pclos() call after defining the polygon
  - The mapcolor were changed to mapcolors
  - The getmcolor were changed to getmcolors
```

```
  - The mapcolors needs casting to an integer
  - loop constructs removed from restore_cmap and get_cmap
  - the index variable was removed
*/
```

Related Information

---

sunflower.c Example C Language Program

```
/*
sunflower.c

Make a sunflower-like pattern out of circles.
    Usage: sunflower <<nseeds> <<seedsize> <<growth>
Try "sunflower 400.05 1.1"

Paul Haeberli - 1984
*/

#include <gl/gl.h>
#include <gl/device.h>
#include <stdio.h>
#include <math.h>

int seeds = 0;

main(argc,argv)
int argc;
char **argv;
{
   int nseeds;
   float seedsize, grow;
   short val;

   if (argc<4) {
      fprintf(stderr,
          "Usage: sunflower <nseeds> <seedsize> <growth>\n");
      exit(1);
   }

   nseeds = atoi(argv[1]);
   seedsize = atof(argv[2]);
   grow = atof(argv[3]);
   prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
       YMAXSCREEN*3/4);
   winopen("sunflower");
        singlebuffer();
        zbuffer (FALSE);
        cmode();
    gconfig ();
   makeframe();
   sunflower(nseeds,seedsize,grow);
   while (1) {
      if (qread(&val) == REDRAW)  {
         makeframe();
         sunflower(nseeds,seedsize,grow);
      }
   }
}

sunflower(nseeds,seedsize,grow)
int nseeds;
float seedsize, grow;
{
   float rad = 20.0;
   int parity = 0;

   scale(10.0,10.0,0.0);
   pushmatrix();
   while (rad < 100.0) {
```

```
      rotate(1800/nseeds,'z');
      scale(grow,grow,0.0);
      makering(nseeds,seedsize);
      rad = rad * grow;
   }
   popmatrix();
}
makering(nseeds,seedsize)
int nseeds;
float seedsize;
{
   int i;
   for (i=0; i<nseeds; i++) {
      pushmatrix();
      rotate((i*3600)/nseeds,'z');
      drawseed(seedsize);
      popmatrix();
   }
}
drawseed(seedsize)
float seedsize;
{
   seeds++;
   circ(1.0,0.0,seedsize);
}
makeframe()
{
   int xsize, ysize;
   float aspect;
   reshapeviewport();
   getsize(&xsize,&ysize);
   color(7);
   clear();
   color(0);
   aspect = xsize/(float)ysize;
   ortho2(-50.0,50.0,-50.0/aspect,50.0/aspect);
}
```

Related Information

---

text.c Example C Language Program

```
/*
text.c:
A text drawing sample program using the charstr() subro utine.
*/
#include <gl/gl.h>
#include <gl/device.h>
main()
{
   Device dev;
   short val;
   initialize();
   while (TRUE) {
      if (qtest()) {
         dev = qread(&val);
         if (dev == ESCKEY) {
            gexit();
            exit();
         }
         else if (dev == REDRAW) {
```

```
            reshapeviewport();
            drawtext();
         }
      }
   }
}
initialize()
{
   int gid;
   prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4, YMAXSCREEN/4,
       YMAXSCREEN*3/4);
   gid = winopen("text");
   winconstraints();
   qdevice(ESCKEY);
   qdevice(REDRAW);
   qenter(REDRAW,gid);
}
drawtext()
{
   color(BLACK);
   clear();
   color(RED);
   cmov2i(300,380);
   charstr("The first line is drawn ");
   charstr("in two parts. ");
   cmov2i(300, 368);
   charstr("This line is 12 pixels lower. ");
}
```

Related Information

The **curson or cursoff** subroutine .

---

tpbig.c Example C Language Program

```
/*
tpbig.c:
Basic graphics program demonstrating arcs, polygons, character
strings, and use of a textport.
*/
#include <gl/gl.h>
#include <gl/device.h>
#include <stdio.h>
long cone[][2] = {100, 300, 150, 100, 200, 300};
char *singlechar;
main()
{
   int gid;
   short val;
   singlechar = malloc(2); /* Space for a character and a Null */
   memcpy(singlechar, "X", 2);
   prefposition(0,XMAXSCREEN,0,YMAXSCREEN);
   gid = winopen("tpbig");
   qdevice(ESCKEY);
   qdevice(REDRAW);
   qenter(REDRAW,gid);
   reshapeviewport();
   drawstuff();
   textport(50,300,750,900);
   tpon();
   while(TRUE) {
      switch(qread(&val)) {
```

```
      case ESCKEY:
         gexit();
         exit();
      case REDRAW:
         reshapeviewport();
         drawstuff();
      }
   }
}
drawstuff()
{
   register long i, j;             /* draw an ice-cream cone */
   color(WHITE);
   clear();
   color(YELLOW);
   polf2i(3, cone);                /* draw the ice-cream cone */
   color(GREEN);                   /* first scoop is mint */
   arcfi(150, 300, 50, 0, 1800);   /* only half of it shows */
   color(RED);                     /* second scoop is cherry */
   circf(150.0, 400.0, 50.0);
   color(BLACK);
   poly2i(3, cone);                /* outline the cone in black */
   /* Next, draw a few filled and unfilled arcs in the upper
   * left corner of the screen.
    */
  arcf(100.0, 650.0, 40.0, 450, 2700);
   arci(100, 500, 40, 450, 2700);
   arcfi(250, 650, 80, 2700, 450);
   arc(250.0, 500.0, 80.0, 2700, 450);
  /* Now, put up a series of filled and unfilled rectangles with
   * the names of their colors printed inside of them across the
   * rest of the top of the screen.
    */
   color(GREEN);
   recti(400, 600, 550, 700);
   cmov2i(420, 640);
   charstr("Green");
   color(RED);
   rectfi(600, 600, 800, 650);
   color(BLACK);
   cmov2(690.0, 620.0);
   charstr("Red");
   color(BLUE);
   rect(810.0, 700.0, 1000.0, 20.0);
   cmov2i(900, 300);
   charstr("Blue");
   /* Now draw some text with a ruler on top to measure it by. */
   /* First the ruler: */
  color(BLACK);
   move2i(300, 400);
   draw2i(650, 400);
   for (i = 300; i <= 650; i += 10) {
      move2i(i, 400);
      draw2i(i, 410);
   }
     /* Then some text: */
  cmov2i(300, 380);
   charstr("The first line is drawn ");
   charstr("in two parts.");
   cmov2i(300, 368);
   charstr("This line is only 12 pixels lower.");
   cmov2i(300, 354);
   charstr("Now move down 14 pixels ...");
   cmov2i(300, 338);
   charstr("And now down 16 ...");
```

```
    cmov2i(300, 320);
    charstr("Now 18 ...");
    cmov2i(300, 300);
    charstr("And finally, 20 pixels.");

    /* Finally, show off the entire font.  The cmov2i() before
     * each character is necessary in case that character is not
     * defined.
      */
    for (i = 0; i < 4; i++)
       for (j = 0; j < 32; j++) {
          cmov2i(300 + 9*j, 200 - 18*i);
          *singlechar = (char)(32*i + j);
          charstr(singlechar);
       }
    for (i = 0; i < 4; i++) {
       cmov2i(300, 100 - 18*i);
       for (j = 0; j < 32; j++) {
          *singlechar = (char)(32*i + j);
          charstr(singlechar);
       }
    }
}

/*
  Changes:
         - Changed memcopy(singlechar, "X", 2); to memcpy ...
        - Added the & in this statement:
          switch(qread(&val)) {
       - Removed the textinit(); call for case ESCKEY
*/
```

Related Information

The **arc** subroutine, **arcf** subroutine, **circf** subroutine, **polf** subroutine, **poly** subroutine, **rect** subroutine, **rectf** subroutine, **tpon** subroutine .

---

vlsi.c Example C Language Program

```
/*
vlsi.c

A simple vlsi graphical editor.  RIGHTMOUSE clears the screen.
LEFTMOUSE picks the current color from one of 4 in the bottom
left-hand corner, and draws the rectangles.  To draw, hold down
LEFTMOUSE on the point where you want one of the four
corners of the rectangle to be, and then move the mouse to the
opposite corner of the rectangle you want to specify, BEFORE you
let go of the LEFTMOUSE.
*/

#include <gl/gl.h>
#include <gl/device.h>

main()
{
  register i;
  Device dummy, xend, yend, xstart, ystart, type;
  short wm;

  prefposition(0, XMAXSCREEN-50, 0, YMAXSCREEN-50);
  winopen("vlsi");
  save_cmap();
  mapcolor(0, 255, 255, 255);    /* WHITE */
  mapcolor(1, 0, 0, 255);        /* BLUE */
  mapcolor(2, 0, 255, 0);        /* RED */
  mapcolor(3, 0, 150, 255);      /* PURPLE */
  mapcolor(4, 255, 0, 0);        /* GREEN */
  mapcolor(5, 150, 0, 255);      /* LIGHT BLUE */
```

```
  mapcolor(6, 255, 255, 0);      /* YELLOW */
  mapcolor(7, 150, 100, 0);      /* BROWN */
  for (i = 8; i < 24; i++)
    mapcolor(i, 0, 0, 0);        /* BLACK */
  for (i = 24; i < 32; i++)
    mapcolor(i, 255, 255, 255);  /* WHITE */
  qdevice(LEFTMOUSE);
  tie(LEFTMOUSE, MOUSEX, MOUSEY);
  qdevice(MIDDLEMOUSE);
  tie(MIDDLEMOUSE, MOUSEX, MOUSEY);
  qdevice(RIGHTMOUSE);
  qdevice(KEYBD);
  setcursor(0, 16, 16);
  restart();
  while (TRUE)
    switch (type = qread(&dummy)) {
    case KEYBD:
      restore_cmap();
      greset();
      gexit();
      exit(0);
    case RIGHTMOUSE:
      qread(&dummy);
      restart();
      break;
    case MIDDLEMOUSE:
    case LEFTMOUSE:
      qread(&xstart);
      qread(&ystart);
      if (xstart < 60) {
        if (10 <= xstart && xstart <= 50) {
          if (10 <= ystart && ystart <= 50)
            wm = 1;
          else if (60 <= ystart && ystart <= 100)
            wm = 2;
          else if (110 <= ystart && ystart <= 150)
            wm = 4;
          else if (160 <= ystart && ystart <= 200)
            wm = 8;
          writemask(wm);
          qread(&dummy);
          qread(&dummy);
          qread(&dummy);
        }
      }
      else {
        qread(&dummy);
        qread(&xend);
        qread(&yend);
        if (xend > 60) {
          if (type == LEFTMOUSE)
            color(31);           /* draw */
          else
            color(0);            /* erase */
          rectfi(xstart, ystart, xend, yend);
        }
      }
    }
}
restart()
{
  writemask(0xfff);
  color(0);
  clear();
  color(1);
  rectfi(10, 10, 50, 50);
```

```
  color(2);
  rectfi(10, 60, 50, 100);
  color(4);
  rectfi(10, 110, 50, 150);
  color(8);
  rectfi(10, 160, 50, 200);
  move2i(60, 0);
  draw2i(60, 767);
  color(31);
  writemask(0);
}

/*This saves the colormap*/
#define lo_end 0
#define hi_end 255
short *CarrayR, *CarrayG, *CarrayB;

save_cmap()
{
  CarrayR = calloc (lo_end+hi_end,sizeof(short));
  CarrayG = calloc (lo_end+hi_end,sizeof(short));
  CarrayB = calloc (lo_end+hi_end,sizeof(short));

  getmcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}

/*This restores the colormap*/
restore_cmap()
{
  mapcolors ((Int16 const)lo_end,(Int16 const)hi_end,
     CarrayR, CarrayG, CarrayB);
}
/*
  Changes:
    -  Added the restoring of the colormap
*/
```

Related Information

The **greset** subroutine, **tie** subroutine, **writemask** subroutine .

---

worms.c Example C Language Program

```
/*
 @@@        @@@    @@@@@@@@@@     @@@@@@@@@@    @@@@@@@@@@@@
 @@@        @@@   @@@@@@@@@@@@    @@@@@@@@@@@   @@@@@@@@@@@@
 @@@        @@@  @@@@      @@@@   @@@@          @@@@ @@@  @@@@
 @@@   @@   @@@  @@@        @@@   @@@           @@@  @@@   @@@
 @@@  @@@@  @@@  @@@        @@@   @@@           @@@  @@@   @@@
 @@@@ @@@@ @@@@  @@@        @@@   @@@           @@@  @@@   @@@
  @@@@@@@@@@@@   @@@@      @@@@   @@@           @@@  @@@   @@@
   @@@@  @@@@     @@@@@@@@@@@@    @@@           @@@  @@@   @@@
    @@    @@       @@@@@@@@@@     @@@           @@@  @@@   @@@
        Eric P. Scott
                        Caltech High Energy Physics
                               October, 1980
*/

#include <stdio.h>
#include <gl/gl.h>
#include <gl/device.h>

#define INCREMENT    1.0
#define MAXCOLS      100
#define MAXROWS      75
#define SEG0         20
#define SEG1         21
#define TRAIL_OBJ    22
```

```
int Wrap;
short *ref[MAXROWS];

static int flavor[]={ 1, 2, 3, 4, 5, 6 };

static int segobj[]={ SEG1, SEG0, SEG0, SEG0, SEG0, SEG0 };

static short xinc[]= { 1,  1,  1,  0, -1, -1, -1,  0 },
             yinc[]= {-1,  0,  1,  1,  1,  0, -1, -1 };

static struct worm {
    int orientation, head;
    short *xpos, *ypos;
} worm[40];

static char *field;
static int length=16, number=3, trail=' ';
static struct options {
    int nopts;
    int opts[3];
} nrmal[8]={
    { 3, { 7, 0, 1 } } ,
    { 3, { 0, 1, 2 } } ,
    { 3, { 1, 2, 3 } } ,
    { 3, { 2, 3, 4 } } ,
    { 3, { 3, 4, 5 } } ,
    { 3, { 4, 5, 6 } } ,
    { 3, { 5, 6, 7 } } ,
    { 3, { 6, 7, 0 } } },
 upper[8]={
    { 1, { 1, 0, 0 } } ,
    { 2, { 1, 2, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 2, { 4, 5, 0 } } ,
    { 1, { 5, 0, 0 } } ,
    { 2, { 1, 5, 0 } } },
 left[8]={
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 2, { 2, 3, 0 } } ,
    { 1, { 3, 0, 0 } } ,
    { 2, { 3, 7, 0 } } ,
    { 1, { 7, 0, 0 } } ,
    { 2, { 7, 0, 0 } }},
 right[8]={
    { 1, { 7, 0, 0 } } ,
    { 2, { 3, 7, 0 } } ,
    { 1, { 3, 0, 0 } } ,
    { 2, { 3, 4, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 2, { 6, 7, 0 } }},
 lower[8]={
    { 0, { 0, 0, 0 } } ,
    { 2, { 0, 1, 0 } } ,
    { 1, { 1, 0, 0 } } ,
    { 2, { 1, 5, 0 } } ,
    { 1, { 5, 0, 0 } } ,
    { 2, { 5, 6, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } }},
 upleft[8]={
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
```

```
    { 0, { 0, 0, 0 } } ,
    { 1, { 3, 0, 0 } } ,
    { 2, { 1, 3, 0 } } ,
    { 1, { 1, 0, 0 } }},
 upright[8]={
    { 2, { 3, 5, 0 } } ,
    { 1, { 3, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 1, { 5, 0, 0 } }},
 lowleft[8]={
    { 3, { 7, 0, 1 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 1, { 1, 0, 0 } } ,
    { 2, { 1, 7, 0 } } ,
    { 1, { 7, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } }},
 lowright[8]={
    { 0, { 0, 0, 0 } } ,
    { 1, { 7, 0, 0 } } ,
    { 2, { 5, 7, 0 } } ,
    { 1, { 5, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } ,
    { 0, { 0, 0, 0 } } };

int m1, m2, m3;
int coffset;
int slowmode;
int bigblox;
int CO, LI;

main(argc,argv)
int argc;
char *argv[];
{
  float ranf();
  register int x, y;
  register int n;
  register struct worm *w;
  register struct options *op;
  register int h;
  register short *ip;
  int last, bottom;
  char *tcp;
  register char *term;
  char tcb[100];

  srand(getpid());
  CO = MAXCOLS;
  LI = MAXROWS;
  CO = 60;
  LI = 45;
  bottom = LI-1;
  last = CO-1;

  /* make a work area */
  keepaspect(400,300);
  prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
      YMAXSCREEN*3/4);
  winopen("worms");
  makeframe();
  makeobjects();
```

```
  qdevice(RIGHTMOUSE);
  qdevice(MIDDLEMOUSE);
  qdevice(LEFTMOUSE);
  for (x=1;x<argc;x++) {
    register char *p;
    p=argv[x];
    if (*p=='-') p++;
    switch (*p) {
    case 'f':
      field="WORM";
      break;
    case 'l':
      if (++x==argc) goto usage;
      if ((length=atoi(argv[x]))<2||length>1024) {
        fprintf(stderr,"%s: Invalid length\n",*argv);
        exit(1);
      }
      break;
    case 'n':
      if (++x==argc) goto usage;
      if ((number=atoi(argv[x]))<1||number>40) {
        fprintf(stderr,"%s: Invalid number of worms\n",*argv);
        exit(1);
      }
      break;
    case 't':
      trail='.';
      break;
    default:
usage:
      fprintf(stderr,
          "Usage: %s [-field] [-length #] [-number #] [-trail]\n",
           *argv);
      exit(1);
      break;
    }
  }
  ip=(short *)malloc(LI*CO*sizeof (short));
  for (n=0;n<LI;) {
    ref[n++]=ip;
    ip+=CO;
  }
  for (ip=ref[0],n=LI*CO;--n>=0;)
    *ip++=0;
  if (Wrap) ref[bottom][last]=1;
  for (n=number, w= &worm[0];--n>=0;w++) {
    w->orientation=w->head=0;
    if (!(ip=(short *)malloc(length*sizeof (short)))) {
      fprintf(stderr,"%s: out of memory\n",*argv);
      exit(1);
    }
    w->xpos=ip;
    for (x=length;--x>=0;) *ip++ = -1;
    if (!(ip=(short *)malloc(length*sizeof (short)))) {
      fprintf(stderr,"%s: out of memory\n",*argv);
      exit(1);
    }
    w->ypos=ip;
    for (y=length;--y>=0;) *ip++ = -1;
  }
  if (field) {
    register char *p;
    pushmatrix();
    p=field;
    for (y=bottom;--y>=0;) {
```

```
    pushmatrix();
    for (x=CO;--x>=0;)  {
      putfield();
      translate(INCREMENT,0.0,0.0);
    }
    popmatrix();
    translate(0.0,INCREMENT,0.0);
  }
  popmatrix();
}
for (;;) {
  checkmouse();
  for (n=0,w= &worm[0];n<number;n++,w++) {
    if ((x=w->xpos[h=w->head])<0) {
      x=w->xpos[h]=0;
      y=w->ypos[h]=bottom;
      pushmatrix();
      translate((float)x,(float)y,0.0);
      if(bigblox)
        scale(2.0,2.0,1.0);
      putsegment(flavor[n%6],segobj[n%6]);
      popmatrix();
      ref[y][x]++;
    }
    else y=w->ypos[h];
    if (++h==length) h=0;
    if (w->xpos[w->head=h]>=0) {
      register int x1, y1;
      x1=w->xpos[h];
      y1=w->ypos[h];
      if (--ref[y1][x1]==0) {
        pushmatrix();
        translate((float)x1,(float)y1,0.0);
        puttrail();
        popmatrix();
      }
    }
    op= &(x==0 ? (y==0 ? upleft : (y==bottom ? lowleft :
        left)) :
    (x==last ? (y==0 ? upright :
        (y==bottom ? lowright : right)) :
    (y==0 ? upper : (y==bottom ? lower :
        nrmal))))[w->orientation];
    switch (op->nopts) {
    case 0:
      fflush(stdout);
      abort();
      return;
    case 1:
      w->orientation=op->opts[0];
      break;
    default:
      w->orientation=op->opts[
                     (int)(ranf()*(float)op->nopts)];
    }
    x+=xinc[w->orientation];
    y+=yinc[w->orientation];
    if (!Wrap||x!=last||y!=bottom) {
      pushmatrix();
      translate((float)x,(float)y,0.0);
      if(bigblox)
```

```
          scale(2.0,2.0,1.0);
        putsegment(flavor[n%6],segobj[n%6]);
        popmatrix();
      }
      ref[w->ypos[h]=y][w->xpos[h]=x]++;
    }
  }
}
checkmouse()
{
  short dev, val;
  static int upcount;
  if(upcount++ != 20)
    return;
  if(slowmode)
    sleep(2);
  upcount = 0;
  gsync();
  while(qtest()) {
    dev = qread(&val);
    switch(dev) {
    case RIGHTMOUSE:
      m1 = val;
      coffset++;
      break;
    case MIDDLEMOUSE:
      m2 = val;
      if(val)
        slowmode = 1-slowmode;
      break;
    case LEFTMOUSE:
      m3 = val;
      if(val)
        bigblox = 1-bigblox;
      break;
    case REDRAW:
      reshapeviewport();
      makeframe();
      break;
    }
    if(m1 && m3) {
      color(0);
      clear();
    }
  }
}
float ranf() {
  return ((rand()>>1) % 10000)/10000.0;
}
putfield()
{
  color(3);
  callobj(SEG0);
}
putsegment(col,obj)
int col;
int obj;
{
  color((col+coffset)%8);
  callobj(obj);
}
```

```
puttrail()
{
  callobj(TRAIL_OBJ);
}
makeobjects()
{
  makeobj(SEG0);
  rectf(-INCREMENT/3.0, -INCREMENT/3.0, INCREMENT/3.0,
      INCREMENT/3.0);
  closeobj();
  makeobj(SEG1);
  rectf(-INCREMENT/3.0, -INCREMENT/3.0, INCREMENT/3.0,
      INCREMENT/3.0);
  closeobj();
  makeobj(TRAIL_OBJ);
  color(7);
  rectf(-INCREMENT/3.0, -INCREMENT/3.0, INCREMENT/3.0,
      INCREMENT/3.0);
  closeobj();
}
makeframe()
{
  color(0);
  clear();
  ortho2(-1.5,CO+0.5,-1.5,LI+0.5);
  color(7);
  recti(-1,-1,CO,LI);
}
```

Related Information

The **gsync** subroutine .

---

xfonts.c Example C Language Program

```
/*
xfonts.c
This program demonstrates how to use Enhanced X-Windows
fonts in a
GL application
*/
#include <gl/gl.h>
#include <X11/Xlib.h>
#include <stdio.h>
main()
{
  Int32 wid;                    /* the GL window ID */
  Display *dpy;                 /* structure describing X session*/
  int num_fonts;                /* number of fonts X server found */
  char **fontlist;              /* array of strings
                                   (available font names) */
  prefsize (150,150);
  wid = winopen ("xfonts");
  color (BLACK);
  clear();
  /* get the connection to X */
  /* Normally, the default display is unix:0 */
  dpy = XOpenDisplay ("unix:0");
   /* Get the names of all fonts that might be Helvetica fonts */
   /* (have "helv" appearing in their font name) */
  fontlist = XListFonts (dpy, "*helv*", 1000, &num_fonts);
```

```
   /* other useful X font subroutines are:
    * XListFonts
    * XListFontsWithInfo
    * XFreeFontNames
    * XFreeFontInfo
    * XSetFontPath
    * XGetFontPath
    * XFreeFontPath
    */
  /* We have decided, by some criteria, to use the fifth
   * available Helvetica font (assuming that at least five
   * Helvetica fonts are found). We will give it an id of 433.
   */
  if ( num_fonts < 4 ) {
    fprintf (stderr,"Not enough fonts found!!!\n");
    exit (-1);
  }
  loadXfont (433, fontlist[4]);
   /* get rid of the font name list */
  XFreeFontNames (fontlist);
   /* we want to use this font to draw a string */
  font (433);   /* do it */
  cmov2 (43.0, 56.0);
  color (RED);
  charstr ("Hello, World");
  sleep (5);
}
/*
  Changes:
     -  from wid = winopen "xfonts"); to wid = winopen ("xfonts");
*/
```

Related Information

The **loadXfont** subroutine loads an Enhanced X-Windows font into the font table.

Fonts in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* describes font assignment in GL, and discusses relevant subroutines.

---

zbuffer1.c Example C Language Program

```
/*
zbuffer1.c:
A zbuffer() demo program that draws two intersecting planes.
This program requires the z-buffer option.
*/
#include <gl/gl.h>
#include <gl/device.h>
main()
{
   Device dev;
   short val;
   initialize();
   while (TRUE)
   {
      if (qtest())
      {
         dev = qread(&val);
         if (dev == ESCKEY)
         {
```

```
            zbuffer(FALSE);
            gexit();
            exit(0);
         }
         else if (dev == REDRAW)
         {
            reshapeviewport();
            drawpolys();
         }
      }
   }
}
initialize()
{
   int gid;
   float xmax,ymax;
   prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4, YMAXSCREEN/4,
       YMAXSCREEN*3/4);
   gid = winopen("zbuffer1");
   winset(gid);
   winconstraints();
   perspective(900, 1.34, 1.01, 500.0);
   lookat(-150.0, 90.0, 250.0, 50.0, 50.0, 0.0, 0);
   lsetdepth(0xC00000,0x3FFFFF);
   qdevice(ESCKEY);
   qdevice(REDRAW);
   qenter(REDRAW,gid);
   zbuffer(TRUE);
}
drawpolys()
{
   zclear();
   color(BLACK);
   clear();
   color(YELLOW);
   pmv(0.0, 0.0, 100.0);
   pdr(100.0, 0.0, 100.0);
   pdr(100.0, 100.0, 100.0);
   pdr(0.0, 100.0, 100.0);
   pclos();
   color(RED);
   pmv(0.0, 0.0, 50.0);
   pdr(100.0, 0.0, 50.0);
   pdr(100.0, 100.0, 200.0);
   pdr(0.0, 100.0, 200.0);
   pclos();
}
```

Related Information

The **lookat** subroutine, **pclos** subroutine, **pdr** subroutine, **perspective** subroutine, **pmv** subroutine, **zbuffer** subroutine, **zclear** subroutine .

---

zbuffer2.c Example C Language Program

```
/* Example: zbuffer2.c */
/*
This program illustrates z buffering. It draws three parallel
pipeds rotating through each other.
This program requires a 24-bit adapter and the z buffer option.
*/
```

```
#include <gl/gl.h>
#include <gl/device.h>

float v0[3] = {-1.0, -1.0, -1.0} ;
float v1[3] = { -1.0, -1.0, 1.0};
float v2[3] = { -1.0, 1.0, 1.0};
float v3[3] = { -1.0, 1.0, -1.0};
float v4[3] = { 1.0, -1.0, -1.0};
float v5[3] = { 1.0, -1.0, 1.0};
float v6[3] = { 1.0, 1.0, 1.0};
float v7[3] = { 1.0, 1.0, -1.0};

Int32 delaycount;

main(argc, argv)
int     argc;
char    *argv[];
{
   Int32 xrot, yrot, zrot;
   xrot = yrot = zrot = 0;
   if (argc == 1)
      delaycount = 0;
   else
      delaycount = 1;
   keepaspect(1, 1);
   prefposition (0,800,0,800);
   winopen("zbuffer");
   RGBmode();
   if (delaycount == 0)
      doublebuffer();
   gconfig();
   ortho(-4.0, 4.0, -4.0, 4.0, -4.0, 4.0);
   while (1) {
      pushmatrix();
      rotate(xrot, 'x');
      rotate(yrot, 'y');
      rotate(zrot, 'z');
      xrot += 11;
      yrot += 15;
      if (xrot + yrot > 3500) zrot += 23;
      if (xrot > 3600) xrot -= 3600;
      if (yrot > 3600) yrot -= 3600;
      if (zrot > 3600) zrot -= 3600;
      cpack(0);
      clear();
      if (getbutton(LEFTMOUSE)) {
         zbuffer(TRUE);
         zclear();
      }
      else
         zbuffer(FALSE);
      pushmatrix();
      scale(1.2, 1.2, 1.2);
      translate(.3, .2, .2);
      if (getbutton(MIDDLEMOUSE)) {
         reshapeviewport();
      }
      drawcube();
      popmatrix();
      pushmatrix();
      rotate(450+zrot, 'x');
      rotate(300-xrot, 'y');
      scale(1.8, .8, .8);
      drawcube();
      popmatrix();
      pushmatrix();
      rotate(500+yrot, 'z');
      rotate(-zrot, 'x');
      translate(-.3, -.2, .6);
```

```
      scale(1.4, 1.2, .7);
      drawcube();
      popmatrix();
      if (delaycount == 0)
         swapbuffers();
      popmatrix();
   }
}
delay()
{
   sleep(delaycount);
}
drawcube()
{
   cpack(0xff);
   bgnpolygon();
   v3f(v0);
   v3f(v1);
   v3f(v2);
   v3f(v3);
   endpolygon();
   delay();
   cpack(0xff00);
   bgnpolygon();
   v3f(v0);
   v3f(v4);
   v3f(v5);
   v3f(v1);
   endpolygon();
   delay();
   cpack(0xff0000);
   bgnpolygon();
   v3f(v4);
   v3f(v7);
   v3f(v6);
   v3f(v5);
   endpolygon();
   delay();
   cpack(0xffff);
   bgnpolygon();
   v3f(v3);
   v3f(v7);
   v3f(v6);
   v3f(v2);
   endpolygon();
   delay();
   cpack(0xff00ff);
   bgnpolygon();
   v3f(v5);
   v3f(v1);
   v3f(v2);
   v3f(v6);
   endpolygon();
   delay();
   cpack(0xffff00);
   bgnpolygon();
   v3f(v0);
   v3f(v4);
   v3f(v7);
   v3f(v3);
   endpolygon();
}
```

```
/*
  Changes:
     - Adding reshapeviewport when middlemouse
     - Added a prefposition(0,800,0,800);
*/
```

Related Information

The **zbuffer** subroutine, **zclear** subroutine .

Zbuffering in *GL3.2 Version 4.1 for AIX: Programming Concepts (POWER-based Systems Only)* introduces basic concepts about displaying objects closest to the viewer.

---

zoing.c Example C Language Program

```
/*
zoing.c
Make a spiral out of circles.

Paul Haeberli - 1984
*/

#include <gl/device.h>
#include <gl/gl.h>

main()
{
   short dev,val;

   keepaspect(1,1);
   prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
       YMAXSCREEN*3/4);
   winopen("zoing");
   qdevice(ESCKEY);
   drawit();
   while(1) {
      if((dev = qread(&val)) == REDRAW)
         drawit();
      else if (dev == ESCKEY) {
         gexit();
         exit();
      }
   }
}

drawit()
{
   register int i;

   reshapeviewport();
   color(7);
   clear();
   ortho2(-1.0,1.0,-1.0,1.0);
   color(0);
   translate(-0.1,0.0,0.0);
   pushmatrix();
   for(i=0; i<200; i++)  {
      rotate(170,'z');
      scale(0.96,0.96,0.0);
      pushmatrix();
      translate(0.10,0.0,0.0);
      circ(0.0,0.0,1.0);
      popmatrix();
   }
   popmatrix();
}
```

Related Information

Appendix. Notices

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
IBM Corporation
Dept. LRAS/Bldg. 003
11400 Burnet Road
Austin, TX 78758-3498
U.S.A.
Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and

cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Index

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Readers' Comments — We'd Like to Hear from You

GL3.2 for AIX: Graphics Library (GL) Technical Reference (POWER-based Systems only)

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Corporation
Publications Department
Internal Zip 9561
11400 Burnet Road
Austin, TX
78758-3493

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A