IBM

AIX Version 3 for RISC System/6000™

Calls and Subroutines Reference: Graphics

Volume 6

SC23-2198-00

First Edition (March 1990)

This edition of the *AIX Calls and Subroutines Reference for IBM RISC System/6000* applies to IBM AIX Version 3 for RISC System/6000, Version 3 of IBM AIXwindows Environment/6000, IBM AIX System Network Architecture Services/6000, IBM AIX 3270 Host Connection Program/6000, IBM AIX 3278/79 Emulation/6000, IBM AIX Network Management/6000, and IBM AIX Personal Computer Simulator/6000 and to all subsequent releases of these products until otherwise indicated in new releases or technical newsletters.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND 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.

IBM does not warrant that the contents of this publication or the accompanying source code examples, whether individually or as one or more groups, will meet your requirements or that the publication or the accompanying source code examples are error–free.

This publication 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.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758–3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Trademarks and Acknowledgements

The following trademarks and acknowledgements apply to this information:

AIX is a trademark of International Business Machines Corporation.

AIXwindows is a trademark of International Business Machines Corporation.

Apollo is a trademark of Apollo Computer, Inc.

IBM is a registered trademark of International Business Machines Corporation.

NCK is a trademark of Apollo Computer, Inc.

NCS is a trademark of Apollo Computer, Inc.

Network Computing Kernel is a trademark of Apollo Computer, Inc.

Network Computing System is a trademark of Apollo Computer, Inc.

Network File System and NFS are trademarks of Sun Microsystems, Inc.

POSIX is a trademark of the Institute of Electrical and Electronic Engineers (IEEE).

RISC System/6000 is a trademark of International Business Machines Corporation.

SNA 3270 is a trademark of International Business Machines Corporation.

UNIX was developed and licensed by AT&T and is a registered trademark of AT&T Corporation.

X/OPEN is a trademark of X/OPEN Company Limited.

Note to Users

The term "network information services (NIS)" is now used to refer to the service formerly known as "Yellow Pages." The functionality remains the same; only the name has changed. The name "Yellow Pages" is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

Legal Notice to Users Issued by Sun Microsystems, Inc.

"Yellow Pages" is a registered trademark in the United Kingdom of British Telecommunications plc, and may also be a trademark of various telephone companies around the world. Sun will be revising future versions of software and documentation to remove references to "Yellow Pages."

About This Book

AIX Calls and Subroutines Reference for IBM RISC System/6000, SC23–2198, is divided into the following four major sections:

- Volumes 1 and 2, *Calls and Subroutines Reference*: *Base Operating System*, contains reference information about the system calls, subroutines, functions, macros, and statements associated with AIX base operating system runtime services, communications services, and device services.
- Volumes 3 and 4, *Calls and Subroutines Reference*: *User Interface*, contain reference information about the AIXwindows widget classes, subroutines, and resource sets; the AIXwindows Desktop resource sets; the Enhanced X–Windows subroutines, macros, protocols, extensions, and events; the X–Window toolkit subroutines and macros; and the curses and extended curses subroutine libraries.
- Volume 5, *Calls and Subroutines Reference*: *Kernel Reference*, contains reference information about kernel services, device driver operations, file system operations subroutines, the configuration subsystem, the communications subsystem, the high function terminal (HFT) subsystem, the logical volume subsystem, the printer subsystem, and the SCSI subsystem.
- Volumes 6, *Calls and Subroutines Reference*: *Graphics*, contains reference information and example programs for the Graphics Library (GL) and the AIXwindows Graphics Support Library (XGSL) subroutines. These two application programming interfaces to the Advanced Interactive Executive Operating System (referred to in this text as AIX) are used on the IBM RISC System/6000.

Who Should Use This Book

This book is intended for experienced graphics programmers who want to write graphics applications using either GL or XGSL. Readers of this book are expected to know the C programming language and should be familiar with AIX commands, file formats, and special files.

How to Use This Book

Overview of Contents

Chapters 1 and 2 respectively comprise the subroutines and example programs for GL, arranged alphabetically. Likewise, Chapters 3 and 4 contain the subroutines and example programs for XGSL. In addition, special terms used in GL are included at the end of the book.

The examples given in this book are merely examples, provided for the sole purpose of illustrating that the GL or XGSL basic subroutines can be used to create extended or enhanced subroutines. The subroutines are provided "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of each of the GL or XGSL subroutines is with you.

Before writing an application program that uses either of these interfaces, see *Graphics Programming Concepts*, which describes the major concepts and functionality of both GL and XGSL.

Note: XGSL parameters are passed by reference, making the subroutines compatible with FORTRAN, in which parameters are always passed by reference. All parameters are therefore passed as pointers in C programming language.

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies subroutines, commands, keywords, files, directories, and other items whose names are predefined by the system.
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.

Related Publications

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

- *AIX Graphics Programming Concepts for IBM RISC System/6000*, Order Number SC23–2208.
- *AIX User Interface Programming Concepts for IBM RISC System/6000*, Order Number SC23–2209.

Ordering Additional Copies of This Book

To order additional copies of this book, use Order Number SC23–2198.

Contents

Part 1. Graphics Library Reference (GL)

Part 2. AIXwindows Graphics Support Library Reference (XGSL)

Part 1. Graphics Library Reference (GL)

Chapter 1. GL Subroutines

addtopup Subroutine

Purpose

Adds items to an existing pop-up menu.

Library

Graphics Library (**libgl.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.

%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.

%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.

%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.

Note: 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.

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating and Managing Pop-Up Menus in GL in *Graphics Programming Concepts*.

arc Subroutine

Purpose

Draws a circular arc.

Library

Graphics Library (**libgl.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*2 *startang*, *endang*

SUBROUTINE ARCI(*x*, *y*, *radius*, *startang*, *endang***)**

INTEGER*4 *x*, *y*, *radius*

INTEGER*2 *startang*, *endang*

SUBROUTINE ARCS(*x*, *y*, *radius*, *startang*, *endang***)**

INTEGER*2 *x*, *y*, *radius*

INTEGER*2 *startang*, *endang*

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

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GLIntroduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

arcf Subroutine

Purpose

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

Library

Graphics Library (**libgl.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*2 *startang*, *endang*

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

INTEGER*4 *x*, *y*, *radius*

INTEGER*2 *startang*, *endang*

SUBROUTINE ARCFS(*x*, *y*, *radius*, *startang*, *endang***)**

INTEGER*2 *x*, *y*, *radius*

INTEGER*2 *startang*, *endang*

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

1. To draw a scoop of ice cream, the example C language program **tpbig.c** uses the **arcfi** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GLIntroduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

backbuffer Subroutine

Purpose

Enables or disables drawing into the back buffer.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

Configuring the Frame Buffer, Creating Animated Screens, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

backface Subroutine

Purpose

Enables and disables backfacing polygon removals.

Library

Graphics Library (**libgl.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. It is useful for drawing polygons that have different colors on each side. It 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.

Note: 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.

Parameter

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

TRUE = backfacing polygon removal is enabled.

FALSE = backfacing polygon removal is disabled.

Example

1. To demonstrate the difference between allowing or suppressing the display of backfacing polygons, the example C language program **backface.c** uses the **backface** subroutine draw a cube in both ways.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

backface

File

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

Related Information

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

GL Introduction, Removing Hidden Surfaces in *Graphics Programming Concepts*.

bbox2 Subroutine

Purpose

Specifies the bounding box and minimum pixel radius.

Library

Graphics Library (**libgl.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*2 *xmin*, *ymin*
REAL *x1*, *y1*, *x2*, *y2*

SUBROUTINE BBOX2I(*xmin*, *ymin*, *x1*, *y1*, *x2*, *y2***)**
INTEGER*2 *xmin*, *ymin*
INTEGER*4 *x1*, *y1*, *x2*, *y2*

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

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.

File

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

Related Information

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

GL Introduction, Creating Objects (Display Lists) in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

bgnclosedline Subroutine

Purpose

Begins interpretation of vertex routines as closed line vertices.

Library

Graphics Library (**libgl.a**)

C Syntax

void bgnclosedline( )

FORTRAN Syntax

SUBROUTINE BGNCLO

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.

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

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

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

You cannot specify more than 256 vertices between the **bgnclosedline** and **endclosedline** subroutines..

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

Ending a series of closed line vertices with the **endclosedline** subroutine.

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

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

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

bgnline Subroutine

Purpose

Begins interpretation of vertex routines as line vertices.

Library

Graphics Library (**libgl.a**)

C Syntax

void bgnline( )

FORTRAN Syntax

SUBROUTINE BGNLIN

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.

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

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

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

You cannot specify more than 256 vertices between the **bgnline** and **endline** subroutines..

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Ending a series of vertex-based lines with the **endline** subroutine.

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

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

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

bgnpoint Subroutine

Purpose

Begins interpretation of vertex subroutines as points.

Library

Graphics Library (**libgl.a**)

C Syntax

void bgnpoint( )

FORTRAN Syntax

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. 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**
- **lmcolor**
- **lmbind**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

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

You cannot specify more than 256 vertices between the **bgnpoint** and **endpoint** subroutines.

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

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

Ending a series of vertex-based points with the **endpoint** subroutine.

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

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts.*

bgnpolygon Subroutine

Purpose

Begins interpretation of vertex subroutines as polygon vertices.

Library

Graphics Library (**libgl.a**)

C Syntax

void bgnpolygon( )

FORTRAN Syntax

SUBROUTINE BGNPOL

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**
- **lmcolor**
- **lmbind**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

After the **endpolygon** subroutine, 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. You cannot specify more than 256 vertices between the **bgnpolygon** and **endpolygon** subroutines.

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

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

Warning: Calling **concave(TRUE)** guarantees that all polygons are drawn correctly, but may cause degradation of performance.

Example

1. To define the beginning of a description of a polygon, the example C language program **cylinder2.c** uses the **bgnpolygon** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

Ending a vertex-based polygon with the **endpolygon** subroutine.

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

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

bgnsurface or endsurface Subroutine

Purpose

Delimit a NURBS surface definition.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Creating Lighting Effects, and Drawing NURBS Curves and Surfaces in *Graphics Programming Concepts*.

bgntmesh Subroutine

Purpose

Begins interpretation of vertex subroutines as triangle mesh vertices.

Library

Graphics Library (**libgl.a**)

C Syntax

void bgntmesh( )

FORTRAN Syntax

SUBROUTINE BGNTME

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**
- **lmcolor**
- **lmbind**
- **lmdef**
- **n3f**
- **normal**
- **RGBcolor**
- **v**

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

You cannot specify more than 256 vertices between the **bgntmesh** and **endtmesh** subroutines.

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.

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

Ending a series of triangle mesh vertices with the **endtmesh** subroutine.

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

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

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

bgntrim or endtrim Subroutine

Purpose

Delimit a NURBS surface trimming loop.

Library

Graphics Library (**libgl.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 calls to the **bgntrim** and **endtrim** subroutines) are those to 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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing NURBS Curves and Surfaces in *Graphics Programming Concepts*.

blankscreen Subroutine

Purpose

Turns screen refresh on and off.

Library

Graphics Library (**libgl.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.

Note: This subroutine cannot be used to add to a display list.
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 refresh. The settings for the *bool* parameter are:

TRUE stops display and turns screen black IMMEDIATELY.

FALSE restores the display.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Setting the screen blanking timeout with the **blanktime** subroutine.

Creating and Managing Windows, Understanding the Hardware Used by GL, and Windows and Input Control in *Graphics Programming Concepts*.

blanktime Subroutine

Purpose

Sets the screen blanking timeout.

Library

Graphics Library (**libgl.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.*

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

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.

File

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

Related Information

Creating and Managing Windows, Understanding the Hardware Used by GL, and Windows and Input Control in *Graphics Programming Concepts.*

blendfunction Subroutine

Purpose

Specifies the function to be used for alpha blending.

Library

Graphics Library (**libgl.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.

BF_ZERO 0

BF_ONE 1

BF_SC (source RGBA)/(maximum value)

BF_MSC	1 - (source RGBA)/(maximum value)
BF_SA	(source alpha)/(maximum value)
BF_MSA	1 - (source alpha)/(maximum value)
BF_DA	(destination alpha)/(maximum value)
BF_MDA	1 - (destination alpha)/(maximum value)

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

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.

The blending factors BF_DA and BF_MDA are supported on machines with alpha bitplanes.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

blink Subroutine

Purpose

Changes a color map entry at a selectable rate.

Library

Graphics Library (**libgl.a**)

C Syntax

void blink(Int16 *rate* **, Colorindex** *color* **, Int16** *red* **, Int16** *green* **, Int16** *blue* **)**

FORTRAN Syntax

SUBROUTINE BLINK(*rate, color, red, green, blue***)**
INTEGER*2 *rate, color, red, green, blue*

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 colormap 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 60Hz 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.

File

/usr/include/gl/gl.h Contains 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.

Creating a Cursor, Creating Animated Screens, Working in Color Map and RGB Modes, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

blkqread Subroutine

Purpose

Reads multiple entries from the event queue.

Library

Graphics Library (**libgl.a**)

C Syntax

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

FORTRAN Syntax

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

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.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

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.

GL Introduction and Controlling Queues and Devices in GL in *Graphics Programming Concepts*.

c Subroutine

Purpose

Sets the current color in RGB mode.

Library

Graphics Library (**libgl.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.

Notes:

1. This subroutine is available only in RGB mode.
2. This subroutine cannot be used to add to a display list.

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

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

File

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

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.

Setting Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

callobj Subroutine

Purpose

Draws an instance of an object (display list).

Library

Graphics Library (**libgl.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

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Creating Objects (Display Lists) in GL, Setting Attributes, and Working with Coordinate Systems in *Graphics Programming Concepts*.

charstr Subroutine

Purpose

Draws a string of raster characters on the screen.

Library

Graphics Library (**libgl.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 a raster font. After each character is drawn, the character's width is added to the current character position. The text string is drawn in the current raster font and color, using the current writemask.

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

Parameters

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

Example

1. To draw a character string in the current raster font, the example C language program **curved.c** uses the **charstr** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

This subroutine is not available for Japanese Language Support.

File

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

Related Information

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.

GL Introduction and Creating Text Characters in GL in *Graphics Programming Concepts.*

chunksize Subroutine

Purpose

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

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts.*

circ Subroutine

Purpose

Outlines a circle.

Library

Graphics Library (**libgl.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*

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

Example

1. To draw a red circle, the example C language program **boxcirc.c** uses the **circi** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GLIntroduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

circf Subroutine

Purpose

Draws a filled circle.

Library

Graphics Library (**libgl.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*

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

1. To draw a scoop of cherry ice cream, the example C language program **tpbig.c** uses the **circf** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GLIntroduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts.*

clear Subroutine

Purpose

Clears to the screenmask.

Library

Graphics Library (**libgl.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.

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.

Example

1. To clear the viewport before drawing a box and a circle, the example C language program **boxcirc.c** uses the **clear** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Getting Ready to Run GL, Starting GL Functions, Setting Attributes in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

clkoff or clkon Subroutine

Purpose

Turns on or off the keyboard click.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Controlling the Keyboard in *Graphics Programming Concepts.*

closeobj Subroutine

Purpose

Closes an object.

Library

Graphics Library (**libgl.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

1. To specify the end of a graphical object definition, the example C language program **depthcue.c** uses the **closeobj** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts.*

cmode Subroutine

Purpose

Sets color map mode as the current mode.

Library

Graphics Library (**libgl.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 colormap 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 colormap mode unti 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.

File

/usr/include/gl/gl.h Contains 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 in *Graphics Programming Concepts*.

cmov Subroutine

Purpose

Updates the current text character position.

Library

Graphics Library (**libgl.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*

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

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

This subroutine is not available for Japanese Language Support.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Creating Text Characters in GL, Using Viewports and Screenmasks in GL, and Working with Coordinate Systems in GL in *Graphics Programming Concepts*.

color or colorf Subroutine

Purpose

Sets the current color in color map mode.

Library

Graphics Library (**libgl.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, zero, two, or four 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

1. To set the color for subsequent drawing routines, the example C language program **boxcirc.c** uses the **color** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 colormap 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.

Setting Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

compactify Subroutine

Purpose

Unfragments and compacts the memory storage of an object.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

concave Subroutine

Purpose

Forces the system to draw accurate concave polygons.

Library

Graphics Library (**libgl.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 the value for concave polygons. The settings for the *bool* parameter are:

TRUE forces the system to draw accurate concave and convex polygons.

FALSE tells the system to draw only accurate convex polygons. This is the default.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GLIntroduction, Setting Pipeline Options, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

cpack Subroutine

Purpose

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

Library

Graphics Library (**libgl.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.

The **cpack** subroutine produces unpredictable results if called while a lighting model is active.

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

Parameter

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

Example

1. 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.

File

/usr/include/gl/gl.h	Contains 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.

Setting Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

crv Subroutine

Purpose

Draws a cubic spline curve.

Library

Graphics Library (**libgl.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

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in *Graphics Programming Concepts*.

crvn Subroutine

Purpose

Draws a series of curve segments.

Library

Graphics Library (**libgl.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

1. To draw a curve with six control points, the example C language program **curve2.c** uses the **crvn** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in *Graphics Programming Concepts*.

curorigin Subroutine

Purpose

Sets the origin of a cursor.

Library

Graphics Library (**libgl.a**)

C Syntax

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

FORTRAN Syntax

SUBROUTINE CURORI(*index*, *xorigin*, *yorigin*)
INTEGER*2 *index*, *xorigin*, *yorigin*

Description

The **curorigin** subroutine sets the origin of a cursor. Before calling this subroutine, the cursor must be defined with the **defcursor** subroutine and set with the **setcursor** subroutine. The lower left corner of the cursor has coordinates (0,0).

The default origin for a user-defined cursor is (0,0).

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.

File

/usr/include/gl/gl.h	Contains 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.

Creating a Cursor in GL and Creating and Managing Windows in GL in *Graphics Programming Concepts*.

curson or cursoff Subroutine

Purpose

Controls cursor visibility by window.

Library

Graphics Library (**libgl.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

1. To turn off the cursor while drawing, the example C language program **text.c** uses **cursoff** subroutine. 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.

File

/usr/include/gl/gl.h Contains 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.

Creating a Cursor in GL and Creating and Managing Windows in GL in *Graphics Programming Concepts.*

curstype Subroutine

Purpose

Defines the type and size of the cursor.

Library

Graphics Library (**libgl.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.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Creating a Cursor in GL and Creating and Managing Windows in GL in *Graphics Programming Concepts*.

curvebasis Subroutine

Purpose

Sets the current cubic spline curve basis matrix.

Library

Graphics Library (**libgl.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

1. 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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in *Graphics Programming Concepts*.

curveit Subroutine

Purpose

Draws a curve segment by iterating the forward difference matrix.

Library

Graphics Library (**libgl.a**)

C Syntax

void curveit(Int16 *count***)**

FORTRAN Syntax

SUBROUTINE CURVEI(*count***)**
INTEGER*2 *count*

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

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

GL Introduction and Drawing Wire Frame Curves and Surface Patches in *Graphics Programming Concepts*.

curveprecision Subroutine

Purpose

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

Library

Graphics Library (**libgl.a**)

C Syntax

void curveprecision(Int16 *nsegments***)**

FORTRAN Syntax

SUBROUTINE CURVEP(*nsegments***)**
INTEGER*2 *nsegments*

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

1. To draw three curves, the example C language program **curve1.c** uses the **curveprecision** subroutine so that each curve will be approximated by 20 line segments.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts*.

cyclemap Subroutine

Purpose

Cycles through color maps at a specified rate.

Library

Graphics Library (**libgl.a**)

C Syntax

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

FORTRAN Syntax

SUBROUTINE CYCLEM(*duration, map, nextmap***)**
INTEGER*2 *duration, map, nextmap*

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.

File

/usr/include/gl/gl.h	Contains 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.

Configuring the Frame Buffer, Creating Animated Screens, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

czclear Subroutine

Purpose

Clears the color bitplanes and the z-buffer simultaneously.

Library

Graphics Library (**libgl.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. 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.

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

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Getting Ready to Run GL, Starting GL Functions, Configuring the Frame Buffer for GL, Controlling Frame Buffer Update in GL, and Working in Color Map and RGB Modes in GL in *Graphics Programming Concepts*.

defbasis Subroutine

Purpose

Defines a cubic spline basis matrix.

Library

Graphics Library (**libgl.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

1. To define three basis matrices for drawing curves, the example C language program **curve1.c** uses the **defbasis** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts*.

defcursor Subroutine

Purpose

Defines a cursor glyph.

Library

Graphics Library (**libgl.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 either 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.

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.

File

/usr/include/gl/gl.h	Contains 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.

Creating a Cursor in GL and Creating and Managing Windows in GL in *Graphics Programming Concepts*.

deflinestyle Subroutine

Purpose

Defines a linestyle.

Library

Graphics Library (**libgl.a**)

C Syntax

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

FORTRAN Syntax

SUBROUTINE DEFLIN(*index*, *linestyle*)
INTEGER*4 *index*
INTEGER*2 *linestyle*

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.

Note: 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

1. To define a dashed line, the example C language program **curved.c** uses the **deflinestyle** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

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

File

/usr/include/gl/gl.h	Contains 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.

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 Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

defpattern Subroutine

Purpose

Defines patterns used in area fills.

Library

Graphics Library (**libgl.a**)

C Syntax

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

FORTRAN Syntax

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

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.

Note: 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: 16, 32, or 64 for a 16x16-bit, 32x32-bit, or 64x64-bit pattern, respectively.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

On some display adapters, notably the High-Performance 3-D Graphics Processor, there is a performance penalty for non-solid patterns. Also, the High-Performance 3-D Graphics Processor allows only 16x16-bit, 32x32-bit, or 64x64-bit patterns.

File

/usr/include/gl/gl.h	Contains 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.

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 Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

defpup Subroutine

Purpose

Defines a pop-up menu.

Library

Graphics Library (**libgl.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

1. To define a pop-up menu, the example C language program **curved.c** uses the **defpup** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Creating and Managing Pop-Up Menus in GL in *Graphics Programming Concepts*.

defrasterfont Subroutine

Purpose

Defines bitmaps for a raster font.

Library

Graphics Library (**libgl.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*
INTEGER*2 *height*, *numchars*, *numraster*
INTEGER*2 *raster*(*numraster*), *chars*(**4***numchars**)**

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, cursors, and fonts are available to all windows when using multiple windows. The ASCII code page table encoding is assumed.

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;
```

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

1. To define a new raster font, the example C language program **curved.c** uses the **defrasterfont** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

This subroutine is not available for Japanese Language Support.

File

/usr/include/gl/gl.h Contains 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 with the **getheight** subroutine.

defrasterfont

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.

GL Introduction and Creating Text Characters in GL in *Graphics Programming Concepts*.

delobj Subroutine

Purpose

Deletes an object.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

deltag Subroutine

Purpose

Deletes tags from objects.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

depthcue Subroutine

Purpose

Turns depth-cueing on and off.

Library

Graphics Library (**libgl.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

1. To turn depth-cueing on, the example C language program **depthcue.c** uses the **depthcue** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Performing Depth-Cueing, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

dopup Subroutine

Purpose

Displays the specified pop-up menu.

Library

Graphics Library (**libgl.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 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 -1.

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 the pop-up menu to display.

Example

1. To display a popup menu, the example C language program **curved.c** uses the **dopup** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating and Managing Pop-Up Menus in GL in *Graphics Programming Concepts.*

doublebuffer Subroutine

Purpose

Sets the display mode to double buffer mode.

Library

Graphics Library (**libgl.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.

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

Example

1. To allow smooth motion when moving and redrawing a cube, the example C language program **backface.c** sets the display mode for double buffering with the **doublebuffer** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

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.

Configuring the Frame Buffer, Creating Animated Screens, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

draw Subroutine

Purpose

Draws a line.

Library

Graphics Library (**libgl.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*

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

1. To draw the edges of a cube, the example C language program **depthcue.c** uses the **drawi** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Drawing with Move-Draw Style Subroutines, Performing Depth-Cueing, Setting Attributes, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

drawmode Subroutine

Purpose

Specifies the target frame buffer for the drawing subroutines.

Library

Graphics Library (**libgl.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 is chosen. 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 modes and their functions appear in the following table.

Mode Identifier	Function
UNDERDRAW	Redirects drawing into the background (underlay) bitplanes.
NORMALDRAW	Redirects all drawing into the main frame buffer bitplanes.
OVERDRAW	Redirects drawing into the foreground (overlay) bitplanes.
PUPDRAW	Redirects **mapcolor** and **getmcolor** subroutines to affect the pop-up menus only.
CURSORDRAW	Redirects **mapcolor** and **getmcolor** subroutines to affect the cursor only.

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) will 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 will automatically go 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.

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) will 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 will automatically go 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.

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

1. To set the drawing mode for operations on the pop-up menus, the example C language program **prompt.c** uses the **drawmode** subroutine with the PUPDRAW mode identifier.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, Creating Animated Screens, Removing Hidden Surfaces, and Working in Color Map and RGB Modes in *Graphics Programming Concepts.*

editobj Subroutine

Purpose

Opens a display list for editing.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

endclosedline Subroutine

Purpose

Ends the interpretation of vertex subroutines as closed line vertices.

Library

Graphics Library (**libgl.a**)

C Syntax

void endclosedline( )

FORTRAN Syntax

SUBROUTINE ENDCLO

Description

The **endclosedline** subroutine ends the scope of a preceding **bgnclosedline** subroutine. You cannot specify more than 256 vertices between the **bgnclosedline** and **endclosedline** subroutines.

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.

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

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

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

endfullscrn Subroutine

Purpose

Ends full-screen mode.

Library

Graphics Library (**libgl.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.

File

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

Related Information

Giving a program the entire screen as a window with the **fullscrn** subroutine.

Windows and Input Control Overview for GL in *Graphics Programming Concepts.*

endline Subroutine

Purpose

Ends interpretation of vertex subroutines as line vertices.

Library

Graphics Library (**libgl.a**)

C Syntax

void endline( )

FORTRAN Syntax

SUBROUTINE ENDLIN

Description

The **endline** subroutine ends the scope of a preceding **bgnline** subroutine. You cannot specify more than 256 vertices between the **bgnline** and **endline** subroutines.

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

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

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

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

endpick Subroutine

Purpose

Turns off picking mode.

Library

Graphics Library (**libgl.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

1. To turn off picking mode, the example C language program **pick1.c** calls the **endpick** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Working with Coordinate Systems in GL, and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

endpoint Subroutine

Purpose

Ends interpretation of vertex routines as points.

Library

Graphics Library (**libgl.a**)

C Syntax

void endpoint( )

FORTRAN Syntax

SUBROUTINE ENDPOI

Description

The **endpoint** subroutine ends the scope of a preceding **bgnpoint** subroutine. You cannot specify more than 256 vertices between the preceding **bgnpoint** and **endpoint** subroutines.

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

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

endpolygon Subroutine

Purpose

Ends interpretation of vertex routines as polygon vertices.

Library

Graphics Library (**libgl.a**)

C Syntax

void endpolygon( )

FORTRAN Syntax

SUBROUTINE ENDPOL

Description

The **endpolygon** subroutine ends the scope of a preceding **bgnpolygon** subroutine. You cannot specify more than 256 vertices between the preceding **bgnpolygon** and **endpolygon** subroutines.

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

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

Example

1. To end the description of a polygon, the example C language program **cylinder2.c** uses the **endpolygon** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Drawing vertex-based polygons with the **bgnpolygon** subroutine.

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

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

endselect Subroutine

Purpose

Turns off selecting mode.

Library

Graphics Library (**libgl.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

1. To turn off selecting mode, the example C language program **select1.c** uses the **endselect** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

endtmesh Subroutine

Purpose

Ends interpretation of vertex subroutines as triangle mesh vertices.

Library

Graphics Library (**libgl.a**)

C Syntax

void endtmesh( )

FORTRAN Syntax

SUBROUTINE ENDTME

Description

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

You cannot specify more than 256 vertices between the **bgntmesh** and **endtmesh** subroutines.

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Drawing triangle mesh vertices with the **bgntmesh** subroutine.

Setting the current color vector 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.

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

font Subroutine

Purpose

Selects a raster font for drawing text strings.

Library

Graphics Library (**libgl.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

1. To select a raster font defined with the **defrasterfont** subroutine, the example C language program **curved.c** uses the **font** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

This subroutine is not available for Japanese Language Support.

File

/usr/include/gl/gl.h Contains 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 with the **getheight** subroutine.

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

GL Introduction and Creating Text Characters in GL in *Graphics Programming Concepts*.

freepup Subroutine

Purpose

Deallocates a menu.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating and Managing Pop-Up Menus in GL in *Graphics Programming Concepts*.

frontbuffer Subroutine

Purpose

Enables or disables drawing into the front buffer.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

Configuring the Frame Buffer, Creating Animated Screens, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

fudge Subroutine

Purpose

Specifies pixel values that are added to a window.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

fullscrn Subroutine

Purpose

Gives a program the entire screen as a window.

Library

Graphics Library (**libgl.a**)

C Syntax

void fullscrn( )

FORTRAN Syntax

SUBROUTINE FULLSC

Description

The **fullscrn** subroutine gives a program the entire screen as a window. The subroutine calls

```
viewport(0,XMAXSCREEN,0,YMAXSCREEN)
```

and sets up the default coordinate system to be defined by 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.

File

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

Related Information

Ending fullscreen mode with the **endfullscrn** subroutine.

Initializing the graphics system without changing the color map with the **gbegin** subroutine.

Creating a window with the **winopen** subroutine.

Creating and Managing Windows in GL in *Graphics Programming Concepts*.

gammaramp Subroutine

Purpose

Defines a color map ramp for gamma correction.

Library

Graphics Library (**libgl.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*.

Note: 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.

File

/usr/include/gl/gl.h	Contains 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 in *Graphics Programming Concepts*.

gbegin Subroutine

Purpose

Initializes the graphics system without changing the color map.

Library

Graphics Library (**libgl.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.

The recommendation is to use the **winopen** subroutine for initialization functions to take advantage of the window manager.

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Giving a program the entire screen as a window 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.

GL Introduction, Getting Ready to Run GL, Starting GL Functions, Setting Attributes in GL, and Windows and Input Control Overview for GL in *Graphics Programming Concepts*.

gconfig Subroutine

Purpose

Reconfigures the system.

Library

Graphics Library (**libgl.a**)

C Syntax

void gconfig( )

FORTRAN Syntax

SUBROUTINE GCONFI

Description

The **gconfig** subroutine reconfigures the system by setting the requested modes. This subroutine must follow the **overlay**, **underlay**, **doublebuffer**, **multimap**, **onemap**, **RGBmode**, **cmode**, and **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.

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

Example

1. To configure the system after calling the **overlay** subroutine and the **doublebuffer** subroutine, the example C language program **ovrlay.c** uses the **qconfig** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Getting Ready to Run GL, Starting GL Functions, Configuring the Frame Buffer for GL, and Working in Color Map and RGB Modes in GL in *Graphics Programming Concepts*.

genobj Subroutine

Purpose

Returns a unique integer for use as an object identifier.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts.*

gentag Subroutine

Purpose

Returns a unique integer for use as a tag.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts.*

getbackface Subroutine

Purpose

Returns the state of backfacing filled polygon removal.

Library

Graphics Library (**libgl.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.

File

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

Related Information

Allowing or suppressing the display of backfacing polygons with the **backface** subroutine.

GL Introduction, Querying the System, and Removing Hidden Surfaces in *Graphics Programming Concepts*.

getbuffer Subroutine

Purpose

Indicates which buffers are enabled for writing.

Library

Graphics Library (**libgl.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

Buffer Enabled	Symbolic Name
None	NOBUFFER
Back buffer (default)	BCKBUFFER
Front buffer	FRNTBUFFER
Both buffers	BOTHBUFFERS
z-buffer drawing	DRAWZBUFFER

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Configuring the Frame Buffer, Creating Animated Screens, Querying the System, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

getbutton Subroutine

Purpose

Returns the state (up or down) of a button.

Library

Graphics Library (**libgl.a**)

C Syntax

Int32 getbutton(Device *number***)**

FORTRAN Syntax

LOGICAL FUNCTION GETBUT(*number***)**
INTEGER*2 *number*

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

1. To check the state of various buttons, the example C language program **scrn_rotate.c** uses the **getbutton** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant definitions for devices.

Related Information

Returning the current state of a valuator with the **getvaluator** subroutine.

GL Introduction, Controlling Queues and Devices in GL, Controlling the Keyboard in GL, and Querying the System in GL in *Graphics Programming Concepts*.

getcmmode Subroutine

Purpose

Returns the organization of the color map.

Library

Graphics Library (**libgl.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

FALSE = multimap mode

TRUE = onemap mode

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

getcolor Subroutine

Purpose

Returns the current color.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

Creating Animated Screens, Querying the System, Setting Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

getcpos Subroutine

Purpose

Returns the current character position.

Library

Graphics Library (**libgl.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.

This subroutine is not available for Japanese Language Support.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Creating Text Characters in GL, and Querying the System in GL in *Graphics Programming Concepts*.

getcursor Subroutine

Purpose

Returns the cursor characteristics.

Library

Graphics Library (**libgl.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.

File

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

Related Information

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

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

Creating a Cursor in GL, Querying the System, and Creating and Managing Windows in GL in *Graphics Programming Concepts*.

getdcm Subroutine

Purpose

Indicates whether depth-cue mode is on or off.

Library

Graphics Library (**libgl.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.

File

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

Related Information

Turning depth-cueing on and off with the **depthcue** subroutine.

GL Introduction, Performing Depth–Cueing, and Querying the System in *Graphics Programming Concepts.*

getdescender Subroutine

Purpose

Returns the baseline extent of the longest character descender.

Library

Graphics Library (**libgl.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

The length in pixels of the longest descender in the current font.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 with the **getheight** subroutine.

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

GL Introduction, Creating Text Characters in GL, and Querying the System in GL in *Graphics Programming Concepts*.

getdev Subroutine

Purpose

Reads a list of valuators.

Library

Graphics Library (**libgl.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.

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.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

Related Information

Returning the current state of a valuator with the **getvaluator** subroutine.

GL Introduction, Controlling Queues and Devices in GL, Controlling the Keyboard in GL, and Querying the System in GL in *Graphics Programming Concepts*.

getdisplaymode Subroutine

Purpose

Returns the current display mode.

Library

Graphics Library (**libgl.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

Display Mode	Symbolic Name
RGB single buffer mode	DMRGB
color map single buffer mode	DMSINGLE
color map double buffer mode	DMDOUBLE
RGB double buffer mode	DMRGBDOUBLE

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Configuring the Frame Buffer, Creating Animated Screens, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

getdrawmode Subroutine

Purpose

Returns the current drawing mode.

Library

Graphics Library (**libgl.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

Draw Mode	Symbolic Name
main frame buffer	NORMALDRAW
overlay bitplanes	OVERDRAW
underlay (background) bitplanes	UNDERDRAW
pop-up menus	PUPDRAW
cursor mode	CURSORDRAW

Example

1. To get the current drawing mode so that it can be restored later, the example C language program **prompt.c** uses the **getdrawmode** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, Creating a Cursor, Querying the System, Removing Hidden Surfaces, and Working in Color Map and RGB Mode in *Graphics Programming Concepts*.

getfont Subroutine

Purpose

Returns the index of the current raster font.

Library

Graphics Library (**libgl.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.

This subroutine is not available for Japanese Language Support.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Creating Text Characters in GL, and Querying the System in GL in *Graphics Programming Concepts*.

getgpos Subroutine

Purpose

Returns the current graphics position.

Library

Graphics Library (**libgl.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.

File

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

Related Information

Getting the current text character position with the **getcpos** subroutine.

GL Introduction, Drawing with Move–Draw Style Subroutines, Querying the System, and Working in Color Map and RGB Modes in *Graphics Programming Concepts.*

getheight Subroutine

Purpose

Returns the maximum character height in the current raster font.

Library

Graphics Library (**libgl.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.

This subroutine is not available for Japanese Language Support.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Creating Text Characters in GL, and Querying the System in GL in *Graphics Programming Concepts.*

getlsrepeat Subroutine

Purpose

Returns the linestyle repeat count.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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 Attributes, Understanding the Hardware Used by GL, and Using the GL High–Level Drawing Library in *Graphics Programming Concepts.*

getlstyle Subroutine

Purpose

Returns the current linestyle.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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 Attributes, Understanding the Hardware Used by GL, and Using the GL High–Level Drawing Library in *Graphics Programming Concepts*.

getlwidth Subroutine

Purpose

Returns the current linewidth.

Library

Graphics Library (**libgl.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.

File

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

Related Information

Specifying a linewidth with the **linewidth** 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 Attributes, Understanding the Hardware Used by GL, and Using the GL High–Level Drawing Library in *Graphics Programming Concepts*.

getmap Subroutine

Purpose

Returns the number of the current color map.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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 and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

getmatrix Subroutine

Purpose

Returns the current transformation matrix.

Library

Graphics Library (**libgl.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

1. To get the current transformation matrix after manipulating it, the example C language program **scrn_rotate.c** uses the **getmatrix** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Querying the System, and Working with Coordinate Systems in *Graphics Programming Concepts*.

getmcolor Subroutine

Purpose

Gets a copy of the RGB values for a color map entry.

Library

Graphics Library (**libgl.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

1. To save the values in the color map before changing them with the **mapcolor** subroutine, the example C language program **ovrlay.c** uses the **getmcolor** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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 colormap entry to an RGB value with the **mapcolor** subroutine.

Querying the System and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

getmcolors Subroutine

Purpose

Returns a range of color map RGB values.

Library

Graphics Library (**libgl.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*2 *start_idx*, *end_idx*,
INTEGER*2 *r(1)*, *g(1)*, *b(1)*

Description

The **getmcolors** subroutine returns a range of color map table entries. The range that is returned 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 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;
        do (i=0; i< (end_idx – start_idx + 1); i++)
                getmcolors (start_idx+i, &r[i], &g[1], &b[1]);
}
```

The **getmcolors** subroutine can be used to read the underlay, overlay, cursor, popup, 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.

File

/usr/include/gl/gl.h	Contains 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 and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

getmmode Subroutine

Purpose

Returns the current matrix mode.

Library

Graphics Library (**libgl.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:

Mode Name	Mode
MSINGLE	Single matrix mode
MPROJECTION	Projection matrix mode
MVIEWING	Viewing matrix mode.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Creating Lighting Effects, Querying the System, and Setting Pipeline Options in *Graphics Programming Concepts*.

getnurbsproperty Subroutine

Purpose

Returns the current value of a trimmed NURBS surfaces display property

Library

Graphics Library (**libgl.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. The following is a list of the display properties that can be affected.

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

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

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.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Drawing NURBS Curves and Surfaces, and Querying the System in *Graphics Programming Concepts*.

getopenobj Subroutine

Purpose

Returns the current open object.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Creating Objects (Display Lists), and Querying the System in *Graphics Programming Concepts*.

getorigin Subroutine

Purpose

Returns the position of a window.

Library

Graphics Library (**libgl.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

1. To determine the origin of the window, the example C language program **paint.c** uses the **getorigin** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

getpattern Subroutine

Purpose

Returns the index of the current fill pattern.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

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 Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts.*

getplanes Subroutine

Purpose

Returns the number of available bitplanes.

Library

Graphics Library (**libgl.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, 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

1. To get the number of bit-planes available, the example C language program **scrn_rotate.c** uses the **getplanes** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, and Querying the System in *Graphics Programming Concepts*.

getscrmask Subroutine

Purpose

Returns the current screenmask.

Library

Graphics Library (**libgl.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

1. To get the current screenmask so that it can be restored later, the example C language program **prompt.c** uses the **getscrmask** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Querying the System in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

getsize Subroutine

Purpose

Returns the size of a window.

Library

Graphics Library (**libgl.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

1. To determine the size of the window, the example C language program **paint.c** uses the **getsize** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

getsm Subroutine

Purpose

Returns the shading model the system uses to draw filled polygons.

Library

Graphics Library (**libgl.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.

Return Values

There are two possible return values:

FLAT	The system draws a filled polygon with a constant color across the entire surface of the polygon.
GOURAUD	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.

File

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

Related Information

Selecting the shading model used to draw polygons with the **shademodel** subroutine.

Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Querying the System, Setting Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

getvaluator Subroutine

Purpose

Returns the current state of a valuator.

Library

Graphics Library (**libgl.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

1. To obtain the mouse coordinates whenever the left mouse button is pressed, the example C language program **select1.c** uses the **getvaluator** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

Related Information

Reading a list of valuators with the **getdev** subroutine.

GL Introduction, Controlling Queues and Devices in GL, Controlling the Keyboard in GL, and Querying the System in GL in *Graphics Programming Concepts*.

getviewport Subroutine

Purpose

Returns the dimensions of the current viewport.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Querying the System in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

getwritemask Subroutine

Purpose

Returns the current writemask.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, Querying the System, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

getzbuffer Subroutine

Purpose

Determines whether z-buffering is on or off.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h

Contains 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.

GL Introduction, Querying the System, and Removing Hidden Surfaces in *Graphics Programming Concepts*.

gexit Subroutine

Purpose

Terminates a graphics program.

Library

Graphics Library (**libgl.a**)

C Syntax

void gexit( )

FORTRAN Syntax

Description

The **gexit** subroutine is the final graphics routine in a program. It waits for the graphics pipeline to empty and 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.

Example

1. To end graphics processing, the example C language program **scrn_rotate.c** uses the **gexit** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Resetting all global state attributes to their initial values with the **greset** subroutine.

GL Introduction, Getting Ready to Run GL, and Starting GL Functions in *Graphics Programming Concepts.*

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

1. To do a basic graphics setup, the example C language program **prompt.c** uses the **ginit** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Getting Ready to Run GL, Starting GL Functions, Setting Attributes in GL, Controlling Queues and Devices in GL, and Windows and Input Control Overview for GL in *Graphics Programming Concepts*.

greset Subroutine

Purpose

Resets all global state attributes to their initial values.

Library

Graphics Library (**libgl.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	Part 1 of 2
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
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

Global State Attributes	Part 2 of 2
Attribute	**Initial Value**
RGB writemask	undefined
shademodel	GOURAUD
shaderange	0,7,–0x800000,+0x7FFFFF
viewport	entire window
writemask	all planes enabled
zbuffer 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		
Index	**Name**	**Red**	**Green**	**Blue**
0	BLACK	0	0	0
1	RED	255	0	0
2	GREEN	0	255	0
3	YELLOW	255	255	0
4	BLUE	0	0	255
5	MAGENTA	255	0	255
6	CYAN	0	255	255
7	WHITE	255	255	255
All others	Unnamed	Undefined		

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

Example

1. To reset all global attributes, the example C language program **vlsi.c** uses the **greset** subroutine on any keyboard event.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Getting Ready to Run GL, Starting GL Functions, Setting Attributes in GL, Configuring the Frame Buffer for GL, Controlling Queues and Devices in GL, Controlling the Keyboard in GL, Creating a Cursor in GL, Windows and Input Control Overview for GL, and Working with Coordinate Systems in GL in *Graphics Programming Concepts*.

gRGBcolor Subroutine

Purpose

Gets the current RGB color values.

Library

Graphics Library (**libgl.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.

Notes:

1. This subroutine cannot be used to add to a display list.
2. This subroutine is available only in RGB mode. It will not function in color map mode.

Parameters

red	Specifies a pointer to the location into which to copy the current red value.
green	Specifies a pointer to the location into which to copy the current green value.
blue	Specifies a pointer to the location into which to copy the current blue value.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

Querying the System, Setting Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

gRGBmask Subroutine

Purpose

Returns the current RGB writemask.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, Querying the System, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

gselect Subroutine

Purpose

Puts the system in selecting mode.

Library

Graphics Library (**libgl.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/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 pick/select region 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

1. To enter selecting mode, the example C language program **select1.c** uses the **gselect** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Working with Coordinate Systems in GL, and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

gsync Subroutine

Purpose

Waits for a vertical retrace period.

Library

Graphics Library (**libgl.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

1. To help smooth the display while it is changing the frame buffer, the **worms.c** example C language program uses the **gsync** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Configuring the Frame Buffer, Creating Animated Screens, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

gversion Subroutine

Purpose

Returns graphics hardware and library version information.

Library

Graphics Library (**libgl.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.

The **gversion** subroutine can be called before the first call to the **winopen** subroutine.

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

Return Value

The version of GL being used.

Parameters

v	Specifies a buffer into which to 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.

File

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

Related Information

Creating a new window with the **winopen** subroutine.

GL Introduction, Getting Ready to Run GL, and Starting GL Functions in *Graphics Programming Concepts*.

iconsize Subroutine

Purpose

Specifies the icon size of a window.

Library

Graphics Library (**libgl.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.

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.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

icontitle Subroutine

Purpose

Specifies the icon title for the current window.

Library

Graphics Library (**libgl.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.

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.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

initnames Subroutine

Purpose

Initializes the name stack.

Library

Graphics Library (**libgl.a**)

C Syntax

void initnames( )

FORTRAN Syntax

SUBROUTINE INITNA

Description

The **initnames** subroutines initializes the name stack for use during picking or selecting.

Example

1. To clear all of the names from the name stack, the example C language program **pick1.c** calls the **initnames** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

isobj Subroutine

Purpose

Indicates whether a given object number actually identifies an object.

Library

Graphics Library (**libgl.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 or FALSE

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts.*

isqueued Subroutine

Purpose

Indicates whether the specified device is enabled for queuing.

Library

Graphics Library (**libgl.a**)

C Syntax

Int32 isqueued(Int16 *device***)**

FORTRAN Syntax

LOGICAL FUNCTION ISQUEU(*device***)**
INTEGER*2 *device*

Description

The **isqueued** subroutine indicates whether the specified device is enabled for queuing.

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

1. To determine whether the keyboard is enabled, the example C language program **prompt.c** uses the **isqueued** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant definitions for devices.

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.

GL Introduction, Controlling Queues and Devices in GL, and Controlling the Keyboard in GL in *Graphics Programming Concepts*.

istag Subroutine

Purpose

Indicates whether a given tag is used within the current open object.

Library

Graphics Library (**libgl.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 or FALSE

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Returning a unique integer for use as a tag number with the **gentag** subroutine.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

keepaspect Subroutine

Purpose

Specifies the aspect ratio of a window.

Library

Graphics Library (**libgl.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

1. To restrict the window to a specific aspect ratio, the example C language program **colored.c** uses the **keepaspect** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

lampoff or lampon Subroutine

Purpose

Turns the keyboard display lights off or on.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Controlling the Keyboard in *Graphics Programming Concepts*.

lgetdepth Subroutine

Purpose

Gets the distance of the near and far clipping planes.

Library

Graphics Library (**libgl.a**)

C Syntax

void lgetdepth(Int32 *near*, **Int32** *far*)

FORTRAN Syntax

SUBROUTINE LGETDE(*near*, *far*)
INTEGER*2 *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.

File

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

Related Information

Setting the viewport depth range with the **lsetdepth** subroutine.

GL Introduction, Querying the System in GL, Working with Coordinate Systems in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

linesmooth Subroutine

Purpose

Turns line antialiasing on and off.

Library

Graphics Library (**libgl.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 mode. The **linesmooth** hardware replaces the least significant 4 bits of the current color index with bits that represent pixel coverage. Therefore, a 16–entry block of the color map (whose lowest entry is a multiple of 16) must be initialized as a ramp between the background color (lowest index) to the line color (highest index).

Before drawing the lines, clear the area to the background color using the **poly** or **clear** subroutine. If you define many such ramps, you can draw antialiased lines with different colors and intensities by changing the current color index (only the upper bits are significant). You can draw depth–cued, antialiased lines in this manner. The following conditions are required for antialiased lines to draw properly:

1. linewidth = 1
2. linestyle = **0xFFFF**
3. value of the **lsrepeat** subroutine = **1**

The **zsource** and **zfunction** subroutines can be used with the **linesmooth** subroutine for depth or color values. When the **zsource** subroutine is used with **ZSRC_COLOR**, intersecting lines behave more correctly.

The **linesmooth** subroutine does not support subpixel positioning of line vertices.

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

Parameter

mode	SML_ON enables smooth lines. SML_OFF disables smooth lines.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

Configuring the Frame Buffer, Setting Attributes, Smoothing Jagged Lines with Antialiasing, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

linewidth Subroutine

Purpose

Specifies the linewidth.

Library

Graphics Library (**libgl.a**)

C Syntax

void linewidth(Int16 *number***)**

FORTRAN Syntax

SUBROUTINE LINEWI(*number***)**
INTEGER*2 *number*

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

1. To draw a two-pixel thick border around a prompt string, the example C language program **prompt.c** uses the **linewidth** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

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 Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

lmbind Subroutine

Purpose

Makes a material, light, or lighting model definition active.

Library

Graphics Library (**libgl.a**)

C Syntax

void lmbind(Int16 *target*, **Int32** *index*)

FORTRAN Syntax

SUBROUTINE LMBIND(*target*, *index*)
INTEGER*2 *target*
INTEGER*4 *index*

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

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:

MATERIAL 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, or 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 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

1. To use the materials, lights, and lighting model defined with the **lmdef** subroutine, the example C language program **cylinder2.c** uses the **lmbind** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Creating Lighting Effects, and Setting Pipeline Options in *Graphics Programming Concepts*.

lmcolor Subroutine

Purpose

Changes the target of the color commands while lighting is active.

Library

Graphics Library (**libgl.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: This subroutine cannot be used to add to a display list.

Parameters

mode Specifies the name of the mode to be used in conjunction with RGB color subroutines. Possible modes are:

LMC_COLOR	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	Color subroutines set the current EMISSION color property of the current material.
LMC_AMBIENT	Color subroutines set the current AMBIENT color property of the current material.
LMC_DIFFUSE	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	Color subroutines set the current SPECULAR color property of the current material.

LMC_AD	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	RGB subroutines are ignored.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Creating Lighting Effects, Setting Pipeline Options, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

lmdef Subroutine

Purpose

Defines a new material, light, or lighting model.

Library

Graphics Library (**libgl.a**)

C Syntax

void lmdef
(Int16 *deftype*, **Int32** *index*,
Int16 *numpoints*, **Float32** *properties*[ ])

FORTRAN Syntax

SUBROUTINE LMDEF(*deftype*, *index*, *numpoints*, *properties***)**
INTEGER*2 *deftype numpoints*
INTEGER*4 *index*
REAL *properties*(*numpoints*)

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 contain predefined definitions. These predefined definitions have set all properties to their default values and cannot be changed. Their values are as follows:

Definition/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 the greatest 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 **lmdef** 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

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

The default lighting model values are as follows:

AMBIENT	0.2, 0.2, 0.2
LOCALVIEWER	0.0
ATTENUATION	1.0, 0.0

Note: 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:

- DEFMATERIAL — Indicates that this routine defines the properties of a material.
- DEFLIGHT — Indicates that this routine defines the properties of a light source.
- DEFLMODEL — 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. Altogether, there are 11 defined symbolic constants (command tokens) that identify the properties of a definition. The valid symbolic constants for the *properties* parameter are as follows:

- EMISSION — 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 — 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. Assignments are as follows:

If the definition type is...	then AMBIENT assigns...
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 — 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 — 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 — Assigns the material specular scattering exponent of the material. Following this symbolic constant, enter a whole number between 0.0 and 128.0 inclusive. The higher the value, the smoother the surface appearance and the more focused the specular highlight.

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 — Assigns that 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 — 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.

LOCALVIEWER 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 Assigns the light attenuation factor for the lighting model (scene). Following this symbolic constant, enter two attenuation factor values to specify:

k-of Attenuation offset factor.

k-rate Attenuation rate factor.

Example

1. To define the properties of two materials and two lights, and to define a lighting model, the example C language program **cylinder2.c** uses the **lmdef** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Creating Lighting Effects, Setting Pipeline Options, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

loadmatrix Subroutine

Purpose

Loads a transformation matrix.

Library

Graphics Library (**libgl.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.

Parameter

matrix Specifies the matrix to be loaded onto the matrix stack.

Example

1. To load the modelling/viewing matrix with the identity matrix, the example C language program **localatten.c** uses the **loadmatrix** subroutine after changing the matrix mode to **MVIEWING** with the **mmode** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

loadname Subroutine

Purpose

Loads a name onto the name stack.

Library

Graphics Library (**libgl.a**)

C Syntax

void loadname(Int16 *name***)**

FORTRAN Syntax

SUBROUTINE LOADNA(*name***)**
INTEGER*2 *name*

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

1. To place a name on the top of the name stack, the example C language program **pick1.c** calls the **loadname** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

loadXfont Subroutine

Purpose

Loads an Enhanced X-Windows font into the font table.

Library

Graphics Library (**libgl.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 and **XListFontsWithInfo** subroutine. The example program on loading X fonts shows an example of the use of the **XListFonts** subroutine.

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

1. To add a font to the fonts defined for a process, the example C language program **xfonts.c** uses the **loadXfont** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Creating Text Characters, Getting Ready to Run GL, and Starting GL Functions in *Graphics Programming Concepts*.

AIXwindows Overview and Enhanced X-Windows Introduction in *User Interface Programming Concepts*.

logicop Subroutine

Purpose

Specifies a logical operation for pixel writes.

Library

Graphics Library (**libgl.a**)

C Syntax

void logicop(Int32 *opcode***)**

FORTRAN Syntax

SUBROUTINE LOGICO(*opcode***)**
INTEGER*4 *opcode*

Description

The **logicop** subroutine specifies the bit-wise logical operation for pixel writes. The logical 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.

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. The **blendfunction** subroutine does not function in systems without alpha bitplanes.

Parameter

opcode One of the 16 possible logical operations defined in the following table:

Value	Operation
LO_ZERO	0
LO_AND	src AND dst
LO_ANDR	src AND (NOT dst)
LO_SRC	src
LO_ANDI	(NOT src) AND dst
LO_DST	dst
LO_XOR	src XOR dst
LO_OR	src OR dst
LO_NOR	NOT (src OR dst)
LO_XNOR	NOT (src XOR dst)
LO_NDST	NOT dst
LO_ORR	src OR (NOT dst)
LO_NSRC	NOT src
LO_ORI	(NOT src) OR dst
LO_NAND	NOT (src AND dst)
LO_ONE	1

Note: The numeric assignment of the 16 operation names were chosen to be identical to those defined by the AIXwindows system.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Specifying the alpha blending ratio with the **blendfunction** subroutine.

GL Introduction, Configuring the Frame Buffer, and Controlling Frame Buffer Update in *Graphics Programming Concepts.*

lookat Subroutine

Purpose

Defines a viewing transformation.

Library

Graphics Library (**libgl.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***)**
INTEGER*4 *viewx*, *viewy*, *viewz*, *pointx*, *pointy*, *pointz*, *twist*

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, **lookat** 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 entirely 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

1. To define the point of view, the example C language program **zbuffer1.c** uses the **lookat** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Defining the viewer's position in polar coordinates with the **polarview** subroutine.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

IRGBrange Subroutine

Purpose

Sets the range of colors used for depth-cueing.

Library

Graphics Library (**libgl.a**)

C Syntax

void IRGBrange
(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*2 *rmin*, *gmin*, *bmin*, *rmax*, *gmax*, *bmax*

INTEGER*4 *znear*, *zfar*

Description

The **IRGBrange** 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 for *znear* and *zfar* is -0x800000 to +0x7FFFFF.

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.

File

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

Related Information

GL Introduction, Performing Depth-Cueing, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

lsetdepth Subroutine

Purpose

Sets up a 3-D viewport.

Library

Graphics Library (**libgl.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.

The valid values of the parameters range from –0x800000 to +0x7FFFFF.

Acceptable mappings include all those where the values of the *near* and *far* parameters are within the supported range, including mappings where *near* > *far*.

Note: 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.

Parameters

near	Specifies the screen coordinate of the near clipping plane.
far	Specifies the screen coordinate of the far clipping plane.

Example

1. To set the range of z-axis values to store in the bitplanes, the example C language program **depthcue.c** uses the **lsetdepth** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Turning depth-cueing on and off with the **depthcue** subroutine.

Setting the viewport depth range with the **lsetdepth** subroutine.

GL Introduction, Configuring the Frame Buffer for GL, Performing Depth-Cueing in GL, Working with Coordinate Systems in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

lshaderange Subroutine

Purpose

Sets the range of color indexes used in depth-cueing.

Library

Graphics Library (**libgl.a**)

C Syntax

void lshaderange
(Colorindex *lowindex*, **Colorindex** *highindex*,
Int32 *znear*, **Int32** *zfar*)

FORTRAN Syntax

SUBROUTINE LSHADE(*lowindex*, *highindex*, *znear*, *zfar***)**
INTEGER*2 *lowindex*, *highindex*
INTEGER*4 *znear*, *zfar*

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 values for the *znear* and *zfar* parameters range from –0x800000 to +0x7FFFFFF. The default is **lshaderange(0, 7,** *zmin*, *zmax***)** where the *zmin* parameter and *zmax* parameter are the minimum and maximum *z* values listed previously.

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

1. To map the z-axis values to a spread of colors in the color map, the example C language program **depthcue.c** uses the **lshaderange** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Turning depth-cueing on and off with the **depthcue** subroutine.

Setting the range of RGB colors used for depth-cueing with the **lRGBrange** subroutine.

Setting the viewport depth range with the **lsetdepth** subroutine.

GL Introduction, Performing Depth-Cueing, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

lsrepeat Subroutine

Purpose

Sets the repeat factor for the current linestyle.

Library

Graphics Library (**libgl.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.

Parameter

factor Multiplier of the linestyle pattern.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

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 Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

makeobj Subroutine

Purpose

Creates an object (display list).

Library

Graphics Library (**libgl.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

1. To specify the beginning of a display list defining a graphical object, the example C language program **depthcue.c** uses the **makeobj** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

maketag Subroutine

Purpose

Inserts a marker tag into a display list.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts.*

mapcolor Subroutine

Purpose

Changes a color map entry to a specified RGB value.

Library

Graphics Library (**libgl.a**)

C Syntax

void mapcolor
(Colorindex *index*,
Int16 *red*, **Int16** *green*, **Int16** *blue*)

FORTRAN Syntax

SUBROUTINE MAPCOL (*index*, *red*, *green*, *blue*)
INTEGER*2 *index*, *red*, *green*, *blue*

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)** 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

1. To edit the color map, the example C language program **colored.c** uses the **mapcolor** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 in *Graphics Programming Concepts*.

mapcolors Subroutine

Purpose

Loads a range of the color map.

Library

Graphics Library (**libgl.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*2 *start_idx*, *end_idx*
INTEGER*2 *r(1)*, *g(1)*, *b(1)*

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;
        do (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)** 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.

File

/usr/include/gl/gl.h	Contains 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 in *Graphics Programming Concepts*.

mapw Subroutine

Purpose

Computes the inverse mapping from screen coordinates to modeling coordinates.

Library

Graphics Library (**libgl.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*
INTEGER*2 *screenx*, *screeny*
REAL *modelx1*, *modely1*, *modelz1*, *modelx2*, *modely2*, *modelz2*

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Working with Coordinate Systems in GL, and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

mapw2 Subroutine

Purpose

Computes the inverse mapping from screen coordinates to 2-D modeling coordinates.

Library

Graphics Library (**libgl.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*
INTEGER*2 *screenx, screeny*
REAL *modelx, modely*

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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Working with Coordinate Systems in GL, and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

maxsize Subroutine

Purpose

Specifies the maximum size of a window.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts.*

minsize Subroutine

Purpose

Specifies the minimum size of a window.

Library

Graphics Library (**libgl.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.

Files

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

mmode Subroutine

Purpose

Sets the current matrix mode.

Library

Graphics Library (**libgl.a**)

C Syntax

void mmode(Int16 *mode***)**

FORTRAN Syntax

SUBROUTINE MMODE(*mode***)**
INTEGER*2 *mode*

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 in projection matrix mode, do not use the subroutines **pushmatrix** and **popmatrix**.

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.)

Notes:

1. This subroutine can be called within primitives.
2. This subroutine cannot be used to add to a display list.

Parameter

mode Specifies the current matrix mode to be set.

There are three possible values for this function:

Mode Name	Mode
MSINGLE	Single matrix
MPROJECTION	Projection matrix
MVIEWING	Viewing matrix

Example

1. To specify that matrix operations manipulate only the modeling and viewing matrix (not the projection matrix), the example C language program **cylinder1.c** uses the **mmode** subroutine with MVIEWING as the value of the *mode* parameter.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Creating Lighting Effects, Setting Pipeline Options, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

move Subroutine

Purpose

Moves the current graphics position to a specified point.

Library

Graphics Library (**libgl.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*

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

1. To move the graphics position without drawing any lines, the example C language program **depthcue.c** uses the **movei** subroutine when defining a graphical object.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing with Move-Draw Style Subroutines in *Graphics Programming Concepts*.

multimap Subroutine

Purpose

Organizes the color map as 16 small maps.

Library

Graphics Library (**libgl.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.

The reorganization into multiple independent maps does not take effect until the **gconfig** subroutine is called.

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

File

/usr/include/gl/gl.h Contains 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 in *Graphics Programming Concepts.*

multmatrix Subroutine

Purpose

Premultiplies the current transformation matrix.

Library

Graphics Library (**libgl.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.

Parameter

matrix	Specifies the matrix that is to multiply the current top matrix of the matrix stack.

Example

1. To set up the transformation matrix to draw a Bezier curve, the example C language program **curve3.c** calls the **multmatrix** subroutine with the control point matrix, the Bezier basis matrix, and the precision matrix.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts.*

n3f Subroutine

Purpose

Specifies a normal vector for lighting calculations.

Library

Graphics Library (**libgl.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 except one: **normal** can be used for display lists and **n3f** cannot.

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

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

1. To specify the normal vectors of a polygon, the example C language program **cylinder2.c** calls the **n3f** subroutine between the **bgnpolygon** subroutine and the **endpolygon** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Drawing with Begin–End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

newpup Subroutine

Purpose

Allocates and initializes a structure for a new menu.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating and Managing Pop-Up Menus in GL in *Graphics Programming Concepts*.

newtag Subroutine

Purpose

Inserts a tag at an offset from an existing tag.

Library

Graphics Library (**libgl.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.

File

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

Related Information

Marking a location in a display list with the **maketag** subroutine.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

noborder Subroutine

Purpose

Specifies a window without any borders.

Library

Graphics Library (**libgl.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.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

noise Subroutine

Purpose

Filters valuator motion.

Library

Graphics Library (**libgl.a**)

C Syntax

void noise(Device *valuator*, **Int16** *delta*)

FORTRAN Syntax

SUBROUTINE NOISE(*valuator*, *delta*)
INTEGER*2 *valuator*, *delta*

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.

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.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

Related Information

Assigning an initial value to a valuator with the **setvaluator** subroutine.

GL Introduction, Controlling Queues and Devices in GL, and Controlling the Keyboard in GL in *Graphics Programming Concepts*.

noport Subroutine

Purpose

Specifies that a program does not require a window.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

normal Subroutine

Purpose

Sets the current normal vector.

Library

Graphics Library (**libgl.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 except one: **normal** can be used for display lists and **n3f** cannot.

Notes:

1. This subroutine can be called within primitives.
2. This subroutine cannot be used to add to a display list.

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 of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Drawing with Begin–End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

nurbscurve Subroutine

Purpose

Controls the shape of a NURBS trimming curve.

Library

Graphics Library (**libgl.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:

$$*\,(ctlarray + n \times stride)$$

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 N_ST and N_STW, corresponding to two dimensional and three dimensional (rational) control points.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing NURBS Curves and Surfaces in *Graphics Programming Concepts.*

nurbssurface Subroutine

Purpose

Controls the shape of a NURBS surface

Library

Graphics Library (**libgl.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 N_XYZ and N_XYZW, corresponding to three-dimensional and four-dimensional (rational) control points.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing NURBS Curves and Surfaces in *Graphics Programming Concepts*.

objdelete Subroutine

Purpose

Deletes subroutines from a display list.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

objinsert Subroutine

Purpose

Inserts routines in an object at a specified location.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

objreplace Subroutine

Purpose

Replaces existing display list routines with new ones.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Creating Objects (Display Lists) in *Graphics Programming Concepts*.

onemap Subroutine

Purpose

Organizes the color map as one large map.

Library

Graphics Library (**libgl.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 colormap 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 reorganization of the color map into one large map does not take effect until the **gconfig** subroutine is called. The system is initially in onemap 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.

File

/usr/include/gl/gl.h Contains 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 in *Graphics Programming Concepts*.

ortho or ortho2 Subroutine

Purpose

Defines an orthographic transformation.

Library

Graphics Library (**libgl.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 $-1 \leq z \leq 1$ 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 $-w \leq x, y, z \leq +w$. 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. To define the two–dimensional world coordinates that exactly fit the defined viewport, the example C language program **paint.c** uses the **ortho2** subroutine.
2. To map a rectangular volume of space to the viewport, the example C language program **backface.c** uses the **ortho** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

overlay Subroutine

Purpose

Sets the number of user-defined bitplanes used for overlay drawing.

Library

Graphics Library (**libgl.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, 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 non-zero. 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.

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.

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

Parameter

planes Specifies the number of bitplanes to use for overlay drawing.

Example

1. To set the number of overlay bitplanes to two (2), the example C language program **ovrlay.c** uses the **overlay** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Configuring the Frame Buffer, and Controlling Frame Buffer Update in *Graphics Programming Concepts*.

patch Subroutine

Purpose

Draws a surface patch.

Library

Graphics Library (**libgl.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

1. To draw three surface patches, the example C language program **patch1.c** uses the **patch** subroutine after defining the settings for each patch.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts*.

patchbasis Subroutine

Purpose

Sets the current spline surface basis matrices.

Library

Graphics Library (**libgl.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

1. To set the basis matrix for drawing the curves that represent each surface patch, the example C language program **patch1.c** calls the **patchbasis** subroutine before calling the **patch** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts.*

patchcurves Subroutine

Purpose

Sets the number of curves used to represent a patch.

Library

Graphics Library (**libgl.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

1. To define to the **patch** subroutine the number of curves to use in drawing three surface patches, the example C language program **patch1.c** uses the **patchcurves** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts*.

patchprecision Subroutine

Purpose

Sets the precision at which curves are drawn in a patch.

Library

Graphics Library (**libgl.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

1. To set the precision for drawing the curves that represent surface patches, the example C language program **patch1.c** calls the **patchprecision** subroutine before calling the **patch** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts.*

pclos Subroutine

Purpose

Closes a filled polygon.

Library

Graphics Library (**libgl.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 AIX subroutine **pclose,** which closes an AIX pipe, in the **libc.a** library.

Example

1. To specify the end of a filled polygon definition, the example C language program **zbuffer1.c** uses the **pclos** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Drawing with Move-Draw Style Subroutines, and Setting Attributes in *Graphics Programming Concepts*.

pdr Subroutine

Purpose

Specifies the next point in a filled polygon.

Library

Graphics Library (**libgl.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*

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

1. To draw the edges of a filled polygon, the example C language program **zbuffer1.c** uses the **pdr** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Drawing with Move-Draw Style Subroutines, and Setting Attributes in *Graphics Programming Concepts*.

perspective Subroutine

Purpose

Defines a perspective projection transformation.

Library

Graphics Library (**libgl.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

1. To load a projection transformation as the current transformation matrix, the example C language program **zbuffer1.c** uses the **perspective** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts.*

pick Subroutine

Purpose

Places the system in picking mode.

Library

Graphics Library (**libgl.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 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 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.

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/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

1. To pick objects, the example C language program **pick1.c** calls the **pick** subroutine when the left mouse button is pressed.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Using Viewports and Screenmasks in GL, and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

picksize Subroutine

Purpose

Sets the dimensions of the picking region.

Library

Graphics Library (**libgl.a**)

C Syntax

void picksize(Int16 *deltax*, **Int16** *deltay*)

FORTRAN Syntax

SUBROUTINE PICKSI(*deltax*, *deltay*)
INTEGER*2 *deltax*, *deltay*

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.

File

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

Related Information

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

GL Introduction and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

pmv Subroutine

Purpose

Moves to the starting point for a filled polygon.

Library

Graphics Library (**libgl.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*

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

1. To move to the beginning position for drawing a filled polygon, the example C language program **zbuffer1.c** uses the **pmv** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Drawing with Move-Draw Style Subroutines, Performing Depth-Cueing, Setting Attributes, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

pnt Subroutine

Purpose

Draws a point in modeling coordinates.

Library

Graphics Library (**libgl.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*

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

1. To draw random points, the example C language program **depthcue.c** uses the **pnt** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Drawing with Move-Draw Style Subroutines, Performing Depth-Cueing, Setting Attributes, and Working with Coordinate Systems in *Graphics Programming Concepts*.

pntsmooth Subroutine

Purpose

Turns point antialiasing on and off.

Library

Graphics Library (**libgl.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 mode. The pntsmooth hardware replaces the least significant 4 bits of the current color index with bits that represent pixel coverage. Therefore, a 16–entry block of the color map (whose lowest entry is a multiple of 16) must be initialized as a ramp between the background color (lowest index) to the line color (highest index).

Before drawing the points, clear the area to the background color using the **poly** or **clear** subroutine. If you define many such ramps, you can draw antialiased points with different colors and intensities by changing the current color index (only the upper bits are significant). You can draw depth–cued, antialiased points in this manner.

The **zsource** and **zfunction** subroutines can be used with the **pntsmooth** subroutine for depth or color values. When the **zsource** subroutine is used with **ZSRC_COLOR**, intersecting points behave more correctly.

The **pntsmooth** subroutine does not support subpixel positioning of line vertices.

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

Parameter

mode	SMP_ON	Turn on point–antialiasing capabilities.
	SMP_OFF	Turn off point–antialiasing capabilities.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

Configuring the Frame Buffer, Smoothing Jagged Lines with Antialiasing, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

polarview Subroutine

Purpose

Defines the viewer's position in polar coordinates.

Library

Graphics Library (**libgl.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.

File

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

Related Information

Defining a viewing transformation with the **lookat** subroutine.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

polf Subroutine

Purpose

Draws a filled polygon.

Library

Graphics Library (**libgl.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

1. To draw an ice cream cone, the example C language program **tpbig.c** uses the **polf2i** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 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.

GLIntroduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

poly Subroutine

Purpose

Draws a polygon.

Library

Graphics Library (**libgl.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

1. To outline an ice cream cone in black, the example C language program **tpbig.c** uses the **poly2i** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GLIntroduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

popattributes Subroutine

Purpose

Pops the attribute stack.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

Setting Attributes and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

popmatrix Subroutine

Purpose

Pops the transformation matrix stack.

Library

Graphics Library (**libgl.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

1. To restore the previous transformation matrix after altering it with the **translate** modeling subroutine, the example C language program **backface.c** uses the **popmatrix** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

popname Subroutine

Purpose

Pops a name off the name stack.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

popviewport Subroutine

Purpose

Pops the viewport stack.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

prefposition Subroutine

Purpose

Constrains the location and size of a window.

Library

Graphics Library (**libgl.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 constrains the location and size, in pixels, of a window. With the **prefposition** subroutine, the applications programmer can prevent the user from moving or 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 the constraints from the window. Note that doing this nullifies all constraints, not just those of position and 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 rubber band outline is not shown.

If the **prefposition** subroutine call is followed by a call to the **winconstraints** subroutine, the current window is resized and repositioned. The repositioning occurs at the time of the **winconstraints** call; the **prefposition** subroutine has no effect on the current window until that time.

Constraining the position of a window is heavily discouraged in a multi-application windowing environment because it creates conflicts with other applications. This statement applies only to position; constraining the size of a window is acceptable.

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

1. To specify the window's original location and size, the example C language program **colored.c** uses the **prefposition** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 the 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 in GL in *Graphics Programming Concepts*.

prefsize Subroutine

Purpose

Constrains the size of a window.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

Creating and Managing Windows in GL in *Graphics Programming Concepts*.

pushattributes Subroutine

Purpose

Pushes down the attribute stack.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

Setting Attributes and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

pushmatrix Subroutine

Purpose

Pushes down the transformation matrix stack.

Library

Graphics Library (**libgl.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

1. To save the current transformation matrix before altering it with the **translate** modeling subroutine, the example C language program **backface.c** uses the **pushmatrix** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

pushname Subroutine

Purpose

Pushes a new name on the name stack.

Library

Graphics Library (**libgl.a**)

C Syntax

void pushname(Int16 *name***)**

FORTRAN Syntax

SUBROUTINE PUSHNA(*name***)**
INTEGER*2 *name*

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

1. To push the name stack and put a new name on the top, the example C language program **pick1.c** calls the **pushname** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Picking and Selecting Overview for GL in *Graphics Programming Concepts*.

pushviewport Subroutine

Purpose

Duplicates the current viewport.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

pwlcurve Subroutine

Purpose

Describes a piecewise linear trimming curve for NURBS surfaces.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing NURBS Curves and Surfaces in *Graphics Programming Concepts*.

qdevice Subroutine

Purpose

Enables a input device (keyboard, button, or valuator) for event queuing.

Library

Graphics Library (**libgl.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

1. To enable input from various devices, the example C language program **scrn_rotate.c** uses the **qdevice** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

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.

GL Introduction, Controlling Queues and Devices in GL, and Controlling the Keyboard in GL in *Graphics Programming Concepts*.

qenter Subroutine

Purpose

Creates an event queue entry.

Library

Graphics Library (**libgl.a**)

C Syntax

void qenter(Int16 *qtype*, **Int16** *value*)

FORTRAN Syntax

SUBROUTINE QENTER(*qtype*, *value*)
INTEGER*2 *qtype*, *value*

Description

The **qenter** subroutine takes two 16-bit integers, the *qtype* and *value* parameters, and enters them into the event queue. There is no way to distinguish user-defined from system-defined entries unless disjointed sets of device numbers are used.

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

1. To enter a REDRAW device event into the queue, the example C language program **scrn_rotate.c** uses the **qenter** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

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.

GL Introduction and Controlling Queues and Devices in GL in *Graphics Programming Concepts*.

qread Subroutine

Purpose

Reads the first entry in the event queue.

Library

Graphics Library (**libgl.a**)

C Syntax

Int32 qread(Int16 *data***)**

FORTRAN Syntax

INTEGER*4 FUNCTION QREAD(*data***)**
INTEGER*2 *data*

Description

The **qread** subroutine, when there is an entry in the event queue, returns the device number of the queue entry, writes the data of the entry into the *data* parameter, and removes the entry from the queue. If there is not an entry in the queue, the **qread** subroutine executes when an entry is made.

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

1. To read input from the event queue, the example C language program **scrn_rotate.c** uses the **qread** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

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.

GL Introduction and Controlling Queues and Devices in GL in *Graphics Programming Concepts*.

qreset Subroutine

Purpose

Empties the event queue.

Library

Graphics Library (**libgl.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

1. To delete all input events from any devices, the example C language program **scrn_rotate.c** uses the **qreset** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

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.

GL Introduction and Controlling Queues and Devices in GL in *Graphics Programming Concepts*.

qtest Subroutine

Purpose

Checks the contents of the event queue.

Library

Graphics Library (**libgl.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

1. To check if there is any input from the queued devices, the example C language program **scrn_rotate.c** uses the **qtest** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

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.

GL Introduction and Controlling Queues and Devices in GL in *Graphics Programming Concepts*.

rcrv Subroutine

Purpose

Draws a rational cubic spline curve.

Library

Graphics Library (**libgl.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.

Parameter

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts*.

rcrvn Subroutine

Purpose

Draws a series of rational curve segments.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts*.

rdr Subroutine

Purpose

Draws a line relative to the current graphics point.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Drawing with Move-Draw Style Subroutines, Performing Depth-Cueing, Setting Attributes, and Working with Coordinate Systems in *Graphics Programming Concepts*.

readpixels Subroutine

Purpose

Returns values of specific pixels in color map mode.

Library

Graphics Library (**libgl.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. Do not use the **readpixels** subroutine in new development.

Notes:

1. This subroutine is available only in color map mode.
2. This subroutine cannot be used to add to a display list.

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 Values

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Reading and Writing Pixels in GL, Using Viewports and Screenmasks in GL, and Working in Color Map and RGB Modes in GL in *Graphics Programming Concepts*.

readRGB Subroutine

Purpose

Returns values of specific pixels in RGB mode.

Library

Graphics Library (**libgl.a**)

C Syntax

Int32 readRGB
(Int16 *number*,
RGBvalue *red*[ ], **RGBvalue** *green*[ ], **RGBvalue** *blue*[ ])

FORTRAN Syntax

INTEGER*4 FUNCTION READRG(*number***,** *red*, *green*, *blue)*
INTEGER*2 *number*
CHARACTER*(*) *red*, *green*, *blue*;

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. Do not use the **readRGB** subroutine in new development.

Notes:

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.

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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Reading and Writing Pixels in GL, Using Viewports and Screenmasks in GL, Configuring the Frame Buffer for GL, Creating Animated Screens in GL, and Working in Color Map and RGB Modes in GL in *Graphics Programming Concepts*.

readsource Subroutine

Purpose

Specifies the source for pixels to be read by various subroutines.

Library

Graphics Library (**libgl.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 four defined values for this parameter are listed in the following table:

Values for the source Parameter	
Value	**Function**
SRC_AUTO	Selects the front buffer in single buffer mode and the back buffer in double buffer mode.
SRC_FRONT	Front color buffer. Valid for both single and double buffer operation.
SRC_BACK	Back color buffer. Valid during double buffer operation only.
SRC_ZBUFFER	Z-buffer. Valid during both single and double buffer operation.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Reading and Writing Pixels in GL, and Configuring the Frame Buffer for GL in *Graphics Programming Concepts*.

rect Subroutine

Purpose

Draws an unfilled rectangle.

Library

Graphics Library (**libgl.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*

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

1. To draw the outline of a rectangle in green, the example C language program **tpbig.c** uses the **recti** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

Drawing a filled rectangle with the **rectf** subroutine.

GLIntroduction, Setting Attributes, and Using the GL High–Level Drawing Library in *Graphics Programming Concepts*.

rectcopy Subroutine

Purpose

Copies a rectangle of pixels with an optional zoom.

Library

Graphics Library (**libgl.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*2 *xll*, *yll*, *xur*, *yur*, *newx*, *newy*

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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Reading and Writing Pixels in GL in *Graphics Programming Concepts*.

rectf Subroutine

Purpose

Draws a filled rectangle.

Library

Graphics Library (**libgl.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*

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

1. To draw a filled, red rectangle, the example C language program **tpbig.c** uses the **rectfi** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

Drawing a rectangle with the **rect** subroutine.

GL Introduction, Setting Attributes, and Using the GL High–Level Drawing Library in *Graphics Programming Concepts*.

rectread or lrectread Subroutine

Purpose

Reads a rectangular array of pixels into host memory.

Library

Graphics Library (**libgl.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*2 *xll*, *yll*, *xur*, *yur*, *parray***(1)**

INTEGER*4 LRECTR(*xll*, *yll*, *xur*, *yur*, *parray***)**
INTEGER*2 *xll*, *yll*, *xur*, *yur*
INTEGER*4 *parray***(1)**

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, not the screen or viewport.

The **readsource** subroutine specifies the pixel source from which the pixels are read.

Notes:

1. Both the **rectread** and **lrectread** subroutines leave the current character position unpredictable.
2. This subroutine 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, not 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.

Example

1. To get the color index value for a single pixel, the example C language program **paint.c** uses the **rectread** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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.

Drawing a rectangular array of pixels into the frame buffer with the **rectwrite** or **lrectwrite** subroutine.

GL Introduction, Reading and Writing Pixels in GL, Using Viewports and Screenmasks in GL, and Working in Color Map and RGB Modes in GL in *Graphics Programming Concepts*.

rectwrite or lrectwrite Subroutine

Purpose

Draws a rectangular array of pixels into the frame buffer.

Library

Graphics Library (**libgl.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*2 *xll*, *yll*, *xur*, *yur*, *parray***(1)**

SUBROUTINE LRECTW(*xll*, *yll*, *xur*, *yur*, *parray***)**
INTEGER*2 *xll*, *yll*, *xur*, *yur*
INTEGER*4 *parray***(1)**

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 colormap, 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.

Note that 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).

Note: This subroutine 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) • (yur-yll+1). For RGBA values, pack the bits in the form 0x*AABBGGRR*, where:

AA Contains the alpha value.

BB Contains the blue value.

GG Contains the green value.

RR Contains the red value.

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.

File

/usr/include/gl/gl.h Contains 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** or **lrectread** subroutine.

Specifying a zoom factor for rectangle copies and writes with the **rectzoom** subroutine.

GL Introduction, Reading and Writing Pixels in GL, Configuring the Frame Buffer for G, and Working in Color Map and RGB Modes in GL in *Graphics Programming Concepts*.

rectzoom Subroutine

Purpose

Specifies a zoom factor for rectangle copies and writes.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Reading and Writing Pixels in GL in *Graphics Programming Concepts.*

reshapeviewport Subroutine

Purpose

Sets the viewport to the dimensions of the current window.

Library

Graphics Library (**libgl.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

1. To reshape the viewport on a REDRAW event, the example C language program **scrn_rotate.c** uses the **reshapeviewport** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

RGBcolor Subroutine

Purpose

Sets the current color in RGB mode.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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 Attributes, Understanding the Hardware Used by GL, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

RGBmode Subroutine

Purpose

Sets a display mode that bypasses the color map.

Library

Graphics Library (**libgl.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 controls 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

The **RGBmode** subroutine is useful for monitors with 12 or more bitplanes.

Example

1. To perform lighting calculations, the example C language program **localatten.c** puts the system in RGB mode with the **RGBmode** subroutine. This lets the system calculate the correct colors for realistic shading.

File

/usr/include/gl/gl.h Contains 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 in *Graphics Programming Concepts.*

RGBwritemask Subroutine

Purpose

Grants write access to a subset of available bitplanes.

Library

Graphics Library (**libgl.a**)

C Syntax

void RGBwritemask
(Int16 *red*,
Int16 *green*,
Int16 *blue*)

FORTRAN Syntax

SUBROUTINE RGBWRI(*red, green, blue***)**
INTEGER*2 *red, green, blue*

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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

ringbell Subroutine

Purpose

Rings the keyboard bell.

Library

Graphics Library (**libgl.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

1. To ring the bell as a ball moves, the example C language program **ovrlay.c** uses the **ringbell** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Controlling the Keyboard in *Graphics Programming Concepts.*

rmv Subroutine

Purpose

Moves the graphics position relative to the current point.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Drawing with Move-Draw Style Subroutines in *Graphics Programming Concepts*.

rot Subroutine

Purpose

Rotates graphical primitives (floating-point version).

Library

Graphics Library (**libgl.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

1. To rotate a cylinder about the y- and z-axes, the example C language program **cylinder1.c** uses the **rot** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

rotate Subroutine

Purpose

Rotates graphical primitives (fixed-point version).

Library

Graphics Library (**libgl.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

1. To model the sides of a cube using a square, the example C language program **backface.c** uses the **rotate** modeling subroutine to alter the current transformation matrix.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts.*

rpatch Subroutine

Purpose

Draws a rational surface patch.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing Wire Frame Curves and Surface Patches in GL in *Graphics Programming Concepts*.

rpdr Subroutine

Purpose

Performs a relative polygon draw.

Library

Graphics Library (**libgl.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*

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Drawing with Move-Draw Style Subroutines, and Setting Attributes in *Graphics Programming Concepts*.

rpmv Subroutine

Purpose

Performs a relative move to the starting point of a filled polygon.

Library

Graphics Library (**libgl.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*

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Drawing with Move-Draw Style Subroutines, and Setting Attributes in *Graphics Programming Concepts*.

sbox, sboxi, or sboxs Subroutine

Purpose

Draw a screen-aligned rectangle.

Library

Graphics Library (**libgl.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*

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.

File

/usr/include/gl/gl.h Contains 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.

GLIntroduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

sboxf, sboxfi, or sboxfs Subroutine

Purpose

Draw a filled screen-aligned rectangle.

Library

Graphics Library (**libgl.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*

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 do 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.

File

/usr/include/gl/gl.h	Contains 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.

GLIntroduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts.*

scale Subroutine

Purpose

Scales and mirrors drawing primitives.

Library

Graphics Library (**libgl.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.

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

1. To draw two differently sized cylinders, the example C language program **cylinder2.c** draws one cylinder, then uses the **scale** subroutine before drawing the second cylinder.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

screenspace Subroutine

Purpose

Makes a program interpret graphics positions as absolute screen coordinates.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Working with Coordinate Systems in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

scrmask Subroutine

Purpose

Defines a a rectangular 2-D clipping mask.

Library

Graphics Library (**libgl.a**)

C Syntax

void scrmask
**(Screencoord ****left*,
**Screencoord ****right*,
**Screencoord ****bottom*,
**Screencoord ****top*)

FORTRAN Syntax

SUBROUTINE SCRMAS(*left*, *right*, *bottom*, *top*)
INTEGER*2 *left*, *right*, *bottom*, *top*

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

1. To define a new screenmask, the example C language program **prompt.c** uses the **scrmask** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Working with Coordinate Systems in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

setbell Subroutine

Purpose

Sets the duration of the keyboard bell sound.

Library

Graphics Library (**libgl.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

1. To set the duration of the bell sound for a short beep, the example C language program **ovrlay.c** uses the **setbell** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Controlling the Keyboard in *Graphics Programming Concepts.*

setcursor Subroutine

Purpose

Sets the cursor characteristics.

Library

Graphics Library (**libgl.a**)

C Syntax

void setcursor
(Int16 *index*, **Colorindex** *color*, **Colorindex** *writemask*)

FORTRAN Syntax

SUBROUTINE SETCUR(*index*, *color*, *writemask*)
INTEGER*2 *index*, *color*, *writemask*

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.

File

/usr/include/gl/gl.h	Contains 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.

Creating a Cursor in GL and Creating and Managing Windows in GL in *Graphics Programming Concepts*.

setdblights Subroutine

Purpose

Sets the lights on the dial and switch box.

Library

Graphics Library (**libgl.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.

File

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

Related Information

GL Introduction and Controlling the Keyboard in *Graphics Programming Concepts.*

setlinestyle Subroutine

Purpose

Selects a linestyle.

Library

Graphics Library (**libgl.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.

Parameter

index Specifies an index into the table of linestyles built by the **deflinestyle** subroutine

Example

1. To use a linestyle previously defined by the **deflinestyle** subroutine, the example C language program **colored.c** uses the **setlinestyle** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 NURBS Curves and Surfaces, Drawing Wire Frame Curves and Surface Patches, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Setting Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts.*

setmap Subroutine

Purpose

Selects one of 16 small color maps in multimap mode.

Library

Graphics Library (**libgl.a**)

C Syntax

void setmap(Int16 *mapnum*)

FORTRAN Syntax

subroutine setmap(*mapnum*)
integer*2 *mapnum*

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. Note that 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.

File

/usr/include/gl/gl.h Contains 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 in *Graphics Programming Concepts*.

setnurbsproperty Subroutine

Purpose

Sets a property for the display of trimmed NURBS surfaces.

Library

Graphics Library (**libgl.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.

N_ERRORCHECKING	If value is 1.0, some error checking is enabled. If error checking is disabled, the system runs slightly faster. The default value is 0.0.
N_PIXEL_TOLERANCE	The value is the maximum length, in pixels, of edges of polygons on the screen used to render trimmed NURBS surfaces. The default value is 50.0 pixels.

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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Drawing NURBS Curves and Surfaces in *Graphics Programming Concepts*.

setpattern Subroutine

Purpose

Selects a pattern for filling polygons and rectangles.

Library

Graphics Library (**libgl.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.

Parameter

index Specifies the index into the table of defined patterns.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 NURBS Curves and Surfaces, Drawing Wire Frame Curves and Surface Patches, Drawing with Begin-End Style Subroutines, Drawing with Move-Draw Style Subroutines, Setting Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

setpup Subroutine

Purpose

Enables or disables a given pop-up menu entry.

Library

Graphics Library (**libgl.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:
	PUP_NONE enables a menu entry.
	PUP_GREY disables a menu entry.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Creating and Managing Pop-Up Menus in GL in *Graphics Programming Concepts.*

setvaluator Subroutine

Purpose

Assigns an initial value to a valuator.

Library

Graphics Library (**libgl.a**)

C Syntax

void setvaluator
(Device *val*,
Int16 *init*, **Int16** *min*, **Int16** *max*)

FORTRAN Syntax

SUBROUTINE SETVAL(*val*, *init*, *min*, *max***)**
INTEGER*2 *val*, *init*, *min*, *max*

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.

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.

File

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

Related Information

Returning the current state of a valuator with the **getvaluator** subroutine.

GL Introduction, Controlling the Keyboard in GL, and Controlling Queues and Devices in GL in *Graphics Programming Concepts*.

shademodel Subroutine

Purpose

Selects the shading style used to draw filled polygons.

Library

Graphics Library (**libgl.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.

Parameter

mode Specifies one of two possible flags:

FLAT Instructs the system to draw filled polygons in a constant color.

GOURAUD Instructs the system to draw filled polygons with Gouraud shading. (This is the default shading model.)

Example

1. To render the cube faster, the example C language program **backface.c** calls the **shademodel** subroutine with the value of the *mode* parameter set to **FLAT.** This shades each side of the cube in a constant color.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

Returning the shading model used to draw polygons with the **getsm** 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, Setting Attributes, Understanding the Hardware Used by GL, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts.*

singlebuffer Subroutine

Purpose

Invokes single buffer mode.

Library

Graphics Library (**libgl.a**)

C Syntax

void singlebuffer( )

FORTRAN Syntax

SUBROUTINE SINGLE

Description

The **singlebuffer** subroutine invokes single buffer mode, in which the system simultaneously updates and displays the image data in the active bitplanes. Consequently incomplete or changing pictures can appear 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.

File

/usr/include/gl/gl.h Contains 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.

Configuring the Frame Buffer, Creating Animated Screens, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

splf Subroutine

Purpose

Draws a shaded filled polygon.

Library

Graphics Library (**libgl.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.

Notes:

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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Setting Attributes, and Using the GL High-Level Drawing Library in *Graphics Programming Concepts*.

stepunit Subroutine

Purpose

Specifies that a window change size in discrete steps.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

strwidth Subroutine

Purpose

Returns the width of the specified text string.

Library

Graphics Library (**libgl.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

1. To get the number of pixels needed to draw a prompt string, the example C language program **prompt.c** uses the **strwidth** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

This subroutine is not available for Japanese Language Support.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Creating Text Characters in GL, Picking and Selecting Overview for GL, Working with Coordinate Systems in GL, and Querying the System in GL in *Graphics Programming Concepts*.

subpixel Subroutine

Purpose

Controls the placement of point, line, and polygon vertices

Library

Graphics Library (**libgl.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 — Specifies a value for the screen coordinates. The settings for the *bool* paramter are:

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.

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.

File

/usr/include/gl/gl.h Contains 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.

Configuring the Frame Buffer, Smoothing Jagged Lines with Antialiasing, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

swapbuffers Subroutine

Purpose

Exchanges the front and back buffers in double buffer mode.

Library

Graphics Library (**libgl.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

1. To display a cube after drawing it in the back buffer, the example C language program **backface.c** uses the **swapbuffers** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

Configuring the Frame Buffer, Creating Animated Screens, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

swapinterval Subroutine

Purpose

Defines a minimum time between buffer swaps.

Library

Graphics Library (**libgl.a**)

C Syntax

void swapinterval(Int16 *interval* **)**

FORTRAN Syntax

SUBROUTINE SWAPINT(*interval***)**
INTEGER*2 *interval*

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.

File

/usr/include/gl/gl.h	Contains 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 **swapbuffer** subroutine.

Configuring the Frame Buffer, Creating Animated Screens, and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

swaptmesh Subroutine

Purpose

Toggles the triangle mesh register pointer.

Library

Graphics Library (**libgl.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).

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

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Drawing with Begin-End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

swinopen Subroutine

Purpose

Creates a restricted subwindow.

Library

Graphics Library (**libgl.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.

File

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

Related Information

Creating a new window with the **winopen** subroutine.

Creating and Managing Windows in GL in *Graphics Programming Concepts*.

textport Subroutine

Purpose

Allocates an area of the screen for the textport.

Library

Graphics Library (**libgl.a**)

C Syntax

void textport
(Screencoord *left*, **Screencoord** *right*,
Screencoord *bottom*, **Screencoord** *top*)

FORTRAN Syntax

SUBROUTINE TEXTPO(*left*, *right*, *bottom*, *top***)**
INTEGER*2 *left*, *right*, *bottom*, *top*

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

1. To define a textport, the example C language program **tpbig.c** uses the **textport** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Working with the Textport in GL in *Graphics Programming Concepts.*

tie Subroutine

Purpose

Ties two valuators to a button.

Library

Graphics Library (**libgl.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

1. To enter the current mouse coordinates into the event queue whenever the left or middle mouse buttons are pressed, the example C language program **vlsi.c** uses the **tie** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

/usr/include/gl/gl.h	Contains constant and variable type definitions for GL.
/usr/include/gl/device.h	Contains constant and variable type definitions for devices.

Related Information

Returning the current state of a button with the **getbutton** subroutine.

GL Introduction, Controlling the Keyboard in GL, and Controlling Queues and Devices in GL in *Graphics Programming Concepts*.

tpoff Subroutine

Purpose

Turns off the textport.

Library

Graphics Library (**libgl.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 appear 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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Working with the Textport in GL in *Graphics Programming Concepts.*

tpon Subroutine

Purpose

Turns on the textport.

Library

Graphics Library (**libgl.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

1. To enable a textport for drawing character strings into, the example C language program **tpbig.c** uses the **tpon** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction and Working with the Textport in GL in *Graphics Programming Concepts*.

translate Subroutine

Purpose

Translates graphical primitives.

Library

Graphics Library (**libgl.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

1. To model the sides of a cube using a square, the example C language program **backface.c** uses the **translate** modeling subroutine and alters the current transformation matrix.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

underlay Subroutine

Purpose

Sets the number of user-defined bitplanes used for underlay drawing.

Library

Graphics Library (**libgl.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, 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.

Note: 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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Configuring the Frame Buffer, and Controlling Frame Buffer Update in *Graphics Programming Concepts*.

unqdevice Subroutine

Purpose

Disables an input device for event queuing

Library

Graphics Library (**libgl.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

1. To disable input from the keyboard, the example C language program **prompt.c** uses the **unqdevice** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

Files

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

/usr/include/gl/device.h Contains constant and variable type definitions for devices.

Related Information

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

Emptying the event queue with the **qreset** subroutine.

GL Introduction and Controlling Queues and Devices in GL in *Graphics Programming Concepts*.

v Subroutine

Purpose

Transfers a 2–D, 3–D, or 4–D vertex to the graphics pipe.

Library

Graphics Library (**libgl.a**)

C Syntax

void v2s(Int16 *vector*[2])
void v2i(Int32 *vector*[2])
void v3s(Int16 *vector*[3])
void v3i(Int32 *vector*[3])
void v4s(Int16 *vector*[4])
void v4i(Int32 *vector*[4])
void v2f(Float32 *vector*[2])
void v2d(Float64 *vector*[2])
void v3f(Float32 *vector*[3])
void v3d(Float64 *vector*[3])
void v4f(Float32 *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.

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

Parameter

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

1. To specify the vertices of a polygon, the example C language program **cylinder2.c** calls the **v3f** subroutine between the **bgnpolygon** subroutine and the **endpolygon** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

Drawing with Begin–End Style Subroutines and Understanding the Hardware Used by GL in *Graphics Programming Concepts*.

viewport Subroutine

Purpose

Sets the area of the window used for all drawing.

Library

Graphics Library (**libgl.a**)

C Syntax

void viewport
(Screencoord *left*, **Screencoord** *right*,
Screencoord *bottom*, **Screencoord** *top*)

FORTRAN Syntax

SUBROUTINE VIEWPO(*left*, *right*, *bottom*, *top*)
INTEGER*2 *left*, *right*, *bottom*, *top*

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

1. To define a viewport for displaying an image, the example C language program **paint.c** uses the **viewport** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Working with Coordinate Systems in GL, and Using Viewports and Screenmasks in GL in *Graphics Programming Concepts*.

winclose Subroutine

Purpose

Closes the identified window.

Library

Graphics Library (**libgl.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.

Note: This 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.

File

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

Related Information

Creating a window with the **winopen** subroutine.

Creating and Managing Windows in GL in *Graphics Programming Concepts*.

winconstraints Subroutine

Purpose

Binds window constraints to the current window.

Library

Graphics Library (**libgl.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

1. 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.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

windepth Subroutine

Purpose

Indicates the stacking order of windows on the screen.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

window Subroutine

Purpose

Defines a perspective projection transformation in terms of x and y coordinates.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction and Working with Coordinate Systems in *Graphics Programming Concepts*.

winget Subroutine

Purpose

Returns the identifier of the current window.

Library

Graphics Library (**libgl.a**)

C Syntax

Int32 winget( )

FORTRAN Syntax

INTEGER*4 FUNCTION WINGET

Description

The **winget** subroutine returns the identifier of the current window.

Note: This 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.

File

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

Related Information

Setting the current window with the **winset** subroutine.

Creating and Managing Windows in GL in *Graphics Programming Concepts*.

winmove Subroutine

Purpose

Moves the current window by its lower left corner.

Library

Graphics Library (**libgl.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 does not change the size and shape of the window.

If the current window position is constrained, it continues to be constrained after the **winmove** subroutine is called, but the new position applies.

If the current window is not constrained, the **winmove** subroutine does not introduce a constraint. The applications user can still pick up and move the 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.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

winopen Subroutine

Purpose

Creates a window.

Library

Graphics Library (**libgl.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

1. To create a window with the previously defined characteristics, the example C language program **colored.c** uses the **winopen** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

winpop Subroutine

Purpose

Raises the current window on top of all other windows.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

winposition Subroutine

Purpose

Changes the size and position of the current window.

Library

Graphics Library (**libgl.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.

If the current window position and/or size is constrained, it continues to be constrained after the **winposition** subroutine is called, but the new position and size apply.

If the current window is not constrained, the **winposition** subroutine does not introduce a constraint. The applications user can still pick up and move the 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.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

winpush Subroutine

Purpose

Lowers the current window beneath all other windows.

Library

Graphics Library (**libgl.a**)

C Syntax

void winpush( )

FORTRAN Syntax

SUBROUTINE WINPUS

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.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

winset Subroutine

Purpose

Sets the current window.

Library

Graphics Library (**libgl.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.

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

Parameter

windowid Specifies which window to set as current.

Example

1. To make the newly opened window the current window, the example C language program **colored.c** uses the **winset** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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 in GL in *Graphics Programming Concepts*.

wintitle Subroutine

Purpose

Adds a title bar to the current window.

Library

Graphics Library (**libgl.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.

File

/usr/include/gl/gl.h	Contains 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 in GL in *Graphics Programming Concepts*.

wmpack Subroutine

Purpose

Specifies RGBA writemask with a single packed integer.

Library

Graphics Library (**libgl.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.

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

Parameter

pack Specifies a packed integer containing the RGBA writemask.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, Removing Hidden Surfaces, and Smoothing Jagged Lines with Antialiasing in *Graphics Programming Concepts*.

writemask Subroutine

Purpose

Grants write permission to available bitplanes.

Library

Graphics Library (**libgl.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.

Notes:

1. This subroutine is intended for use only in color map mode. To set the writemask in RGB mode, use the **RGBwritemask** subroutine.
2. This subroutine cannot be used to add to a display list.

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

1. To draw a rectangle in one of four colors, the example C language program **vlsi.c** uses the **writemask** subroutine to enable writing into only one of four bitplanes.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Configuring the Frame Buffer, Controlling Frame Buffer Update, and Working in Color Map and RGB Modes in *Graphics Programming Concepts*.

writepixels Subroutine

Purpose

Paints a row of pixels on the screen in color map mode.

Library

Graphics Library (**libgl.a**)

C Syntax

void writepixels(Int16 *number*, **Colorindex** *colors*[ ])

FORTRAN Syntax

SUBROUTINE WRITEP(*number*, *colors*)
INTEGER*2 *number*
INTEGER*2 *colors*(*number*)

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. Do not use the **writepixels** subroutine in new development.

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

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.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Reading and Writing Pixels in GL, Using Viewports and Screenmasks in GL, Configuring the Frame Buffer for GL, Creating Animated Screens in GL, and Working in Color Map and RGB Modes in GL in *Graphics Programming Concepts*.

writeRGB Subroutine

Purpose

Paints a row of pixels on the screen in RGB mode.

Library

Graphics Library (**libgl.a**)

C Syntax

void writeRGB
(Int16 *number*,
RGBvalue *red*[ ], **RGBvalue** *green*[ ], **RGBvalue** *blue*[ ])

FORTRAN Syntax

SUBROUTINE WRITER(*number*, *red*, *green*, *blue*)
INTEGER*2 *number*
CHARACTER*(*) *red*, *green*, *blue*

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. Do not use the **writeRGB** subroutine in new development.

Notes:

1. This subroutine is available only in RGB mode.
2. This subroutine cannot be used to add to a display list.

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Reading and Writing Pixels in GL, Using Viewports and Screenmasks in GL, and Working in Color Map and RGB Modes in GL in *Graphics Programming Concepts*.

zbuffer Subroutine

Purpose

Enables or disables the z-buffer for storing depth information.

Library

Graphics Library (**libgl.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	TRUE enables the z-buffer. FALSE disables the z-buffer.

Example

1. To disable the display of hidden surfaces, the example C language program **zbuf.c** enables z-buffering with the **zbuffer** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Configuring the Frame Buffer, and Removing Hidden Surfaces in *Graphics Programming Concepts.*

zclear Subroutine

Purpose

Initializes the z-buffer.

Library

Graphics Library (**libgl.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.

Example

1. To initialize the z-buffer, the example C language program **zbuf.c** uses the **zclear** subroutine.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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

GL Introduction, Getting Ready to Run GL, Starting GL Functions, and Configuring the Frame Buffer for GL in *Graphics Programming Concepts*.

zdraw Subroutine

Purpose

Enables or prohibits drawing to the z-buffer.

Library

Graphics Library (**libgl.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 prohibits drawing to the z-buffer. Calling the **frontbuffer** subroutine with a value of TRUE also prohibits drawing to the z-buffer.

Calling the **zdraw** subroutine with a value of TRUE affects only the pixel writing subroutines: **writepixels**, **writeRGB**, **rectwrite**, **lrectwrite**, and **rectcopy**. 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.

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

Parameter

bool	TRUE allows drawing into the z buffer. FALSE makes the z buffer read only.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h	Contains 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.

GL Introduction, Configuring the Frame Buffer, and Removing Hidden Surfaces in *Graphics Programming Concepts.*

zfunction Subroutine

Purpose

Specifies the function used for depth comparison.

Library

Graphics Library (**libgl.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.

Usually, the comparison between the source and destination *z* value is a comparison of depth values (this is what is normally referred to as z-buffering). Use the **zsource** subroutine to compare color values.

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

Parameter

func Expects one of eight possible flags used when comparing *z* values. The available flags are:

ZF_NEVER, never overwrite the destination pixel value.

ZF_LESS, overwrite the destination pixel value if the *z* of the source pixel value is less than the z of destination value.

ZF_EQUAL, overwrite the destination pixel value if the *z* of the source pixel value is equal to the z of destination value.

ZF_LEQUAL, 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, overwrite the destination pixel value if the z of the source pixel value is greater than the z of destination value.

ZF_NOTEQUAL, overwrite the destination pixel value if the z of the source pixel value is not equal to the z of destination value.

ZF_GEQUAL, 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, always overwrite the destination pixel value.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Controlling Frame Buffer Update, and Removing Hidden Surfaces in *Graphics Programming Concepts*.

zsource Subroutine

Purpose

Selects either depth or color as the source for *z* comparisons.

Library

Graphics Library (**libgl.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.

The High-Performance 3-D Color Graphics Processors currently support only the ZSRC_DEPTH setting. The ZSRC_COLOR setting has no effect and is ignored.

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

Parameter

source Specifies one of two possible flags:

ZSRC_COLOR - buffering done by color value.

ZSRC_DEPTH - buffering done by *z* value.

Implementation Specifics

This subroutine is part of GL in the AIXwindows environment.

File

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

Related Information

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.

GL Introduction, Controlling Frame Buffer Update, and Removing Hidden Surfaces in *Graphics Programming Concepts*.

zwritemask Subroutine

Purpose

Specifies the z-buffer writemask.

Library

Graphics Library (**libgl.a**)

C Syntax

void zwritemask(Int32 *mask***)**

FORTRAN Syntax

SUBROUTINE ZWRITE(*mask***)**
INTEGER*4 *mask*

Description

The **zwritemask** subroutine specifies which bits of the z-buffer are written during normal z-buffer operation (not by **zdraw**). This subroutine is significant only while drawing in z-buffer mode. When the **z** compare is TRUE, the current destination *z* is replaced by the new source *z*. The **zwritemask** subroutine controls which bits of the destination *z* are actually replaced, and which retain their previous values.

The **zwritemask** subroutine is ignored while drawing directly to the z-buffer, as when the value of the *bool* parameter in the **zdraw** subroutine is TRUE. During the **zdraw** subroutine, the current writemask applies to the z-buffer as well as to the color bitplanes.

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

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.

File

/usr/include/gl/gl.h Contains 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.

GL Introduction, Controlling Frame Buffer Update, and Removing Hidden Surfaces in *Graphics Programming Concepts*.

Chapter 2. GL Example Programs

backface.c Example C Language Program

```
/*
backface.c

Draw a cube that can run with backface() turned on or
off. Turn backface() on with with the F key.
Turn backface() off with 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.0);
    color(RED);
    rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
    popmatrix();
  /* right face */
    pushmatrix();
    translate(CUBE_SIZE.0, 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.0);
    rotate(1800, 'y');
    color(BLUE);
    rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
    popmatrix();
  /* left face */
    pushmatrix();
    translate(-CUBE_SIZE.0, 0.0, 0.0);
    rotate(-900, 'y');
    color(CYAN);
    rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
    popmatrix();
```

```
  /* top face */
    pushmatrix();
    translate(0.0, CUBE_SIZE.0, 0.0);
    rotate(-900, 'x');
    color(MAGENTA);
    rectfi(-CUBE_SIZE,-CUBE_SIZE,CUBE_SIZE,CUBE_SIZE);
    popmatrix();
  /* bottom face */
    pushmatrix();
    translate(0.0, -CUBE_SIZE.0, 0.0);
    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();
}
```

boxcirc.c Example C Language Program

```
/*
boxcirc.c:

A simple example which draws 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();
}
```

colored.c Example C Language Program

```
/*
colored.c

Edit the color map and display the results in the graphics window.

This program requires a 24-bit adapter.
*/

#include <gl/gl.h>
#include <gl/device.h>

#define CURRENTCOLOR 253
#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(512+i*64+j, indextovalue(j),
                          indextovalue(greenindex),
                          indextovalue(blueindex));
                    break;
                case 1:                 /* green */
                    mapcolor(512+i*64+j, indextovalue(redindex),
                          indextovalue(j),
                          indextovalue(blueindex));
                    break;
                case 2:                 /* blue */
                    mapcolor(512+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(512+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);
}
```

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 (or rcrv) is called, a basis and precision matrix must be
 defined.
*/

#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, 0.5, 0 },
                          { 0, 1, 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 },
                         { -3.0/6.0, 0, 3.0/6.0, 0 },
                         { 1.0/6.0, 4.0/6.0, 1.0/6.0, 0 } };

#define BEZIER 1
#define CARDINAL 2
#define BSPLINE 3

Coord geom1[4][3] = { { 100.0, 100.0, 0.0},
                      { 200.0, 200.0, 0.0},
                      { 200.0, 0.0, 0.0},
                      { 300.0, 100.0, 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();
            }
        }
    }
}
```

curve1.c

```
initialize() {
    int gid;
    prefposition(200, 500, 100, 400);
    gid = winopen("curve1");
    qdevice(ESCKEY);
    qdevice(REDRAW);
    qenter(REDRAW,gid);
}
drawcurve() {
    color(BLACK);
    clear();
    translate(150.0, 150.0, 0.0);
    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);                        /* draw the curve based on
                                          four control points in
                                          geom1 */
    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 */
}
```

curve2.c Example C Language Program

```
/* 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, 0.5, 0 },
                          { 0, 1, 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 },
                         { -3.0/6.0, 0, 3.0/6.0, 0 },
                         { 1.0/6.0, 4.0/6.0, 1.0/6.0, 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");
    qdevice(ESCKEY);
    qdevice(REDRAW);
    qenter(REDRAW,gid);
}
drawcurve() {
    color(BLACK);
    clear();
    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 */
}
```

curve3.c Example C Language Program

```
/* Example C Language Program curve3.c */

/*
This program draws a Bezier curve segment using curveit().  The
Cardianl 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>

Matrix beziermatrix = { { -1, 3, -3, 1 },
                        { 3, -6, 3, 0 },
                        { -3, 3, 0, 0 },
                        { 1, 0, 0, 0 } };

#define BEZIER 1

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} };

Matrix precisionmatrix = { { 6.0/8000.0, 0, 0, 0},
                           { 6.0/8000.0, 2.0/400.0, 0, 0},
                           { 1.0/8000.0, 1.0/400.0, 1/20.0, 0},
                           { 0, 0, 0, 1} };

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");
    qdevice(ESCKEY);
    qdevice(REDRAW);
    qenter(REDRAW,gid);
}
```

```
drawcurve() {
    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 */
}
```

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);
}
```

cylinder1.c Example C Language Program

```
/* Example C Language 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 the 24-bit adapter and the z-buffer option.
*/

#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 */
                         /* 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 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);
}
```

cylinder2.c Example C Language Program

```
/* 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 sources.

This program requires the 24-bit adapter and the z-buffer option.
*/
#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 */
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);
}
```

db.c Example C Language Program

```
/*
db.c:

A double buffered window manager program.  Draws a cube which is
rotated by movements of the mouse.
*/

#include <gl/gl.h>
#include <gl/device.h>

main()
{
    int moveit, x, y;   /*  current rotation of object */
    Device dev;
    short val;

    x = 0;
    y = 0;
    initialize();

    while(TRUE) {

        while (qtest()) {       /*  process queued tokens */

            dev = qread(&val);

            switch(dev) {
                 case ESCKEY:   /*  exit program with ESC */
                     exit(0);
                     break;
                  case REDRAW:
                     reshapeviewport();
                     drawcube(x,y);
                     break;
                  default:
                     break;
            }
        }
        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("db");

    doublebuffer();
    gconfig();
    shademodel(FLAT);

    qdevice(REDRAW);
    qdevice(ESCKEY);
    qenter(REDRAW,gid);
```

```
    perspective(400, 3.0/2.0, 0.001, 100000.0);
    translate(0.0, 0.0, -3.0);
}

drawcube(rotx,roty)
int rotx, roty;
{
    color(BLACK);
    clear();
    color(WHITE);
    pushmatrix();
    rotate(rotx,'x');
    rotate(roty,'y');
    cube();
    scale(0.3,0.3,0.3);
    cube();
    popmatrix();
    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();
}
```

depthcue.c Example C Language Program

```
/*
depthcue.c:

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 are currently set to 350.0 and 1000.0, respectively.
Give the command line 2 extra arguements as floating point near
and far values to change where Z gets clipped, which will also
alter the distribution of brightness to darkness of the visual
depth cues. Press the middle mouse button to quit.

This program requires the 24-bit adapter and the z-buffer option.
*/

#include <gl/gl.h>
#include <gl/device.h>
#include <math.h>

float hrand();

main (argc,argv)
int argc;
char **argv;
{
    int val, i;
    float near, far;

    prefposition(XMAXSCREEN/4,XMAXSCREEN*3/4,YMAXSCREEN/4,
          YMAXSCREEN*3/4);
    winopen("depthcue");
    doublebuffer();
    gconfig();
    if (argc == 3)  {
        near = atof(argv[1]);
        far = atof(argv[2]);
    }
    else {
        near = 350.0;
        far = 1000.0;
    }
    reshapeviewport();
    perspective(600, 1.0, near, far);
    lookat(0.0, 0.0, 700.0, 0.0, 0.0, 0.0, 0);
    qdevice(KEYBD);

    makeobj(1);

    /* 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();

/* load the color map with a cyan ramp */
    for (i = 0; i <= 127; i++)
        mapcolor(128+i, 0, 2*i, 2*i);

/* set the range of z values that will be stored
    in the bitplanes */
    lsetdepth(0xC000, 0x3FFF);

/* set up the mapping of z values to color map indexes:
    z value 0xC000 is mapped to index 128 and z value 0x3FFF is
    mapped to index 255 */
    lshaderange(128,255,0xC000,0x3FFF);

/* turn on depthcue mode:  the color index of each pixel in points
    and lines is determined from the z value of the pixel */
    depthcue(1);

/* 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(1);
        popmatrix();
        swapbuffers();
    }
    gexit();
}

/* this routine returns random numbers in the specified range */
float hrand(low,high)
float low,high;
{
    float val;

    val = ((float)( (short)rand(0) & 0xffff)) / ((float)0xffff);
    return( (2.0 * val * (high-low)) + low);
}
```

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 equivalent 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();
           } 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]);
        }
}
```

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);
}
```

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();
}
```

localatten.c Example C Language Program

```
/* Example C Language Program localatten.c */

/*
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();
    }
}
```

octahedron.c Example C Language Program

```
/*
octahedron.c

This program defines a drawing subroutine called drawoctahedron
that uses the mesh subroutines. (The drawoctahedron subroutine
is not part of the Graphics Library.)

The cpack subroutine sets vertex colors, so ignore them if you
are studying the program to understand the logic of mesh drawing.
All the rotation and hidden surface removal are handled in the
main() routine. The calculations of rotation angles simply cause
the octahedron to tumble in an interesting way.

This program requires a 24-bit adapter and the z-buffer option.
/*

#include <gl/gl.h>

float octdata[6][3] = {
    {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}
};
```

```
drawoctahedron()
{
   bgntmesh();
    cpack(0xff0000);            /* color blue */
    v3f(octdata[0]);
    cpack(0x00ff00);            /* color green */
    v3f(octdata[1]);
    swaptmesh();
    cpack(0x0000ff);            /* color red */
    v3f(octdata[2]);
    swaptmesh();
    cpack(0xffff00;             /* color cyan */
    v3f(octdata[4]);
    swaptmesh();
    cpack(0xffffff);            /* color white */
    v3f(octdata[5]);
    swaptmesh();
    cpack(0x00ff00);            /* color green */
    v3f(octdata[1]);
    cpack(0xff00ff);            /* color magenta */
    v3f(octdata[3]);
    cpack(0x0000ff);            /* color red */
    v3f(octdata[2]);
    swaptmesh();
    cpack(0xffff00);            /* color cyan */
    v3f(octdata[4]);
    swaptmesh();
    cpack(0xffffff);            /* color white */
    v3f(octdata[5]);
    swaptmesh();
    cpack(0x00ff00);            /* color green */
    v3f(octdata[1]);
   endtmesh();
}

main()
{
   long iang, jang, kang;
   long exitcounter = 0;
```

```
    prefposition(100, 500, 100, 500);
    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 */
    while(1) {
       pushmatrix();              /*save viewing transformation*/
       rotate(iang, 'x');         /*rotate by iang about x axis*/
       rotate(jang, 'y');         /*rotate by jang about y axis*/
       rotate(kang, 'z');         /*rotate by kang about z axis*/
       iang += 10;
       jang += 13;
       if (iang + jang > 3000) kang += 17;
       if (iang > 3600) iang -= 3600;
       if (jang > 3600) jang -= 3600;
       if (kang > 3600) kang -= 3600;
       cpack(0);          /* color black */
       clear();
       zclear();          /* clear the z buffer */
       drawoctahedron();
       swapbuffers();     /* show completed drawing */
       popmatrix();               /* restore viewing transformation */
       exitcounter += 1;
       if (exitcounter == 1000) return;
     }
}
```

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]);
}
```

paint.c Example C Language Program

```
/*
 *        paint -
 *                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);
}
```

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 ba
sed 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 } };
```

patch1.c

```
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;

    prefposition(XMAXSCREEN/4, XMAXSCREEN*3/4, YMAXSCREEN/4,
          YMAXSCREEN*3/4);
    gid = winopen("patch1");

    qdevice(ESCKEY);
    qdevice(REDRAW);
    qenter(REDRAW,gid);

    ortho(0.0-20.0, (float)(XMAXSCREEN*3/4),
          0.0-20.0, (float)(YMAXSCREEN*3/4),
          (float)XMAXSCREEN, -(float)XMAXSCREEN);

}
```

```
drawpatch()
{
   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 */
}
```

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:
                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();
}
```

platelocal.c Example C Language Program

```
/* Example C Language Program platelocal.c */

/* 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). */

/*
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.

This program requires the 24-bit adapter.
*/

#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();
    }
}
```

popup.c Example C Language Program

```
/*
popup.c:
Demonstrates "how to write your own popup menu" routines.
Use LEFTMOUSE instead of RIGHTMOUSE to pop up the menus.
*/

#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;
}
```

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';
}
```

scrn_rotate.c Example C Language Program

```
/*
scrn_rotate.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.

                Paul Mlyniec and David Ratcliffe - 1986
*/
#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);
    winopen("screeen rotation");
    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();
}
```

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();
}
```

setshade.c Example C Language Program

```
/*
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);
    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 short CarrayR[hi_end+1], CarrayG[hi_end+1],
      CarrayB[hi_end+1];
unsigned short index;

get_cmap()
{
    short rcomp, gcomp, bcomp;

    for (index=lo_end; index<=hi_end; index++) {
        getmcolor(index,&rcomp, &gcomp, &bcomp);
        CarrayR[index] = rcomp;
        CarrayG[index] = gcomp;
        CarrayB[index] = bcomp;
    }
}

restore_cmap()
{
    for (index=lo_end; index<=hi_end; index++)
        mapcolor(index, CarrayR[index], CarrayG[index],
              CarrayB[index]);
}
```

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");
    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);
}
```

text.c Example C Language Program

```
/*
text.c:

A text drawing sample program using the charstr() subroutine.
*/

#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. ");
}
```

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 */
    memcopy(singlechar, "X", 2);
    prefposition(0,XMAXSCREEN,0,YMAXSCREEN);
    gid = winopen("tpbig");
    qdevice(ESCKEY);
    qdevice(REDRAW);
    qenter(REDRAW,gid);
    textport(50,300,750,900);
    tpon();

    while(TRUE) {
        switch(qread(val)) {
            case ESCKEY:
                textinit();
                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);
        }
    }
}
```

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");
    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:
                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);
}
```

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);
}
```

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>

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) */


   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.
    */
   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\n");

    sleep (5);
}
```

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();
}
```

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();
}
```

Part 2. AIXwindows Graphics Support Library Reference (XGSL)

Chapter 3. XGSL Subroutines

gsbply Subroutine

Purpose

Defines the beginning of an area to fill.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsbply_ ( )

FORTRAN Syntax

INTEGER function gsbply

Pascal Syntax

FUNCTION gsbply_ : INTEGER [PUBLIC];

Description

The **gsbply** subroutine defines the beginning of a two-dimensional shape or set of shapes to be filled.

The following output routines are valid between a call to the **gsbply** subroutine and a call to the **gseply** subroutine:

gspoly	Draws a polyline.
gscir	Draws a circle.
gsell	Draws an ellipse.
gscarc	Draws a circular arc between two points.
gscrca	Draws a circular arc between two angles.
gsearc	Draws an elliptical arc between two points.
gseara	Draws an elliptical arc between two angles.

Note: Any other subroutines used before the **gseply** subroutine is called do not become part of the shape or set of shapes to be filled and can produce unpredictable results.

Before the fill occurs, the shapes drawn by each routine called between the **gsbply** and **gseply** subroutines are connected. The first point of each shape is linked to the last point of the previous shape, and the last point of the last shape is linked to the first point of the first shape. The shapes may overlap to any degree but must share at least one common point between adjacent shapes.

Processing of the **SIGRETRACT** signal is postponed until the call to the **gseply** subroutine, which defines the end of an area to fill.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern
- Logical operation.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_USUC	Unsuccessful.

Example

1. To define the beginning of an area to fill, the **blit.c** C language program uses the **gsbply** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gseply** subroutine, **gspcls** subroutine, **gsell** subroutine, **gscarc** subroutine, **gscrca** subroutine, **gsearc** subroutine, **gseara** subroutine, **gspoly** subroutine, **gscir** subroutine.

gscarc Subroutine

Purpose

Draws a circular arc between two points.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gscarc_ (*cx, cy, cr, bx, by, ex, ey*)
int **cx, *cy, *cr, *bx, *by, *ex, *ey;*

FORTRAN Syntax

INTEGER function gscarc_(*cx, cy, cr, bx, by, ex, ey*)
INTEGER *cx, cy, cr, bx, by, ex, ey*

Pascal Syntax

FUNCTION gscarc_ (
VAR *cx, cy, cr, bx, by, ex, ey* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gscarc** subroutine draws a counterclockwise circular arc of a specified radius from beginning point to ending point. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of a circle. For displays, the center is restricted to a range from –2048 to 2048.
cr	Defines the radius of a circle.
bx, by	Define the coordinates of the beginning point on a circle.
ex, ey	Define the coordinates of the ending point on a circle.

Notes:

1. If the beginning and ending points are identical, a full circle is drawn.
2. The application must control the accuracy of the end points (*bx, by* and *ex, ey*) when drawing circular arcs. If the start point of the arc and end point of the arc lie within one pixel of the true circle, the arc is drawn successfully. Other values can cause the subroutine to fail. If the **gscarc** subroutine fails because of an inaccurate starting point, the GS_ASTR value is returned. For an inaccurate ending point, the GS_AEND value is returned.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_RDUS	Radius specification not valid
GS_INAC	Virtual terminal inactive
GS_AEND	Ending point not valid
GS_ASTR	Beginning point not valid.

Example

1. To draw a circular between two points, the **arc3.c** C language program uses the **gscarc** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gscrca** subroutine, **gseara** subroutine, **gsearc** subroutine.

gscatt Subroutine

Purpose

Sets the attributes of the single-color cursor.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gscatt_ (*Color*, *Width*, *Height*, *Pattern*, *0x*, *0y*)
int **Color*, **Width*, **Height*;
int *Pattern*, **0x*, *0y*;

FORTRAN Syntax

INTEGER function gscatt_ (*Color*, *Width*, *Height*, *Pattern*, *0x*, *0y*)
INTEGER *Color*, *Width*, *Height*, *Pattern*, *0x*, *0y*

Pascal Syntax

FUNCTION gscatt_ (
VAR *Color*, *Width*, *Height*: **INTEGER**;
Pattern: **ARRAY [1..***k***] of INTEGER;**
0x, *0y*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gscatt** subroutine defines the single-color cursor for the AIXwindows Graphics Support Library (XGSL). The **gscmap** subroutine must initialize the color map before the **gscatt** subroutine can be called.

Only one cursor, either the single-color cursor or the multicolor cursor, can be active in the XGSL at any one time. The **gscatt** subroutine forces all subsequent calls to the **gsmcur** and **gsecur** subroutines to operate on the single-color version of the cursor. To change from the multicolor cursor to the single-color cursor, erase the cursor with the **gsecur** subroutine, and then call the **gscatt** subroutine.

Parameters

Color	Refers to an entry in the color map. If the index value is –1 (negative one), the attribute is unchanged.
Width, *Height*	Define, in pixels, the width and height of the bit pattern to be used as the cursor. If either the *Width* or *Height* parameter equals –1 (negative one), the pattern remains unchanged.
Pattern	Defines the image used as a cursor. Dividing the value of the *Width* parameter by 32 and rounding the result to the next highest integer give the ceiling of the calculation. This ceiling indicates the number of words per row. The *Height* parameter indicates the number of rows. The cursor data must be supplied in row (scan line) major order. To define the cursor pattern fully, calculate the size, in words, of the *Pattern* array by multiplying the previously calculated ceiling by the

value of the *Height* parameter. If the *Width* parameter implies partial use of a word, the rest of the word is unused.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length; that is, the *k* in the routine declaration must be a constant.

Note: The maximum size of the cursor is device-dependent and can be determined by using the **gsqdsp** subroutine.

0x, *0y* — Indicate the origin of the cursor relative to the lower leftmost corner (0, 0) of the cursor pattern. The origin must be placed within the cursor pattern: *0x* < *Width* and *0y* < *Height*. When the application moves the cursor using the **gsmcur** subroutine, the origin of the cursor is placed at the position indicated. If *x* equals –1 (negative one), the origin remains unchanged.

Notes:

1. The cursor attributes cannot be changed while the cursor is visible.
2. No default cursor is defined. All cursor parameters must be set before the cursor is displayed.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_COLI	Color index not valid
GS_CURS	Cursor size not valid
GS_CURO	Cursor origin not valid
GS_CURV	Cursor visible.

Example

1. To set the attributes for a single-color cursor, the **curs.c** C language program uses the **gscatt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsecur** subroutine, **gsmcur** subroutine, **gsqdsp** subroutine.

gsccnv Subroutine

Purpose

Converts a circular arc or full circle into a polyline.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsccnv_ (*cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*, *Len*, *x*, *y*, *Pre*)
int **cx*, **cy*, **cr*;
int **bx*, **by*;
int **ex*, **ey*, **Len*;
int **x*[ ], **y*[ ], **Pre*;

FORTRAN Syntax

INTEGER function gsccnv (*cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*, *Len*, *x*, *y*, *Pre*)
INTEGER *cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*, *Len*, *x*(*), *y*(*), *Pre*

Pascal Syntax

FUNCTION gsccnv_ (
VAR *cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*, *Len*: **INTEGER;**
VAR *x*, *y*: **ARRAY [1..***k***] of INTEGER;**
VAR *Pre*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsccnv** subroutine converts a counterclockwise circular arc definition into a set of vertices. The returned vertices can then be used to draw a circular arc with the **gspoly** subroutine or to fill a circular arc with the **gsfply** subroutine. In general, it can be concatenated with other lists of vertices to draw or fill more complex shapes, such as chord arcs, pie arcs, and rectangles with rounded corners.

When beginning and ending points are identical, the list of vertices contains the full circle, which can then be drawn or filled.

Parameters

cx, cy — Define the coordinates of the center of the circle. These are input parameters, specified by the application.

cr — Defines the radius of the circle. It must not equal 0. This is an input parameter, specified by the application.

Note: If the *cr* parameter is negative, it is automatically converted to a positive value for use by this subroutine.

bx, by — Define the coordinates of the beginning point of the arc. These are input parameters specified by the application.

ex, ey Define the coordinates of the ending point of the arc. These are input parameters specified by the application.

Note: The subroutine allows ample leniency toward the accuracy of the specification of the beginning and ending points. The arc of the specified radius will always start and end exactly at the specified points. If the beginning and ending points are identical, the vertices for a full circle of the specified radius is generated.

Len Specifies, on return, the number of points (vertices) in the *x* and *y* coordinate arrays. If error conditions arise, *Len* is set to a value of 0. Before you call the **gsccnv** subroutine, you must set the value of the *Len* parameter to at least one greater than the value of the *Pre* (precision) parameter.

x, y Define, as coordinate arrays, the vertices that represent the circular arc specified by the application. These parameters are generated and returned by the **gsccnv** subroutine.

For Pascal syntax, the application must declare that the *x* and *y* arrays are fixed-length and that the **gsccnv** subroutine accepts arrays of that length; that is, the *k* in the subroutine declaration must be a constant.

Pre Defines precision level, which specifies the maximum number of line segments that can be generated for a full circle. The number of line segments actually generated depends on the arc length defined by the binning point (*bx*,*by*) and ending point (*ex*,*ey*).

The *Pre* parameter must be one of the following four values which specify the corresponding number of vertices:

- 64 (65 vertices)
- 128 (129 vertices)
- 256 (257 vertices)
- 512 (513 vertices).

All other precision values are reserved and must not be used because their results are unpredictable. The default value for the *Pre* parameter is 64.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_CORD Coordinate not valid

GS_NCOR Number of coordinates not valid.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsfply** subroutine, **gspoly** subroutine.

gscir Subroutine

Purpose

Draws a circle.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gscir_ (*Cx*, *Cy*, *Cr*)
int **Cx*, **Cy*, **Cr*;

FORTRAN Syntax

INTEGER function gscir (*Cx*, *Cy*, *Cr*)
INTEGER *Cx*, *Cy*, *Cr*

Pascal Syntax

FUNCTION gscir_ (
VAR *Cx*, *Cy*, *Cr*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gscir** subroutine draws a circle of the specified radius. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

Cx, Cy	Define the coordinates of the center of the circle.
Cr	Defines the radius of the circle. If the radius is zero, a single point is drawn at the center.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_RDUS	Radius specification not valid
GS_INAC	Virtual terminal inactive.

Example

1. To draw a circle, the **cir1.c** C language program uses the **gscir** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsccnv** subroutine.

gsclrs Subroutine

Purpose

Clears the screen and fills it with the background color.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsclrs_ ( )

FORTRAN Syntax

INTEGER function gsclrs_

Pascal Syntax

FUNCTION gsclrs_: INTEGER [PUBLIC];

Description

The **gsclrs** subroutine fills the frame buffer with the background color (color index zero) as defined by the color map attribute.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_INAC	Virtual terminal inactive.

Example

1. To clear the screen and fill it with the background color, the **arc4.c** C language program uses the **gsclrs** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gscmap Subroutine

Purpose

Specifies the color mapping.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gscmap_ (*Number*, *Red*, *Green*, *Blue*)
int **Number*, **Red*, **Green*, **Blue*;

FORTRAN Syntax

INTEGER function gscmap (*Number*, *Red*, *Green*, *Blue*)
INTEGER *Number*, *Red* (*), *Green* (*), *Blue* (*)

Pascal Syntax

FUNCTION gscmap_ (
VAR *Number* **INTEGER;**
VAR *Red*, *Green*, *Blue*: **ARRAY [0..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gscmap** subroutine specifies the mapping between the color index attribute and the color it produces on the display.

The default color table mapping for the first 16 colors is the same as the default color map attributes in KSR mode. The remaining color values are initialized in a hardware-dependent manner.

Parameters

Number — Indicates how many colors the input-intensity arrays contain.

Red, Green, Blue — Define arrays that contain the intensity levels of the corresponding color. Each entry in an array specifies the intensity value for the corresponding color index.

The value in each entry for the *Red*, *Green*, and *Blue* intensity arrays is between 0x0000 (zero intensity) and 0x3FFF (full intensity). The following additional increments of intensity are possible, depending on the adapter hardware in use:

0x2000	1/2 intensity
0x1000	1/4 intensity
0x0800	1/8 intensity
0x0400	1/16 intensity
0x0200	1/32 intensity
0x0100	1/64 intensity.

Combinations of these values can be used to create intermediate levels of intensity. For example, 0x0C00 gives 3/16 intensity, while 0x3000 gives 3/4 intensity.

For Pascal syntax, the application must declare that the *Red*, *Green*, and *Blue* arrays are fixed-length and that the **gscmap** subroutine accepts an array of that length. That is, the *k* in the routine declaration must be a constant and it should be greater than or equal to the largest value for the *Number* parameter.

Notes:

1. The actual number of bits from bit 13 to bit 0 that affect the color on the display depends on the number of bits in the digital-to-analog converter of the adapter hardware in use. This size information is available by using the **gsqdsp** subroutine.
2. An application cannot change a single arbitrary color entry in the color map (or in the VLT). It must change all the entries for all the colors up to and including the desired entry.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_TABL	Table length not valid
GS_INAC	Virtual terminal inactive.

Examples

1. To set the mapping between a four-color index and the color produced on the display of an arc, the **arc4.c** C language program uses the **gscmap** subroutine.
2. To set the mapping between a color index and the color produced on the display, the **curs.c** C language program uses the **gscmap** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsqdsp** subroutine.

gscrca Subroutine

Purpose

Draws a circular arc between two angles.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gscrca_ (*cx*, *cy*, *cr*, *ba*, *ea*)
int **cx*, **cy*, **cr*;
int **ba*, **ea*;

FORTRAN Syntax

INTEGER function gscrca_ (*cx*, *cy*, *cr*, *ba*, *ea*)
INTEGER *cx*, *cy*, *cr*, *ba*, *ea*

Pascal Syntax

FUNCTION gscrca_ (
VAR *cx*, *cy*, *cr*, *ba*, *ea* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gscrca** subroutine draws a counterclockwise circular arc of a specified radius from the beginning point as defined by an angle specification to the ending point as defined by an angle specification.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gscrca** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of a circle. The center is restricted to a range of values from –2048 to 2048.
cr	Defines the radius of the circle in device coordinates.
ba	Defines the start point of the circular arc as an angle in tenths of degrees, from 0 to 3600.
ea	Defines the end point of the circular arc as an angle in tenths of degrees, from 0 to 3600. **Note:** If the beginning and ending angles are identical, a full circle is drawn.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_ANGL	Angle not valid
GS_RDUS	Radius specification not valid
GS_CORD	Coordinate not valid
GS_INAC	Virtual terminal inactive.

Example

1. To draw a circular arc between two angles, the **arc4.c** C language program uses the **gscrca** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gscarc** subroutine, **gseara** subroutine, **gsearc** subroutine.

gsdjply Subroutine

Purpose

Draws one or more sets of lines.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsdjply_ (*Polylines*, *Points*, *x*, *y*)
int **Polylines*;
int *Points*[ ], *x*[ ], *y*[ ];

FORTRAN Syntax

INTEGER function gsdjply_ (*Polylines*, *Points*, *x*, *y*)
INTEGER *Polylines*, *Points*(*), *x*(*), *y*(*)

Pascal Syntax

FUNCTION gsdjply_ (
VAR *Polylines*: **INTEGER;**
VAR *Points*: **ARRAY [1..***k***] of INTEGER;**
VAR *x*, *y*: **ARRAY [1..***l***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsdjply** subroutine draws one or more polylines. A polyline is a series of straight lines that are connected end-to-end. A polyline is specified by a series of points. If more than one polyline is specified, the individual polylines are not connected (*disjoint* polylines). All lines are drawn as defined by the current relevant attributes.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

Polylines — Defines the number of polylines to draw. This value must be greater than or equal to 1.

Points — An array of integers having for each polyline an integer that defines the number of points in the polyline. The value of the integer for each polyline must be greater than or equal to 2.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant, and it should be greater than or equal to the value of the *Polylines* parameter.

x, y Define, as arrays, the points for line drawing.

For Pascal syntax, the application must declare that the passed arrays are fixed-length and that the routine accepts arrays of that length. That is, the *I* in the routine declaration must be a constant and should be greater than or equal to the sum of the values in the *Points* array.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_CORD Coordinate not valid

GS_NCOR Number of coordinates not valid

GS_INAC Virtual terminal inactive.

Example

1. To draw two sets of polylines, the **djpoly.c** C language program uses the **gsdjply** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gspoly** subroutine, **gsplym** subroutine, **gsmatt** subroutine.

gsdpik Subroutine

Purpose

Provides compatibility for GSL applications that use the **gsdpik** subroutine.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsdpik_ ( )

FORTRAN Syntax

INTEGER function gsdpik

Pascal Syntax

FUNCTION gsdpik_ : INTEGER [PUBLIC];

Description

The **gsdpik** subroutine is provided for compatibility with existing GSL applications. It is ignored by the current implementation of XGSL.

Return Value

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return value:

GS_SUCC Successful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gseara Subroutine

Purpose

Draws an elliptical arc between two angles.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gseara_ (*Cx, Cy, Major, Minor, Ang, Sa, Ea*)
int **Cx, *Cy, *Major, *Minor;*
int **Ang, *Sa, *Ea;*

FORTRAN Syntax

INTEGER function gseara_ (*Cx, Cy, Major, Minor, Ang, Sa, Ea*)
INTEGER *Cx, Cy, Major, Minor, Ang, Sa, Ea*

Pascal Syntax

FUNCTION gseara_ (
VAR *Cx, Cy, Major, Minor, Ang, Sa, Ea* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gseara** subroutine draws a counterclockwise elliptical arc of the specified axes and angle from the beginning point defined by an angle specification to the ending point defined by an angle specification. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 (zero) to 3600. Values outside this range cause the **gseara** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

Cx, Cy	Define the coordinates of the center of the ellipse. The center is restricted to values ranging from –2048 to 2048.
Major, Minor	Define half of the nonzero major and minor axes of the ellipse.
Ang	Defines the angle between the major axis and the X axis. If the value of the *Ang* parameter is 0, the major axis is on the X axis and the minor axis is on the Y axis. The angle is expressed in tenths of degrees, from 0 to 3600.
Sa	Defines the angle of the starting point of the elliptical arc, measured counterclockwise from the major axis. The angle is expressed in tenths of degrees, from 0 to 3600.

Ea Defines the angle of the ending point of the elliptical arc, measured counterclockwise from the major axis. The angle is expressed in tenths of degrees, from 0 to 3600.

If the beginning and ending points are identical, a full ellipse is drawn.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_CORD Coordinate not valid

GS_ELMM Major or minor axis not valid

GS_INAC Virtual terminal inactive

GS_ANGL Angle not valid

GS_NMEM Insufficient memory.

Example

1. To draw several ellipses at different angles around a common center, the **arc1.c** C language program uses the **gseara** subroutine in a loop.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsearc** subroutine.

gsearc Subroutine

Purpose

Draws an elliptical arc between two points.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsearc_ (*cx, cy, Major, Minor, Ang, bx, by, ex, ey, Rot*)
int **cx, *cy, *Major, *Minor, *Ang;*
int **bx, *by, *ex, *ey, *Rot;*

FORTRAN Syntax

INTEGER function gsearc_ (*cx, cy, Major, Minor, Ang, bx, by, ex, ey, Rot*)
INTEGER *cx, cy, Major, Minor, Ang, bx, by, ex, ey, Rot*

Pascal Syntax

FUNCTION gsearc_ (
VAR *cx, cy, Major, Minor, Ang, bx, by, ex, ey, Rot* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsearc** subroutine draws a counterclockwise elliptical arc of specified axes and angle from beginning point to ending point. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsearc** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Linestyle
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of the ellipse. The center is restricted to a range of values from –2048 to 2048.
Major, Minor	Define half of the nonzero major and minor axes of the ellipse.
Ang	Defines the angle between the major axis and the X axis. If the value of the *Ang* parameter is 0 (zero), the major axis is on the X axis and the minor axis is on the Y axis. The angle is expressed in tenths of degrees, from 0 to 3600.
bx, by	Define the coordinates of the beginning point on the ellipse.
ex, ey	Define the coordinates of the ending point on the ellipse.

Rot Specifies whether the application must perform rotational transformation. Possible settings are:

0 The coordinates of the beginning and ending points passed by the application correspond to an arc of an orthogonal ellipse. No rotational transformation is performed, thus improving performance.

1 The beginning and ending points are transformed by the application and lie on the off-axis ellipse.

All other values are reserved and must not be used, as they can produce unpredictable results.

If the beginning and ending points are identical, regardless of whether they are on the ellipse, a full ellipse is drawn.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_ELMM	Major or minor axis not valid
GS_INAC	Virtual terminal inactive
GS_ANGL	Angle not valid
GS_NMEM	Insufficient resources
GS_AEND	End point not valid
GS_ASTR	Start point not valid.

Example

1. To draw several ellipses at different angles around a center point, the **arc2.c** C language program uses the **gsearc** subroutine in a loop.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gscrca** subroutine, **gscarc** subroutine, **gseara** subroutine.

gsecnv Subroutine

Purpose

Converts an ellipse to a polyline.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsecnv_ (*cx*, *cy*, *Major*, *Minor*, *Ang*, *bx*, *by*, *ex*, *ey*, *Rot*, *Len*, *x*, *y*, *Pre*)
int **cx*, **cy*, **Major*, **Minor*;
int **Ang*, **bx*, **by*, **ex*, **ey*;
int **Rot*, **Len*, *x*[ ], *y*[ ], **Pre*;

FORTRAN Syntax

INTEGER function gsecnv (*cx*, *cy*, *Major*, *Minor*, *Ang*, *bx*, *by*, *ex*, *ey*, *Rot*, *Len*, *x*, *y*, *Pre*)
INTEGER *cx*, *cy*, *Major*, *Minor*, *Ang*, *bx*, *by*, *ex*, *ey*, *Rot*, *Len*, *x*(*), *y*(*), *Pre*

Pascal Syntax

FUNCTION gsecnv_ (
VAR *cx*, *cy*, *Major*, *Minor*, *Ang*, *bx*, *by*, *ex*, *ey*, *Rot*, *Len*: **INTEGER;**
VAR *x*, *y*: **ARRAY [1..***k***] of INTEGER;**
VAR *Pre*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsecnv** subroutine converts a counterclockwise elliptical arc definition into an array of vertices. The list of vertices can then be used to draw an elliptical arc with the **gspoly** subroutine or to fill an elliptical arc with the **gsfply** subroutine. In general, it can be concatenated with other lists of vertices to draw or fill more complex shapes, such as chord arcs, pie arcs, or rectangles with round corners.

When the beginning and ending points are identical, the list of vertices contains the full ellipse, which can then be drawn or filled.

Parameters

cx, cy	Define the coordinates of the center of an ellipse.
Major, Minor	Define the semi–major and semi–minor axes of an ellipse.
Ang	Defines the off–axis angle of an ellipse. If the value of the *Ang* parameter is 0 (zero), the major axis is the X axis and the minor axis is the Y axis. A positive value rotates the ellipse counterclockwise; a negative value rotates it clockwise. All values are in degrees and modulo 360.
bx, by	Define the coordinates of the beginning point of an arc.
ex, ey	Define the coordinates of the ending point of an arc.

Note: The subroutine allows ample leniency toward the accuracy of the specification of beginning and ending points. The arc of the specified angle always starts and ends exactly at the specified points. If the beginning and ending points are identical, a full ellipse of the specified angle is generated.

Rot Specifies whether the application must perform rotational transformation. Possible settings are:

0 The coordinates of the beginning and ending points passed by the application correspond to an arc of an orthogonal ellipse. No rotational transformation is performed, thus improving performance.

1 The beginning and ending points are transformed by the application and lie on the off–axis ellipse.

Len Specifies, on return, the number of points (vertices) in the x and y coordinate arrays. If error conditions arise, the *Len* parameter is set to a value of 0. Before you call the **gsecnv** subroutine, you must set the value of the *Len* parameter to at least one greater than the value of the *Pre* (precision) parameter.

x, y Define, as coordinate arrays, the vertices that represent the elliptical shape when drawn or filled.

For Pascal syntax, the application must declare that the x and y arrays are fixed–length and that the **gsecnv** subroutine accepts arrays of that length. That is, the *k* in the subroutine declaration must be a constant.

Pre Defines the precision level, which specifies the maximum number of line segments that can be generated for a full ellipse. The number of line segments actually generated depends on the arc length defined by the beginning point (*bx,by*) and the ending point (*ex,ey*).

The *Pre* parameter must be one of the following four values, which specify the corresponding number of vertices:

- 64 (65 vertices)
- 128 (129 vertices)
- 256 (257 vertices)
- 512 (513 vertices).

All other precision values are reserved and must not be used, as their results are unpredictable. The default value for the *Pre* parameter is 64.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_CORD Coordinate not valid

GS_NCOR Number of coordinates not valid.

Example

1. To convert an elliptical arc to a polyline, the **arc5.c** C language program uses the **gsecnv** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gspoly** subroutine, **gsfply** subroutine.

gsecur Subroutine

Purpose

Erases the enabled cursor and makes it invisible.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsecur_ ( )

FORTRAN Syntax

INTEGER function gsecur_

Pascal Syntax

FUNCTION gsecur_: INTEGER [PUBLIC];

Description

The **gsecur** subroutine makes the enabled cursors invisible.

For adapters with hardware cursor support, the **gsecur** subroutine turns off the cursor. Otherwise, this subroutine reverses the actions that placed the cursor in the frame buffer.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_INAC	Virtual terminal inactive.

Example

1. To make the enabled cursor invisible before changing the cursor pattern, the **curs.c** C language program uses the **gsecur** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsmcur** subroutine.

gsell Subroutine

Purpose

Draws an ellipse.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsell_ (*cx*, *cy*, *Major*, *Minor*, *Ang*)
int **cx*, **cy*, **Major*, **Minor*, **Ang*;

FORTRAN Syntax

INTEGER function gsell_ (*cx*, *cy*, *Major*, *Minor*, *Ang*)
INTEGER *cx*, *cy*, *Major*, *Minor*, *Ang*

Pascal Syntax

FUNCTION gsell_ (
VAR *cx*, *cy*, *Major*, *Minor*, *Ang* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsell** subroutine draws an ellipse of the specified axes and angle. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsell** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Linestyle
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of an ellipse.
Major, Minor	Define half of the nonzero major and minor axes of an ellipse.
Ang	Defines the angle between the major axis and the X axis. If this angle is 0 (zero), the major axis is on the X axis and the minor axis is on the Y axis. The angle is expressed in tenths of degrees, from 0 to 3600.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_ELMM	Major or minor axis not valid
GS_INAC	Virtual terminal inactive
GS_ANGL	Angle not valid
GS_NMEM	Insufficient memory.

Example

1. To draw an ellipse, the **ell1.c** C language program uses the **gsell** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsepik Subroutine

Purpose

Provides compatibility for GSL applications that use the **gsepik** subroutine.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsepik_ (*Pickwind*)
int **Pickwind*;

FORTRAN Syntax

INTEGER function gsepik_ (*Pickwind*)
INTEGER *Pickwind*

Pascal Syntax

FUNCTION gsepik_ (
VAR *Pickwind* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsepik** subroutine is provided for compatibility with existing GSL applications. It is ignored by the current implementation of XGSL.

Parameter

The parameter is ignored. It is provided for compatibility.

Return Value

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return value:

GS_SUCC Successful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gseply Subroutine

Purpose

Defines the end of an area to fill.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gseply_ ( )

FORTRAN Syntax

INTEGER function gseply

Pascal Syntax

FUNCTION gseply_ : INTEGER [PUBLIC];

Description

The **gseply** subroutine defines the end of a two-dimensional shape or set of shapes to be filled. The subroutine then fills each of the valid primitives, or basic graphic elements, drawn since the last **gspcls** or **gsbply** subroutine was called.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern
- Logical operation.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_USUC	Unsuccessful.

Example

1. To define the end of an area to be filled, the **blit.c** C language program uses the **gseply** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsbply** subroutine, **gspcls** subroutine.

gsevds Subroutine

Purpose

Disables the reporting of events.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsevds_ (*Event*)
int **Event*;

FORTRAN Syntax

INTEGER function gsevds_ (*Event*)
INTEGER *Event*

Pascal Syntax

FUNCTION gsevds_ (
VAR *Event*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsevds** subroutine disables the reporting of events of a given type. When the keyboard event is disabled, the keyboard is locked and no keystroke input is placed in the input ring buffer. Similarly, for all other devices, if an event is disabled, the device producing the event is inhibited from placing input into the ring.

A valid input ring must be defined during the XGSL initialization.

Parameter

Event Specifies the event to disable.

The recognized events and their values are as follows:

Value	Event
1	Keystroke
3	Locator movement or button
4	Lighted programmable function key (LPFK)
5	Valuator.

The user can enable the keyboard by pressing the Esc-B key sequence (ANSI enable manual input). After this sequence, keystroke events are again reported.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_EVNT Event type not valid

GS_SUCC Successful

GS_UNSC Unsuccessful.

Example

1. To disable mouse input, the **curs.c** C language program uses the **gsevds** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gseven** subroutine, **gsinit** subroutine, **gsevwt** subroutine.

gseven Subroutine

Purpose

Enables the reporting of events.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gseven_ (*Event*)
int **Event*;

FORTRAN Syntax

INTEGER function gseven (*Event*)
INTEGER *Event*

Pascal Syntax

FUNCTION gseven_ (
VAR *Event*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gseven** subroutine enables the reporting of events of a given type. If the device producing the event is enabled, the **gseven** subroutine allows it to put data into the ring buffer. If the event type is not recognized, no action is taken.

A valid input ring must be defined during the XGSL initialization.

Parameter

Event Specifies the event to disable.

The recognized events and their values are as follows:

Value	Event
1	Keystroke
3	Locator movement or button
4	Lighted programmable function key (LPFK)
5	Valuator.

After XGSL initialization, only the keyboard is enabled. If the application requires the other input devices to be enabled, it must explicitly enable them with this command.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values.

GS_EVNT	Event type not valid
GS_SUCC	Successful
GS_UNSC	Unsuccessful.

Example

1. To enable keystroke events to put data into the ring buffer, the **curs.c** C language program uses the **gseven** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsevds** subroutine, **gsinit** subroutine, **gsevwt** subroutine.

gsevwt Subroutine

Purpose

Waits for an input event.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsevwt_ (*Wait*, *Data***)**
int ****Wait*, *Data*[13];**

FORTRAN Syntax

INTEGER function gsevwt_ (*Wait*, *Data***)**
INTEGER *Wait*, *Data* **(13)**

Pascal Syntax

```
FUNCTION gsevwt_ (
VAR Wait: INTEGER;
VAR Data: ARRAY [0..12] of INTEGER
): INTEGER [PUBLIC];
```

Description

The **gsevwt** subroutine returns the relevant information for the oldest input event in the ring buffer.

If an event is in the ring buffer, then the **gsevwt** subroutine parses the oldest event in the ring buffer. It returns the event type and its data in the buffer provided by the application. If the return code indicates overflow, the most recent input events from enabled devices are lost.

If no event is in the ring and the application does not request a wait, the **gsevwt** subroutine returns immediately. If the application requested a wait, the process execution is suspended until an enabled input event occurs. Then the **gsevwt** subroutine returns the event type and its data in the buffer specified by the *Data* parameter.

Note: The **gsevwt** subroutine uses the application buffer passed to it for temporary storage. If the user has explicitly keyed part of an ANSI control sequence when the application calls the **gsevwt** subroutine with no wait request, then the **gsevwt** subroutine finds a partial event in the ring and leaves part of the parsed data for the event in the application buffer. However, the **gsevwt** subroutine returns a time-out event class. Unless the application returns the same unmodified buffer, or a different buffer containing identical information, the results of the next call to the **gsevwt** subroutine will be incorrect.

A valid input ring must be defined during the XGSL initialization.

Parameters

Wait	Determines whether to wait for an event. If the *Wait* parameter has a value of 0, then the **gsevwt** subroutine does not wait for an event if no event is available.
Data	Specifies the location where XGSL is to store the input data (up to 13 words). The *Data* parameter must be word-aligned.

The possible events are:

- **Keystroke**

  This event type occurs when the user types a single graphic character or a single-byte control character. For these two events, the **gsevwt** subroutine returns a null-terminated byte string of ASCII codes representing the graphic or control characters that were typed. This event may also occur if the user has explicitly keyed an ANSI escape sequence. If so, the **gsevwt** subroutine returns 2 bytes: the Esc character and the next character in the sequence.

  The *Data* parameter consists of a null-terminated ASCII string and is structured as shown in Figure 1.

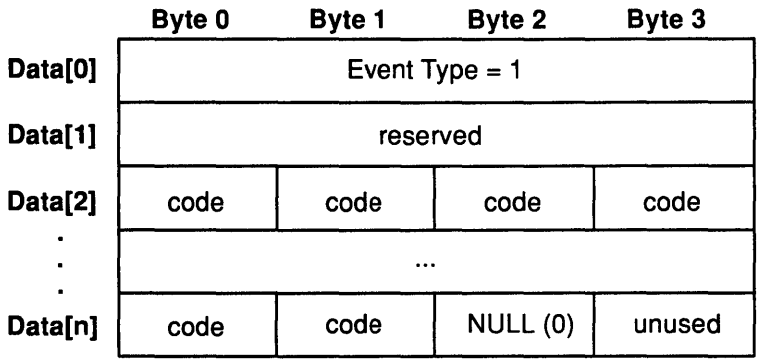


Figure 42. Structure of the *Data* Parameter for a Keystroke Event

  It is important to note that the **gsevwt** subroutine does not detect ANSI escape sequences. However, with the default virtual-terminal keyboard mapping, it is not possible to generate an escape sequence by pressing a single key. Because the **gsevwt** subroutine does parse ANSI control sequences, the routine cannot consider the press of the Esc key an event, so the routine waits for the next character to decide if the escape implies the start of a control sequence. Only if the next character is not the left bracket does the **gsevwt** subroutine return the Esc character and the next character.

- **Control sequence**

  This event type indicates an ANSI control sequence, which is of the form:

  `Esc [ p ; p ; ... p f`

  where `Esc` is the ASCII escape character, `p` represents any parameters (one or more ASCII digits), the `...` (ellipses) represent additional parameters separated by semicolons, and `f` represents the final character that terminates the sequence (ASCII a to z or A to Z).

  The ANSI control sequence occurs when the user either presses a programmable function key on the keyboard or enters an explicit control sequence.

The *Data* parameter is structured as shown in Figure 2.

Data[0]	Event Type = 2
Data[1]	Final Character
Data[2]	Count
Data[3]	Parameter[1]
.	...
.	
.	
Data[n]	Parameter[Count]

Figure 43. Structure of the *Data* Parameter for a Control Sequence Event

The data consists of the parsed control-sequence information. The Final Character field is the valid or invalid final character. The Count field indicates the number of parameters in the control sequence, with a maximum count of 10. These fields are followed by the Parameter fields.

- **Locator**

  This event indicates that the user has moved the locator or pressed a button on the locator.

  The *Data* parameter consists of locator position and status information as shown in Figure 3.

Data[0]	Event Type = 3
Data[1]	X Value
Data[2]	Y Value
Data[3]	Type
Data[4]	Buttons
Data[5]	Time Stamp

Figure 44. Structure of the *Data* Parameter for a Locator Event

The X Value and Y Value fields contain an absolute position (x, y) for a tablet. The Type field contains a 1 if the locator is a tablet.

The Buttons field contains the locator button status. A bit set to 1 indicates that the corresponding button is pressed. For a tablet, the following bits, when set, indicate that the corresponding buttons are pressed (bit 0 is the most significant bit):

Bit	Button
0	None pressed
1	Cursor upper left, stylus tip
2	Cursor upper right
3	Cursor lower left
4	Cursor lower right.

The sixth most significant bit of the Buttons field (bit 5) indicates that the tablet sensor is on (bit set) or off (bit not set).

- **LPFK**

  This event type occurs when the user presses a key on the LPFK. The *Data* parameter consists of the LPFK information as shown in Figure 4.

Data[0]	Event Type = 4
Data[1]	LPFK
Data[2]	Time Stamp

Figure 45. Structure of the *Data* Parameter for a LPFK Event

The LPFK field contains the decimal number of the LPFK pressed by the user, that is, 0 through 31. The Time Stamp field is elapsed time in sixtieths of a second.

- **Valuator**

  This event type occurs when the user turns a valuator dial. The *Data* parameter consists of the valuator information as shown in Figure 5.

Data[0]	Event Type = 5
Data[1]	Valuator
Data[2]	Valuator Delta
Data[3]	Time Stamp

Figure 46. Structure of the *Data* Parameter for a Valuator Event

The Valuator field contains the decimal number, 0 through 7, of the valuator turned by the user. The Valuator Delta field contains the difference between the current valuator value and the last valuator value. The delta for a full turn is 256 for the IBM Valuator. The delta is positive for clockwise rotation and negative for counterclockwise rotation. The Time Stamp field is elapsed time in sixtieths of a second.

- **Key code**

  This event type occurs when the virtual terminal is in nontranslated mode *and* a keyboard key is pressed, held down, or released. The *Data* parameter is structured as shown in Figure 6.

Data[0]	Event Type = 6
Data[1]	Key Position Code
Data[2]	Key Scan Code
Data[3]	Status

Figure 47. Structure of the *Data* Parameter for a Key Code Event

Information on key position codes is contained in the workstation keyboard mapping table.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_ICTL Final character not valid

GS_PARM Too many control sequence parameters

GS_ROVR Ring buffer overflow

GS_SUCC Successful

GS_UDRG Ring buffer undefined.

If the return code indicates overflow, the most recent input events from enabled devices are lost.

Example

1. To instruct the process to wait for an input event, the **curs.c** C language program uses the **gsevwt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsevds** subroutine, **gseven** subroutine, **gsinit** subroutine.

gsfatt Subroutine

Purpose

Sets the fill attributes.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsfatt_ (*Color*, *Pattern*, *Reserved*)
int **Color*, **Pattern*, **Reserved*;

FORTRAN Syntax

INTEGER function gsfatt_ (*Color*, *Pattern*, *Reserved*)
INTEGER *Color*, *Pattern*, *Reserved*

Pascal Syntax

FUNCTION gsfatt_ (
VAR *Color*, *Pattern*, *Reserved*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfatt** subroutine defines the attributes for the class of fill functions, which includes the **gsfci**, **gsfell**, **gsfrec**, and **gsfply** subroutines.

Parameters

Color Specifies the fill color. This refers to an entry in the color map. If the *color* parameter has a value of –1, the attribute is unchanged. The default color after initialization is 15.

Pattern Specifies the fill pattern. Use the following values to specify the corresponding fill patterns:

Value	Display
–1	No change
0	Solid
1	Horizontal lines
2	Vertical lines
3	135-degree lines
4	45-degree lines
5	Cross-hatched (horizontal and vertical lines)
6	Cross-hatched (45- and 135-degree lines)

The default fill pattern is solid (0).

Reserved Represents a parameter that the **gsfatt** subroutine ignores.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values.

GS_SUCC	Successful
GS_COLI	Color index not valid
GS_SYLI	Style index not valid.

Example

1. To set the fill attributes for a text box, the C language program **curs.c** uses the **gsfatt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsfply** subroutine, **gsfrec** subroutine, **gsfci** subroutine, **gsfell** subroutine.

gsfci Subroutine

Purpose

Fills a circle.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsfci_ (*cx*, *cy*, *cr*)
int **cx*, **cy*, **cr*;

FORTRAN Syntax

INTEGER function gsfci_ (*cx*, *cy*, *cr*)
INTEGER *cx*, *cy*, *cr*

Pascal Syntax

FUNCTION gsfci_ (
VAR *cx*, *cy*, *cr* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfci** subroutine fills a circle of a specified radius. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of a circle.
cr	Defines the radius of a circle. If the radius is 0 (zero), a single point is filled at the center.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_RDUS	Radius specification not valid
GS_INAC	Virtual terminal inactive.

Example

1. To fill the defined circle, the **cir2.c** C language program uses the **gsfci** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gscir** subroutine, **gsfatt** subroutine.

gsfell Subroutine

Purpose

Fills an ellipse.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsfell_ (*cx*, *cy*, *Major*, *Minor*, *Ang*)
int **cx*, **cy*, **Major*, **Minor*, **Ang*;

FORTRAN Syntax

INTEGER function gsfell_(*cx*, *cy*, *Major*, *Minor*, *Ang*)
INTEGER *cx*, *cy*, *Major*, *Minor*, *Ang*

Pascal Syntax

FUNCTION gsfell_ (
VAR *cx*, *cy*, *Major*, *Minor*, *Ang* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfell** subroutine fills an ellipse of the specified axes and angle. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsfell** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of an ellipse.
Major, Minor	Define half of the nonzero major and minor axes of an ellipse.
Ang	Defines the angle between the major axis and the X axis. If it is zero, the major axis is on the X axis and the minor axis is on the Y axis. The angle is defined in tenths of degrees, from 0 to 3600, specified in a counterclockwise direction.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_ELMM	Major or minor axis not valid
GS_INAC	Virtual terminal inactive
GS_ANGL	Angle not valid
GS_NMEM	Insufficient memory.

Example

1. To fill the defined ellipse, the **ell2.c** C language program uses the **gsfell** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsell** subroutine, **gsfatt** subroutine.

gsfply Subroutine

Purpose

Draws a filled polygon.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsfply_ (*Number, x, y*)
int **Number, x[ ], y[ ];*

FORTRAN Syntax

INTEGER function gsfply_ (*Number, x, y*)
INTEGER *Number*
INTEGER *x* (*), *y* (*)

Pascal Syntax

FUNCTION gsfply_ (
VAR *Number*: **INTEGER;**
VAR *x, y*: **ARRAY [1..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfply** subroutine fills an area that is described by the points defined in the *Number* and *x* and *y* parameters. The last call to the **gsfatt** subroutine determines the fill color.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern
- Logical operation.

The edges are treated as part of the area to be filled.

Parameters

Number	Defines the number of points in the coordinate arrays. This value must be 3 or more.
x, y	Define the points surrounding the polygon to fill. The points are defined as coordinate arrays. For Pascal syntax, the application must declare that the x and y arrays are fixed-length and that the **gsfply** subroutine accepts an array of that length. That is, the *k* in the routine declaration must be a constant and should be greater than or equal to the largest value of the *Number* parameter.

The **gsfply** subroutine fills a closed polygon with a pattern. If the polygon is not already closed (if the first and last points are not equal), the **gsfply** subroutine generates the polygon by creating an edge between the first and last points. The first and last points described by the parameters can be equal, but this condition is not required and is actually less efficient.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_NCOR	Number of coordinates not valid
GS_NMEM	Insufficient memory
GS_INAC	Virtual terminal inactive.

Example

1. To fill a defined closed polygon, the **blit.c** C language program uses the **gsfply** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsfatt** subroutine, **gsfrec** subroutine.

gsfrec Subroutine

Purpose

Draws a filled rectangle.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsfrec_ (*x1*, *y1*, *x2*, *y2*)
int **x1*, **y1*, **x2*, **y2*;

FORTRAN Syntax

INTEGER function gsfrec_ (*x1*, *y1*, *x2*, *y2*)
INTEGER *x1*, *y1*, *x2*, *y2*

Pascal Syntax

FUNCTION gsfrec_ (
VAR *x1*, *y1*, *x2*, *y2*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfrec** subroutine fills the rectangular area defined by the lower left and upper right coordinate parameters. The last call to the **gsfatt** subroutine determines the fill color.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern
- Logical operation.

Parameters

x1, *y1*	Define the lower left corner of the rectangular area to fill.
x2, *y2*	Define the upper right corner of the rectangular area to fill.
	The edges of the rectangle are treated as part of the area to be filled.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values.

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_INAC	Virtual terminal inactive.

Example

1. To draw a filled rectangle, the **curs.c** C language program uses the **gsfrec** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsfatt** subroutine, **gsfply** subroutine.

gsgtat Subroutine

Purpose

Sets the attributes for the geometric text-drawing functions.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsgtat_ (*Color, Baseline, Pre, Expan, Spac, Height, Upvectx, Upvecty, Alignhz, Alignvt, FontID, Font*)
int **Color*, **Baseline*, **Pre*;
int **Expan*, **Spac*, **Height*;
int **Upvectx*, **Upvecty*;
int **Alignhz*, **Alignvt*, **FontID*;
char **Font*;

FORTRAN Syntax

INTEGER function gsgtat (*Color, Baseline, Pre, Expan, Spac, Height, Upvectx, Upvecty, Alignhz, Alignvt, FontID, Font*)
INTEGER *Color, Baseline, Pre*
INTEGER *Expan, Spac, Height*
INTEGER *Upvectx, Upvecty*
INTEGER *Alignhz, Alignvt, FontID*
CHARACTER*n* *Font*

Pascal Syntax

FUNCTION gsgtat_ (
VAR *Color, Baseline, Pre, Expan, Spac, Height*: **INTEGER;**
VAR *Upvectx, Upvecty, Alignhz, Alignvt, FontID*: **INTEGER;**
VAR *Font*: **ARRAY [0..***k***] of CHAR**
): INTEGER [PUBLIC];

Description

The **gsgtat** subroutine defines the attributes and fonts for the geometric text-drawing functions.

Notes:

1. This subroutine must be called before the **gsgtxt** subroutine, or an error results.
2. The attributes defined by this command are applicable only to geometric text.
3. Attributes are valid only for the currently active font.

Parameters

Color — Specifies an entry in the color map for text color. If the value is –1 (negative one), the attribute is unchanged.

Baseline — Determines the direction of the geometric text drawing. The valid values are:

–1 Attribute remains unchanged

0 Specifies 0 degrees, or left to right in the viewer's terms

1 Specifies 90 degrees, or *up* in the viewer's terms

2 Specifies 180 degrees, or right to left in the viewer's terms

Note: The characters appear upside down.

3 Specifies 270 degrees, or *down* in the viewer's terms.

Note: The *Baseline* parameter does not change character rotation. Use the *Upvectx* and *Upvecty* parameters to rotate text.

Pre Specifies the desired text precision used in drawing text primitives. The valid values are:

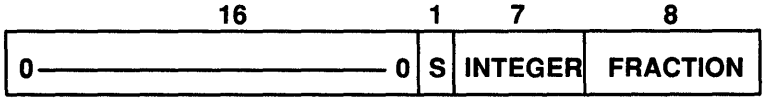
–1 Attribute remains unchanged

1 Character precision

2 Stroke precision.

Expan Defines as a 32-bit fractional integer the deviation of the width/height ratio of the character from the ratio defined in the font. The expansion factor changes only the width of the character.

In the following figure, the most significant 16 bits contain zeros, S represents the sign bit, INTEGER represents the integer portion of the width/height ratio, and FRACTION represents the fractional portion of the ratio.



A 32-bit integer value of 0x8000000 indicates that this attribute is unchanged.

Spac Specifies the character spacing, or additional number of pixels to be inserted between characters. The value is a 16-bit signed integer. The preferred value for this parameter varies, based on the display in use. The maximum value that is allowed is equal to the display width in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged.

Height Specifies the current character height for geometric text in pixels. This value is defined as a 16-bit signed integer, with the maximum value equal to the height of the display in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged.

Upvectx, Upvecty

Specifes the x and y coordinates for the up direction of a character or text string. The valid range for these values is plus or minus the display's dimensions in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged.

The up vector is a two-dimensional vector on the text plane, specified by the current text draw. (The origin of the vector is defined by the **gsgtxt** subroutine.) Only the direction, not the length, of the vector is relevant.

Alignhz Specifies the horizontal alignment of the text for subsequent text drawing. Values are as follows:

–1 Attribute is unchanged

1 Normal

2 Left

3 Center

4 Right.

Alignvt Specifies the vertical alignment of the text for subsequent text drawing. Values are as follows:

–1 Attribute is unchanged

1 Normal

2 Top

3 Cap

4 Half

5 Base

6 Bottom.

FontID Selects a new active geometric font. The specified font ID is a 32-bit integer that also indicates the type of font (either one- or two-byte). The font ID is assigned when the geometric font is designed. Possible values are:

–1 Leaves the active font unchanged. To use this value, a font ID must be selected in a previous call to the **gsgtat** subroutine.

1025 to **32767**
These values are used to specify one-byte geometric fonts, and refer to a value defined in each geometric font file.

32768 to **65535**
These values are used to specify two-byte geometric fonts, and refer to a value defined in each geometric font file.

Only one geometric font is active at any time. To change the font, the **gsgtat** subroutine must be called again with new *FontID* and *Font* parameters. When a new font is specified, the previous font is purged from the font table.

For two-byte geometric text, up to 128 segment IDs can be used per font ID.

The font ID is associated with the selected font and determines if the font is a one- or two-byte font.

Font Contains the null-terminated, full path name of the file used when the font attribute is specified by the user. If a nonzero value for the *FontID* parameter is specified, this parameter must also be specified.

For Fortran syntax, the application must declare that the *Font* parameter is fixed-length. That is, the value for the *n* variable in the routine declaration must be a constant

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the value of the *k* variable in the routine declaration must be a constant.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_COLI	Color index not valid
GS_PREC	Text precision value not valid
GS_EXPN	Character expansion factor not valid
GS_FNTN	File name not valid
GS_INSV	Spacing value not valid
GS_BASL	Baseline direction not valid
GS_HIGH	Height value not valid
GS_UPVT	Up vector value not valid
GS_ALGN	Alignment value not valid.

Example

1. To set the attributes and fonts for geometric text-drawing, the **gtex.c** C language program uses the **gsgtat** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsgtxt** subroutine.

gsgtxt Subroutine

Purpose

Writes geometric text.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsgtxt_ (*x*, *y*, *Number*, *Text*)
int **x*, **y*, **Number*;
char *Text*[ ];

FORTRAN Syntax

INTEGER function gsgtxt (*x*, *y*, *Number*, *Text*)
INTEGER *x*, *y*, *Number*
CHARACTER*n *Text*

Pascal Syntax

FUNCTION gsgtxt_ (
VAR *x*, *y*, *Number*: **INTEGER;**
VAR *Text*: **ARRAY [1..***k***] of CHAR**
): INTEGER [PUBLIC];

Description

The **gsgtxt** subroutine writes geometric characters starting at the baseline position defined by the *x* and *y* parameters. It writes the number of characters indicated by the *Number* parameter according to the relevant attributes.

The relevant attributes in the following list must first be set by the **gsgtat** subroutine:

- Color map
- Plane mask
- Geometric text font
- Geometric text color index
- Character expansion factor
- Character spacing
- Character height
- Character up vector
- Character alignment
- Baseline direction.

Note: If the **gsgtat** subroutine is not called before the **gsgtxt** subroutine, an error results.

Parameters

x, y	Define the coordinates of the baseline position for writing geometric text.
Number	Indicates the number of bytes to write from the *text* string. The maximum number of characters allowed, which is determined by the display and font in use, is 1024 for single-byte fonts and 512 for double-byte fonts.

Text An array that contains the N-bit ASCII codes for the characters to write.

For FORTRAN syntax, the application must declare that the *Text* parameter is fixed-length. That is, the value for the *n* variable in the routine declaration must be a constant The value for the *n* variable should be at least as large as the *Number* parameter.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the value for the *k* variable in the routine declaration must be a constant The value for the *k* variable should be at least as large as the *Number* parameter.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_CORD Coordinate not valid

GS_FBUF Frame buffer overflow

GS_INAC Virtual terminal inactive

GS_NOFT Font not loaded.

Example

1. To write geometric text characters, the **gtex.c** C language program uses the **gsgtxt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsqgtx** subroutine, **gsgtat** subroutine.

gsinit Subroutine

Purpose

Initializes the XGSL subroutines.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsinit_ (*Buffer*, *Size*, *SaveRestore*, *FGrant*, *FRetract*, *OutDev*)
int **Buffer*, **Size*, **SaveRestore*;
int (**FGrant*) ( ), (**FRetract*) ( );
int **OutDev*;

FORTRAN Syntax

INTEGER function gsinit_ (*Buffer*, *Size*, *SaveRestore*, *FGrant*, *FRetract*, *OutDev*)
INTEGER *Buffer* (*), *Size*, *SaveRestore*, *OutDev*
EXTERNAL *FGrant*, *FRetract*

Pascal Syntax

FUNCTION gsinit_ (
VAR *Buffer*: **ARRAY [0..***k***]k of INTEGER;**
VAR *Size*, *SaveRestore*, *FGrant*, *FRetract*, *OutDev*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsinit** subroutine initializes the XGSL. It allocates any private storage required and sets attributes to the default values, where necessary. It also forces the virtual terminal of the application into Monitor mode and sets up the signal-processing routines for the **SIGRETRACT**, **SIGGRANT**, and **SIGMSG** signals.

Parameters

Buffer — Defines the Monitor mode input ring buffer to be used by the XGSL input functions. The *Buffer* parameter must be word-aligned and at least 128 bytes long.

For Pascal syntax, the application must declare that the *Buffer* array is fixed-length and that the **gsinit** subroutine accepts an array of that length. That is, the value of the *k* variable in the routine declaration must be a constant.

Pascal cannot directly provide the address of a routine. An assembler function can be used to derive the address of a routine passed to the XGSL.

Size — Defines the length of the *Buffer* parameter in bytes. Depending on the value of the *Size* parameter, the **gsinit** subroutine performs the following actions:

Size <128	The **gsinit** subroutine does not initialize the XGSL.
Size >=128	The XGSL establishes the virtual terminal linkage to the input ring buffer provided by the application, provides input support, and sets up a **SIGMSG** signal catcher.

SaveRestore Determines whether to save the display frame buffer and adapter state.

If the *SaveRestore* parameter is nonzero, the XGSL saves the current contents of the display frame buffer as well as the current adapter state when the virtual terminal becomes inactive, and then restores both the frame buffer contents and adapter state when it becomes active.

If the *SaveRestore* Parameter is zero, the XGSL saves only the adapter state and assumes that the application either saves the frame buffer or reconstructs it in some fashion.

FGrant Sets up processing of the **SIGGRANT** signal. If *FGrant* is nonzero, it is assumed to be the address of an application-supplied function, and the XGSL calls the function as part of the **SIGGRANT** signal handling. If the *SaveRestore* parameter is nonzero, this function is called before the XGSL restores the frame buffer. This routine might be called before the **gsinit** subroutine returns to the application.

FRetract Sets up processing of the **SIGRETRACT** signal. If the *FRetract* parameter is nonzero, it is assumed to be the address of an application-supplied function, and the XGSL calls the function as part of the **SIGRETRACT** signal handling.

OutDev This parameter is ignored by XGSL. It is provided for compatibility.

If the initialization process is unsuccessful, the virtual terminal is not placed in monitor mode and invocation of any other XGSL routines will cause unpredictable results.

The *FGrant* and *FRetract* routines supplied by the application are called on the signal level and must return. These application routines must not use either the **setjmp** or **longjmp** subroutines.

Return Values

If you use the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_HBUS Cannot access hardware bus

GS_ADPT Display type not valid

GS_FONT Cannot access default font

GS_RING Ring Buffer too small

GS_HDCP File descriptor for hard-copy output not valid

GS_HDLK Unable to create lock file

GS_HDIM Insufficient memory

GS_HDDB Device is busy

GS_HDNA Physical device not attached

GS_HDMG Maximum number of graphics devices open

GS_HDIF No system interprocess communication buffers left

GS_HDSF	The **fork** system call failed
GS_HDGO	Specified graphics device already open
GS_HDGN	Specified graphics device does not exist
GS_HDGU	Specified graphics device driver is unknown.

Example

1. To initialize the XGSL and set up a ring buffer for input, the **arc1.c** C language program uses the **gsinit** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsterm** subroutine.

gslatt Subroutine

Purpose

Sets the line attributes.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gslatt_ (*Color*, *Style***)**
**int ****Color*, ******Style*;

FORTRAN Syntax

INTEGER function gslatt_ (*Color*, *Style***)**
INTEGER *Color*, *Style*

Pascal Syntax

FUNCTION gslatt_ (
VAR *Color*, *Style*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gslatt** subroutine defines the attributes for the class of line-drawing functions.

Parameters

Color Specifies the line color. This refers to a line color entry in the color map. If this entry is –1, the attribute is unchanged. The default color is 15.

Style Specifies the line style. Use the following values to specify the corresponding line style:

Value	Display
–1	No change
0	Solid
1	Dash
2	Dot
3	Dash-dot
4	Dash-dot-dot
100	Continuous solid
101	Continuous dash
102	Continuous dot
103	Continuous dash-dot
104	Continuous dash-dot-dot
105	Continuous user-supplied.

The default style is solid (0).

The line style patterns supplied by the AIXwindows Graphics Support Library (XGSL) are implemented in a device-dependent fashion. All line style indices not described above are reserved.

For line styles 1 to 99, the XGSL line-drawing functions ensure that a line or line segment starts and ends with a run of the line color. For example, the XGSL does not continue the pattern from one polyline segment to another.

For line styles 100 to 150, the XGSL continues the pattern across multiple lines or line segments until the application makes another call to the **gslatt** subroutine to reset the line pattern. In this case, unlike for styles 1 to 99, the XGSL continues the pattern from one polyline segment to another.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_COLI	Color index not valid
GS_SYLI	Style index not valid.

Example

1. To set the line color to a specific color map entry, but leave the line style unchanged, the **arc1.c** C language program uses the **gslatt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gslop** subroutine, **gsline** subroutine, **gsmult** subroutine, **gsdjply** subroutine, **gspoly** subroutine.

gslcat Subroutine

Purpose

Sets the locator attributes.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gslcat_ (*hg*, *vg*)
int **hg*, **vg*;

FORTRAN Syntax

INTEGER function gslcat (*hg*, *vg*)
INTEGER *hg*, *vg*

Pascal Syntax

FUNCTION gslcat_ (
VAR *hg*, *vg*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gslcat** subroutine sets the locator dead zone.

The tablet dead zone is an area of the tablet in which no event reports occur, even if the tablet sensor is present. This dead zone allows the application to make the tablet aspect ratio compatible with the display, and allows tablets of different sizes to appear the same size to an application. The dead zone acts as a border around the tablet. The device driver reports movement only when the x value is greater than or equal to the value of the *hg* parameter, or less than or equal to the maximum tablet value minus the value of the *hg* parameter and the y value is greater than or equal to the value of the *vg* parameter or less than or equal to the maximum tablet value minus the value of the *vg* parameter.

An attempt to set the locator attributes may fail for a variety of reasons, the most likely of which is that the device is not attached.

Note: The **gslcat** subroutine allows an application to set the tablet dead zone so that no events occur even if the device is enabled.

Parameters

hg, vg	Define the horizontal and vertical values for the locator dead zone, in units of 0.25 millimeter.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values.

GS_SUCC	Successful
GS_USUC	Unsuccessful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsline Subroutine

Purpose

Draws a line between two points.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsline_ (*x1*, *y1*, *x2*, *y2*)
int **x1*, **y1*, **x2*, **y2*;

FORTRAN Syntax

INTEGER function gsline (*x1*, *y1*, *x2*, *y2*)
INTEGER *x1*, *y1*, *x2*, *y2*

Pascal Syntax

FUNCTION gsline_ (
VAR *x1*, *y1*, *x2*, *y2*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsline** subroutine draws a line, as defined by the current relevant attributes, from the first point to the second point defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

x1, *y1*	Define the coordinates of the first end point of the line drawn by the **gsline** subroutine.
x2, *y2*	Define the coordinates of the second point of the line drawn by the **gsline** subroutine.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values.

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_INAC	Virtual terminal inactive.

Example

1. To draw the intersecting lines through the center of a series of ellipses, the **arc1.c** C language program uses the **gsline** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsdjply** subroutine, **gslop** subroutine, **gsmult** subroutine, **gspoly** subroutine, **gsulns** subroutine.

gslock Subroutine

Purpose

Provides compatibility for GSL applications that use the **gslock** subroutine.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gslock_( )

FORTRAN Syntax

INTEGER function gslock ( )

Pascal Syntax

FUNCTION gslock_( ): INTEGER [PUBLIC];

Description

The **gslock** subroutine is provided for compatibility with existing GSL applications. Under the current implementation of XGSL, the **gslock** subroutine always returns a value of **GS_INAC**, indicating that you cannot write to the display adapter.

Return Value

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return value:

GS_INAC Virtual terminal inactive.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gslop Subroutine

Purpose

Specifies the logical operation used when drawing lines.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gslop_ (*Operation*)
int **Operation*;

FORTRAN Syntax

INTEGER function gslop_ (*Operation*)
INTEGER *Operation*

Pascal Syntax

FUNCTION gslop_ (
VAR *Operation* **INTEGER;**
): INTEGER [PUBLIC];

Description

The **gslop** subroutine specifies the logical operation used for drawing the AIXwindows Graphics Support Library (XGSL) line-oriented, fill, save/restore, and polymarker primitives. The **gslop** subroutine does not apply to the text primitives.

Parameter

Operation Indicates the logical operation to perform between the primitive being drawn and the current contents of the frame buffer.

The following list of values for the *Operation* parameter specify the operations you can perform. The source represent bits of data to be merged in some way with the corresponding bits of data in the frame buffer (the destination).

Value	Operation
0	Clear destination
15	Set destination
5	No operation
10	Logical inverse of destination
3	REPLACE destination with source
1	AND source with destination
2	AND source with inverse of destination
6	Exclusive-OR source with destination
7	OR source with destination
11	OR source with inverse of destination
12	REPLACE destination with inverse of source

4	AND inverse of source with destination
8	AND inverse of source with inverse of destination
9	Exclusive-OR inverse of source with destination
13	OR inverse of source with destination

Notes:

1. REPLACE (3) is the default logical operation.
2. The XGSL provides the REPLACE and Exclusive-OR logical operations (codes 3 and 6 respectively) for all drawing routines. The **gsxblt** subroutine supports the full set of logical operations.
3. The XGSL performs each of the Boolean operations for each bit of the source and destination color values enabled by the plane mask. The destination receives the color value that results from the operation.
4. The logical operations are performed on the color index rather than the color itself. Therefore, the same operations on different color displays produce different results on each display.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_LONS	Logical operation not supported.

Examples

1. To make the **gsline** subroutine draw lines and the **gseara** subroutine draw ellipses by replacing the contents of the frame buffer, the **arc1.c** C language program uses the **gslop** subroutine.
2. To make the **gsearc** subroutine first draw and then erase an ellipse, the **arc2.c** C language program uses the **gslop** subroutine to set the logical operation to Exclusive-OR.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gseara** subroutine, **gsearc** subroutine, **gsfatt** subroutine, **gslatt** subroutine, **gsline** subroutine.

gslpat Subroutine

Purpose

Sets the lighted program function key (LPFK) indicators.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gslpat_ (*Indicators*)
int **Indicators*;

FORTRAN Syntax

INTEGER function gslpat (*Indicators*)
INTEGER *Indicators*

Pascal Syntax

FUNCTION gslpat_ (
VAR *Indicators*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gslpat** subroutine turns on or off the indicators on the Lighted Program Function Keyboard.

Parameter

Indicators Specifies the state of the LPFK indicators. Each bit of the *Indicators* parameter corresponds to an indicator on the LPFK, with the most significant bit setting the desired state (1 = on, 0 = off) for the indicator for LPFK 0, the next most significant bit setting the state for the indicator for LPFK 1, and so on.

The default state for all indicators is 0, or off.

If an attempt to set the LPFK indicators is unsuccessful, it may be that the device is not attached.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_USUC Unsuccessful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsevwt** subroutine.

gsmask Subroutine

Purpose

Defines planes to be modified.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsmask_ (*Mask***)**
int ****Mask*;**

FORTRAN Syntax

INTEGER function gsmask (*Mask***)**
INTEGER *Mask*

Pascal Syntax

FUNCTION gsmask_ (
VAR *Mask*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsmask** subroutine defines the planes actually modified by the line, text, and fill functions.

Parameter

Mask	Indicates which planes of the display adapter frame buffer can be modified by the output functions. The most significant bits of the input are used to set the plane mask.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_INAC	Virtual terminal inactive.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsmatt Subroutine

Purpose

Sets the polymarker attribute.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsmatt_ (*Color*, *Style*, *Width*, *Height*, *Pattern*, *0x*, *0y*)
int **Color*, **Style*, **Width*, **Height*;
int **Pattern*, **0x*, **0y*;

FORTRAN Syntax

INTEGER function gsmatt (*Color*, *Style*, *Width*, *Height*, *Pattern*, *0x*, *0y*)
INTEGER *Color*, *Width*, *Height*, *Pattern*, *0x*, *0y*

Pascal Syntax

FUNCTION gsmatt_ (
VAR *Color*, *Style*, *Width*, *Height*: **INTEGER;**
Pattern: **ARRAY [1..***k***] of INTEGER;**
0x, *0y*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsmatt** subroutine defines the marker for the AIXwindows Graphics Support Library (XGSL).

Parameters

Color — Refers to a marker color entry in the color map. If the value is –1, the attribute is unchanged. The default value for color is 7, white.

Style — Defines the polymarker *Style* as one of the following:

Value	Display
–1	No change
0	User-defined (by the *Width*, *Height*, *Pattern*, *Ox*, *Oy* parameters)
1	Dot (filled circle)
2	Plus (+)
3	Asterisk (*)
4	Circular shape
5	Cross (x)
6	Unfilled box

Width, Height — Define, in pixels, the width and height of the bit pattern to be used as the marker. If either the *Width* or *Height* parameter equals –1, then the pattern remains unchanged.

Dividing the value of the *Width* parameter by 32 and rounding the result to the next highest integer gives the ceiling of the calculation. This ceiling indicates the number of words per row.

The *Height* parameter indicates the number of rows. The marker data must be supplied in row (scan line) major order.

Pattern Defines the image used as a marker.

To define the marker pattern fully, calculate the value of the *Pattern* parameter by multiplying the previously calculated ceiling by the value of the *Height* parameter. The *Pattern* parameter is expressed in words in length. If the *Width* parameter implies partial use of a word, the rest of the word is unused.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

0x, 0y Indicate the coordinates of the origin of the marker relative to the lower leftmost corner (0,0) of the marker pattern. The origin must be placed inside the marker pattern, so that *0x* < *Width* and *0y* < *Height*. The origin of the marker is placed at the position indicated when the application places a marker with the **gsplym** subroutine. If the *0x* parameter equals –1, the origin remains unchanged.

The maximum size of the marker is device dependent. The maximum size equals the height and width of the display, which can be determined by calling the **gsqdsp** subroutine.

Note: The XGSL subroutines do not make a copy of a user-defined polymarker, a marker drawn for a sequence of points. Any changes or reuse of the storage where a user-defined shape is in use can cause unpredictable results.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_COLI Color index not valid

GS_PMSZ Marker size not valid

GS_PMOR Marker origin not valid

GS_PMSY Marker style not valid.

Example

1. To set the polymarker attributes, the **mark.c** C language program uses the **gsmatt** subroutine several times.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsplym** subroutine, **gsqdsp** subroutine.

gsmcat Subroutine

Purpose

Sets the multicolor cursor attributes.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsmcat_ (*Foregrnd*, *Backgrnd*, *Width*, *Height*, *Pattrn*, *Mask*, *0x*, *0y*, *LogOp*)
int **Foregrnd*, **Backgrnd*, **Width*, **Height*;
int **Pattrn*, **Mask*, **0x*, **0y*, **LogOp*;

FORTRAN Syntax

INTEGER function gsmcat_ (*Foregrnd*, *Backgrnd*, *Width*, *Height*, *Pattrn*, *Mask*, *0x*, *0y*, *LogOp*)
INTEGER *Foregrnd*, *Backgrnd*, *Width*, *Height*
INTEGER *Pattrn*, *Mask*, *0x*, *0y*, *LogOp*

Pascal Syntax

FUNCTION gsmcat_ (
VAR *Foregrnd*, *Backgrnd*, *Width*, *Height*: **INTEGER**;
VAR *Pattrn*: **ARRAY [1..***k***] of INTEGER**;
VAR *Mask*: **ARRAY [1..***k***] of INTEGER**;
VAR *0x*, *0y*, *LogOp*: **INTEGER**
): **INTEGER [PUBLIC]**;

Description

The **gsmcat** subroutine defines and enables the multicolor cursor for the AIXwindows Graphics Support Library (XGSL). The **gscmap** subroutine must initialize the color map before the **gsmcat** subroutine can be called.

Only one cursor of the raster-style cursor, either the multicolor cursor or the single-color cursor, can be active in the XGSL at any one time. The **gsmcat** subroutine forces all subsequent calls to the **gsmcur** and **gsecur** subroutines to operate on the multicolor version of the raster cursor. To change from the single-color cursor to the multicolor cursor, erase the cursor with the **gsecur** subroutine and call the **gsmcat** subroutine.

The multicolor cursor is a two-color, clipped cursor with logical operations. Its largest size is 32 bits in width and 32 bits in height. Although the cursor origin cannot be moved outside the frame buffer boundaries, any portion beyond the origin that falls outside the frame buffer is clipped. In addition, a mask is provided that can be used to allow portions of the frame buffer to *show through* the cursor. Any bits set to 0 (zero) in the mask indicate that the matching bits in the cursor pattern do not affect the underlying frame buffer.

Parameters

Foregrnd — Defines a color entry in the color map. This color is used for the foreground color (bits set to 1) in the multicolor cursor raster. A value of –1 indicates no change to this attribute.

Backgrnd — Defines a color entry in the color map. This color is used for the background color (bits set to 0) in the multicolor cursor raster. A value of –1 indicates no change to this attribute.

Width, Height Define, in pixels, the width and height of the bit pattern and mask to be used as the cursor. The maximum value for width and height of the cursor is 32 bits. If the *Width* or *Height* parameter equals –1, then the pattern and the mask remain unchanged.

Pattrn Defines the raster image used as a cursor. The image must be specified in 32-bit integers, and there must be the number of rows specified by the *Height* parameter. The XGSL uses for each integer only the number of bits specified by the *Width* parameter.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

Mask Defines the mask pattern of the cursor. Each bit in the *Mask* parameter corresponds with a bit in the multicolor cursor specified by the *Pattrn* parameter. If a bit is set (has a value of 1), the matching bit in the pattern is applied to the underlying display raster. If a bit is not set (has a value of 0), the matching bit in the pattern is masked and does not affect the underlying display raster. The size of the *Mask* parameter must match the size of the *Pattrn* parameter exactly.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length; that is, the *k* in the routine declaration must be a constant.

0x, 0y Indicate the origin of the cursor relative to the lower leftmost corner (0,0) of the cursor pattern. The origin must be placed within the cursor pattern, so that *0x* < *Width* and *0y* < *Height*. The origin of the cursor is placed at the position indicated when the application moves the cursor using the **gsmcur** subroutine. If *0x* equals –1, then the origin remains unchanged.

LogOp Defines the logical operation to perform between the cursor pattern that is being drawn and the contents of the frame buffer. The following logical operations are supported:

3 REPLACE

6 Exclusive-OR.

The cursor attributes cannot be changed while the cursor is visible.

No default cursor is defined. All cursor parameters must be set before the cursor is displayed.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_COLI Color index not valid

GS_CURS Cursor size not valid

GS_CURO Cursor origin not valid

GS_CURV Cursor visible

GS_LONS Logical operation not valid.

Example

1. To set the multicolor cursor attributes, the **curs.c** C language program uses the **gsmcat** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gscatt** subroutine, **gsecur** subroutine, **gsmcur** subroutine.

gsmcur Subroutine

Purpose

Makes the cursor visible.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsmcur_ (*x*, *y***)**
**int ****x*, ******y*;

FORTRAN Syntax

INTEGER function gsmcur (*x*, *y***)**
INTEGER *x*, *y*

Pascal Syntax

FUNCTION gsmcur_ (
VAR *x*, *y*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsmcur** subroutine makes the cursor visible if it is not already visible. Because AIXwindows controls the cursor placement, the *x* and *y* parameters are not used.

This subroutine operates on either the single-color cursor or the multicolor cursor. The relevant attributes are different, depending on which cursor style is currently defined and enabled.

For the single-color cursor, the relevant attributes are:

- Color map
- Plane mask
- Cursor pattern
- Cursor color index
- Cursor origin.

For the multicolor cursor, the relevant attributes are:

- Color map
- Plane mask
- Multicolor cursor pattern
- Multicolor cursor mask
- Multicolor cursor foreground color
- Multicolor cursor background color
- Multicolor cursor origin
- Multicolor cursor logical operation.

Set the cursor attributes with the **gscatt** or **gsmcat** subroutine before calling the **gsmcur** subroutine.

The cursor is nondestructive. This property is achieved in a device-dependent manner.

Parameters

x, y These parameters are not used under the AIX window manager.

Return Values

Using the preprocessor **include** statement to incorporate **gslerrno.h** in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_CORD Coordinate not valid

GS_UCUR Undefined cursor

GS_INAC Virtual terminal inactive.

Example

1. To display the cursor, the **curs.c** C language program uses the **gsmcur** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gscatt** subroutine, **gsmcat** subroutine, **gsecur** subroutine.

gsmfld Subroutine

Purpose

Provides compatibility for GSL applications that use the **gsmfld** subroutine.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsmfld_ (*NumFonts*, *FontTable***)**
int ****NumFonts*;**
struct *FontInfo FontTable***[ ];**

FORTRAN Syntax

INTEGER gsmfld (*NumFonts*, *FontTable***)**
INTEGER *NumFonts*, *FontTable***(2,*)**

Pascal Syntax

FUNCTION gsmfld_ (
VAR *NumFonts*: **integer;**
VAR *FontTable* : *FontArray* **): integer[PUBLIC];**

Description

The **gsmfld** subroutine is provided for compatibility with existing GSL applications. Under the current implementation of XGSL, the **gsmfld** subroutine always returns **GS_USUC**, indicating that the subroutine was unsuccessful.

Parameters

The parameters are ignored. They are provided for compatibility.

Return Value

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return value:

GS_USUC	Unsuccessful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsmult Subroutine

Purpose

Draws a multiline, a set of lines that connect alternate pairs of points in a sequence.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsmult_ (*Number, x, y***)**
**int ****Number, x*[ ], *y*[ ];

FORTRAN Syntax

INTEGER function gsmult (*Number, x, y***)**
INTEGER *Number, x* (*), *y* (*)

Pascal Syntax

FUNCTION gsmult_ (
VAR *Number*: **INTEGER;**
VAR *x, y*: **ARRAY [1..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsmult** subroutine draws lines, as defined by the current relevant attributes, between alternate pairs of points defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

Number	Defines the number of points in the coordinate arrays. The parameter value must be a multiple of 2, with 2 as the minimum value.
x, y	Define the points for line drawing. For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_NCOR	Number of coordinates not valid
GS_INAC	Virtual terminal inactive.

Example

1. To draw a multiline, the **pix.c** C language program uses the **gsmult** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gspoly** subroutine, **gsdjply** subroutine, **gsline** subroutine, **gslop** subroutine, **gsulns** subroutine.

gspcls Subroutine

Purpose

Defines the end of a shape to fill.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gspcls_ ( )

FORTRAN Syntax

INTEGER function gspcls

Pascal Syntax

FUNCTION gspcls_ : INTEGER [PUBLIC];

Description

The **gspcls** subroutine defines the end of a particular two-dimensional shape to be filled.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_USUC	Unsuccessful.

Example

1. To define the end of a 2-D shape to be filled, the **blit.c** C language program uses the **gspcls** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsbply** subroutine, **gseply** subroutine, **gsfply** subroutine.

gsplym Subroutine

Purpose

Draws a polymarker, a marker at each point of a set of specified points.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsplym_ (*Number, x, y***)**
**int ****Number*, **x*, **y*;

FORTRAN Syntax

INTEGER function gsplym (*Number, x, y***)**
INTEGER *Number*, *x* **(*)**, *y* **(*)**

Pascal Syntax

FUNCTION gsplym_ (
VAR *Number*: **INTEGER;**
VAR *x*, *y*: **ARRAY [1..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsplym** subroutine places a marker, defined by the current relevant attributes, at each point defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Logical operation
- Polymarker color index
- Polymarker style.

Parameters

Number	Defines the number of points in the coordinate arrays. This value must be greater than or equal to 1.
x, y	Define, as coordinate arrays, the location where the origin of each polymarker is placed. For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_NCOR	Number of coordinates not valid.

Example

1. To draw a polymarker, the **mark.c** C language program uses the **gsplym** subroutine several times.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsdjply** subroutine, **gspoly** subroutine, **gsmatt** subroutine.

gspoly Subroutine

Purpose

Draws a polyline, a set of lines that connects a sequence of points.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gspoly_ (*Number*, *x*, *y*)
**int ****Number*, *x*[ ], *y*[ ];

FORTRAN Syntax

INTEGER function gspoly (*Number*, *x*, *y*)
INTEGER *Number*, *x* (*), *y* (*)

Pascal Syntax

FUNCTION gspoly_ (
VAR *Number*: **INTEGER;**
VAR *x*, *y*: **ARRAY [1..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gspoly** subroutine draws lines, as defined by the current relevant attributes, between each pair of points defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

Number	Defines the number of points in the coordinate arrays. The parameter value must be greater than or equal to 2.
x, *y*	Define the points for line drawing. For Pascal syntax, the application must declare that the passed array is fixed–length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_NCOR	Number of coordinates not valid
GS_INAC	Virtual terminal inactive.

Example

1. To draw a polyline, the **arc5.c** C language program uses the **gspoly** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsdjply** subroutine, **gsline** subroutine, **gslop** subroutine, **gsmult** subroutine, **gsunls** subroutine.

gspp Subroutine

Purpose

Provides compatibility for GSL applications that use the **gspp** subroutine.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gspp_ (*Penspeed***)**
int ****Penspeed*;**

FORTRAN Syntax

INTEGER function gspp_ (*Penspeed***)**
INTEGER *Penspeed*

Pascal Syntax

FUNCTION gspp_ (
VAR *Penspeed*: **INTEGER;**
): INTEGER [PUBLIC];

Description

The **gspp** subroutine is provided for compatibility with existing GSL applications. Under the current implementation of XGSL, the **gspp** subroutine has no effect, but always returns a value of **GS_SUCC**, indicating that the subroutine was successful.

Parameter

The parameter is ignored. It is provided for compatibility.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_USUC	Parameter value not valid.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsqdsp Subroutine

Purpose

Returns characteristics of the display monitor and adapter.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

void gsqdsp_ (*Display***)**
int ****Display*;**

FORTRAN Syntax

subroutine gsqdsp_ (*Display***)**
INTEGER ***Display*** **(32)**

Pascal Syntax

PROCEDURE gsqdsp_ (
VAR ***Display*****: ARRAY [1..32] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsqdsp** subroutine returns an array containing the display adapter and monitor characteristics.

Parameter

Display Contains, on return, the relevant display and monitor characteristics. The following table describes the information in the array. Each entry is a word.

Entry	Description
1	Display and monitor ID.
2	Displayed width in pixels of the frame buffer.
3	Displayed height in pixels of the frame buffer.
4	Physical width in millimeters of display.
5	Physical height in millimeters of display.
6	Number of bit planes or number of bits per pixel.
7	Adapter characteristic flags. The following bits set these characteristics (bit 0 is the most significant bit):

Bit	Description
0	Color or monochrome: 0 = Color 1 = Monochrome.
1	By plane or by pixel: 0 = By plane 1 = By pixel.

	2 Software or hardware cursor: 0 = Software 1 = Hardware.
	3 –31 Reserved bits.
8	Number of bits for red digital-to-analog converter.
9	Number of bits for green digital-to-analog converter.
10	Number of bits for blue digital-to-analog converter.
11	Minimum cursor width in pixels.
12	Minimum cursor height in pixels.
13	Maximum cursor width in pixels.
14	Maximum cursor height in pixels.
15	Color table size.
16	Font class: 1 = Compressed 2 = Uncompressed.
17	Logical operation capability. If the value is 0, the adapter supports all two-operand logical operations and all 256 three-operand operations. If the value is nonzero, the most significant bits represent the two–operand logical operations supported. Bit 0 corresponds to logical operation 0, bit 1 to logical operation 1, and so forth.
18 to **32**	Reserved.

Information from this query can be used to scale application coordinates to those of the frame buffer.

Even if the adapter supports no logical operations, the results of the query indicate that the adapter supports REPLACE and Exclusive-OR (logical operations 3 and 6, respectively). The XGSL emulates the latter, if necessary.

Example

1. To obtain information about the display for further calculations, the **cir1.c** C language program uses the **gsqdsp** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gslop** subroutine.

gsqfnt Subroutine

Purpose

Returns information about the active annotated text font.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

void gsqfnt_ (*Font***)**
int ***Font*****[32];**

FORTRAN Syntax

subroutine gsqfnt_ (*Font***)**
INTEGER ***Font*****(32)**

Pascal Syntax

PROCEDURE gsqfnt_ (
VAR ***Font*****: ARRAY [1..32] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsqfnt** subroutine returns information about the active annotated text font.

Parameter

Font An array that contains, on return, the characteristics of the current annotated text font. The following table describes the information in the array. Each entry is a word. Dimensions are in pixels and the origin is at the lower left corner of the character box.

Entry	Description
1	Class: 1 = Compressed 2 = Uncompressed.
2	Annotated font ID
3	Reserved
4	Attribute flags. The following bits are set to 1 to indicate the corresponding attribute:

Bit	Attribute
0	Proportionally spaced
30	Italic
31	Bold.

5	Number of characters
6	Character baseline
7	Character capsline

8	Character width. For a proportionally spaced font, the width value represents the maximum width allowed
9	Character height
10	Underscore top line
11	Underscore bottom line
12 to **32**	Reserved.

Example

1. To obtain information about the active annotated text font, the **curs.c** C language program uses the **gsqfnt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gstatt** subroutine, **gsqgtx** subroutine.

gsqgtx Subroutine

Purpose

Returns information about the current geometric font.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

void gsqgtx_ (*Font, Select*)
int *Font*[32];
int **Select*;

FORTRAN Syntax

subroutine gsqgtx (*Font, Select*)
INTEGER *Font* (32), *Select*

Pascal Syntax

PROCEDURE gsqgtx_ (
VAR *Font*: **ARRAY [1..32] of INTEGER;**
Select: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsqgtx** subroutine returns information about the active geometric font.

Parameters

Font An array that contains, on return, characteristics of the geometric font specified by the *Select* parameter. Geometric font characteristics are described in the programmable character set (PCS) header of a geometric text file. The following list describes the header information returned in the array. Each entry is a word. Dimensions are in pixels and the origin is at the lower left corner of the character box.

Entry	Description
1	Font ID.
2	Segment ID.
3	Character set: 1 = ASCII 2 = EBCDIC.
4	Range of X between 0 and the right edge of the character box.
5	Range of Y between 0 and the top edge of the character box.
6	Starting character code (from 0x21 to 0xFE).
7	Ending character code (from 0x21 to 0xFE).
8	Font baseline (pixels in the Y direction).

	9	Font capsline (pixels in the X direction).
	10	Default error code.
	11 to **32**	Reserved.

Select Specifies the geometric font ID whose header information you want returned in the *Font* parameter.

A value of –1 for the *Select* parameter returns the following information in the *Font* parameter:

Entry	Description
1	The current active font ID.
2	The number of PCS descriptor headers (segments, for 2–byte character sets) loaded at the time of the call to the **gsqgtx** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsgtat** subroutine, **gsgtxt** subroutine.

gsqlext Subroutine

Purpose

Returns expanded information about the locator attributes.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsqlext_ (*Results***)**
int *Results***[16];**

FORTRAN Syntax

INTEGER function gsqlext (*Results***)**
INTEGER *Results***(16)**

Pascal Syntax

FUNCTION gsqlext_ (
VAR *Results*: **ARRAY[1..16] of INTEGER;**
): INTEGER [PUBLIC];

Description

The **gsqlext** subroutine returns an array containing expanded information about the locator device.

Parameter

Results A 16 integer array that, on return, contains information about the locator. The following table describes the information in the array.

Entry	Description
0	Locator resolution in millimeters per 100 counts.
1	Locator device type. If the most significant bit is a 1 (one), the locator type is a tablet. For a tablet, the next two most significant bits (bits 1 and 2) indicate the sensor type. The following bit patterns for bits 1 and 2 indicate the corresponding sensor type:

Bit Pattern	Sensor Type
00	Sensor type is undefined or no sensor is attached.
01	A stylus is attached.
10	A four-button puck is attached.

2	Maximum horizontal count.
3	Maximum vertical count.
4	The horizontal locator dead zone in units of 0.25 millimeter.
5	The vertical locator dead zone in units of 0.25 millimeter.
6 to 15	Reserved.

If an attempt to obtain the locator attributes is unsuccessful, the device may not be attached.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_USUC Unsuccessful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsqloc** subroutine, **gslcat** subroutine.

gsqloc Subroutine

Purpose

Returns information about the locator.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

void gsqloc_ (*LocatorType*, *xRes*, *yRes*, *Hg*, *Vg*)
int **LocatorType*, **xRes*, **yRes*, **Hg*, **Vg*;

FORTRAN Syntax

subroutine gsqloc (*LocatorType*, *xRes*, *yRes*, *Hg*, *Vg*)
INTEGER *LocatorType*, *xRes*, *yRes*, *Hg*, *Vg*

Pascal Syntax

PROCEDURE gsqloc_ (
VAR *LocatorType*, *xRes*, *yRes*, *Hg*, *Vg*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsqloc** subroutine returns the resolution of the locator device. The current setting for an absolute device dead zone is also returned.

Parameters

LocatorType Indicates the type of locator. If the most significant bit is a 0 (zero), the locator is a mouse. If the most significant bit is a 1 (one), the locator is a tablet. For a tablet, the next most significant 2 bits are:

00	Sensor type is undefined or no sensor is attached.
01	A stylus is attached.
10	A four-button puck is attached.

xRes, yRes Indicate the horizontal and vertical resolution of the pointer device in millimeters per 100 counts.

Hg, Vg Indicate the horizontal and vertical values for the locator dead zone in units of 0.25 millimeters.

If an attempt to get the locator attributes is unsuccessful, the device may not be attached.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_UNSC	Unsuccessful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsqlext** subroutine, **gslcat** subroutine.

gsrrst Subroutine

Purpose

Restores a rectangular block of pixels.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsrrst_ (*Buffer*, *x1*, *y1*, *x2*, *y2*)
int **Buffer*, **x1*, **y1*, **x2*, **y2*;

FORTRAN Syntax

INTEGER function gsrrst (*Buffer*, *x1*, *y1*, *x2*, *y2*)
INTEGER *Buffer* (*), *x1*, *y1*, *x2*, *y2*

Pascal Syntax

FUNCTION gsrrst_ (
VAR *Buffer*: **ARRAY [1..***k***] of INTEGER;**
VAR *x1*,*y1*, *x2*,*y2*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsrrst** subroutine restores a block of pixels from memory to the frame buffer. Use the **gsrsav** subroutine to save a block of pixels.

The relevant attributes are:

- Plane mask
- Logical operation.

Parameters

Buffer — Indicates where the block of pixels is stored in memory.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

x1, y1 — Define the coordinates of the lower left corner of the rectangular area to restore.

x2, y2 — Define the coordinates of the upper right corner of the rectangular area to restore.

Note: The intended purpose of the **gsrsav** and **gsrrst** subroutines is efficient saving and restoring of pixel blocks displayed temporarily at a fixed location in the frame buffer. Because the AIXwindows Graphics Support Library (XGSL) saves the frame buffer contents in a device-dependent fashion, it is generally not possible to move blocks of pixels correctly from one position to another in a plane-oriented adapter by using the **gsrsav** and **gsrrst** subroutines. Use the *display* parameter entries in the **gsqdsp** subroutine to determine whether the display adapter is plane-oriented or pixel-oriented. In addition, careful consideration must be given to the adapter characteristics, block size, and position of the block in the frame buffer before attempting to manipulate the *Buffer* parameter.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_CORD	Coordinate not valid
GS_INAC	Virtual terminal inactive.

Example

1. To restore a block of pixels previously stored in the frame buffer, the **pix.c** C language program uses the **gsrrst** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsrsav** subroutine, **gsxblt** subroutine, **gsqdsp** subroutine.

gsrsav Subroutine

Purpose

Saves a rectangular block of pixels.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsrsav_ (*Buffer, x1, y1, x2, y2*)
int **Buffer*, **x1*, **y1*, **x2*, **y2*;

FORTRAN Syntax

INTEGER function gsrsav (*Buffer, x1, y1, x2, y2*)
INTEGER *Buffer*(*), *x1, y1, x2, y2*

Pascal Syntax

FUNCTION gsrsav_ (
VAR *Buffer*: **ARRAY [1..***k***] of INTEGER;**
VAR *x1,y1, x2,y2*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsrsav** subroutine saves a block of pixels from the frame buffer, to memory storage starting at the indicated address. The *x1, y1* and *x2, y2* parameters define the block of pixels. The saved block can be restored with the **gsrrst** subroutine. The relevant attributes are:

- Plane mask
- Logical operation.

Parameters

Buffer Indicates where the **gsrsav** subroutine should save the block of pixels.

Note: The size of the *Buffer* parameter depends on the size of the rectangle and on the device organization. For devices organized by plane, the plane mask attribute determines the number of planes saved for each pixel. For devices organized by pixel, the entire pixel is always saved. For both organizations, the unit of access to the frame buffer also plays a role in calculating the size of the *Buffer* parameter. The **gsxblt** subroutine provides more details. The **gsrsav** subroutine does not check whether the *Buffer* parameter is too small to contain the pixel block. A memory fault can result if the *Buffer* parameter is too small. To specify a size for the *Buffer* parameter that will hold all images, use the following equation:

```
(((y2-y1+1)/32+2)* (x2-x1+1))
```

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

x1, y1 Define the lower left corner of the rectangular area in the frame buffer to save.

x2, y2 Define the upper right corner of the rectangular area in the fram buffer to save.

Note: The intended purpose of the **gsrsav** and **gsrrst** subroutines is efficient saving and restoring of pixel blocks displayed temporarily at a fixed location in the frame buffer. Because the AIXwindows Graphics Support Library (XGSL) saves the frame buffer contents in a device-dependent fashion, it is generally not possible to move blocks of pixels correctly from one position to another in a plane-oriented adapter by using the **gsrsav** and **gsrrst** subroutines. Use the *Display* parameter entries in the **gsqdsp** subroutine to determine whether the display adapter is plane-oriented or pixel-oriented. In addition, careful consideration must be given to the adapter characteristics, block size, and position of the block in the frame buffer before attempting to manipulate the *Buffer* parameter.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_CORD Coordinate not valid

GS_INAC Virtual terminal inactive.

Example

1. To save a block of pixels in storage, the **pix.c** C language program uses the **gsrsav** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsrrst** subroutine, **gsxblt** subroutine.

gssend Subroutine

Purpose

Provides compatibility for GSL applications that use the **gssend** subroutine.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gssend_ (*Count, SizeBytes, Buffer*)
int **Count*;
int **SizeBytes*;
int **Buffer*;

FORTRAN Syntax

INTEGER gssend (*Count, SizeBytes, Buffer*)

Pascal Syntax

FUNCTION gssend_ (
VAR *Count* **: INTEGER;**
VAR *SizeBytes* **: ARRAY [1..***k***] of INTEGER;**
VAR *Buffer* **: ARRAY [1..***k***] of INTEGER;**
) : INTEGER [PUBLIC];

Description

The **gssend** subroutine is provided for compatibility with existing GSL applications. Under the current implementation of XGSL, the **gssend** subroutine always returns **GS_USUC**, indicating that the subroutine was unsuccessful.

Parameters

The parameters are ignored. They are provided for compatibility.

Return Value

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return value:

GS_USUC Unsuccessful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gstatt Subroutine

Purpose

Sets the text attributes for annotated text.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gstatt_ (*color*, *page*, *baseline*, *font*, *name*)
int **color*, **page*, **baseline*, **font*;
char **name*;

FORTRAN Syntax

INTEGER function gstatt_ (*Color*, *Page*, *Baseline*, *Font*, *Name*)
INTEGER *Color*, *Page*, *Baseline*, *Font*
CHARACTER*n name*

Pascal Syntax

FUNCTION gstatt_ (
VAR *Color*, *Page*, *Baseline*, *Font*: **INTEGER;**
VAR *name*: **ARRAY [0..***k***] of CHAR**
): INTEGER [PUBLIC];

Description

The **gstatt** subroutine defines the attributes for the class of text drawing functions.

Parameters

Color Specifies a text color entry in the color map. If this is –1, the attribute is unchanged.

Page Specifies the code page of a font for the display to use. The only valid value for IBM-supplied fonts is 0 for code page P0. The value –1 indicates no change.

Baseline Determines the direction of the text drawing. The valid values are:

–1 Attribute remains unchanged.

0 Specifies 0 degrees, or left to right in the viewer's terms.

1 Specifies 90 degrees, or *up* in the viewer's terms.

2 Specifies 180 degrees, or right to left in the viewer's terms.

Note: The characters appear upside down.

3 Specifies 270 degrees, or *down* in the viewer's terms.

If the baseline is other than 0 degrees and the font index is 0, then the font named by the *Name* parameter must be a pre-rotated font. When a baseline change is made, another font path name is required.

Font Specifies, for displays, the font to use for text output operations.

If the font index is –1, no change is made. If the font index is 0, then the **gstatt** subroutine uses the font specified by the *Name* parameter. If the font index is a value from 1 to 14, the XGSL uses one of the following predefined fonts:

Font Index	Width x Height (in pixels)	Style
1	9 x 20	Normal
2	9 x 20	Italic
3	9 x 20	Bold
4	8 x 14	Normal
5	4 x 8	Normal
6	18 x 40	Normal
7	12 x 30	Normal
8	9 x 20	Normal
9	6 x 9	Normal
10	6 x 11	Normal
11	7 x 15	Normal
12	7 x 22	Normal
13	11 x 23	Normal
14	11 x 23	Bold

Name Contains the null-terminated full path name of the font file used when the font attribute is specified as user.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_COLI Color index not valid

GS_CPID Code page identifier not valid

GS_BASL Baseline direction not valid

GS_FNTI Font index not valid

GS_FNTN File name not valid.

Example

1. To set the text attributes for annotated text, the **curs.c** C language example program uses the **gstatt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsterm Subroutine

Purpose

Terminates use of the XGSL.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

void gsterm_ ( )

FORTRAN Syntax

subroutine gsterm ( )

Pascal Syntax

PROCEDURE gsterm_ ( ) [PUBLIC];

Description

The **gsterm** subroutine terminates the XGSL. It deallocates any private storage, returns the virtual terminal to KSR mode, and causes the monitor mode signals to be ignored.

Example

1. To deallocate memory reserved by the XGSL and to return the virtual terminal to the normal KSR mode, the **arc1.c** C language program uses the **gsterm** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsinit** subroutine.

gstext Subroutine

Purpose

Writes annotated text.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gstext_ (*x*, *y*, *Number*, *Text***)**
int ***x*, ***y*, ****Number*;
char ****Text*;

FORTRAN Syntax

INTEGER function gstext (*x*, *y*, *Number*, *Text***)**
INTEGER *x*, *y*, *Number*
CHARACTER*n *Text*

Pascal Syntax

FUNCTION gstext_ (
VAR *x*, *y*, *Number*: **INTEGER;**
VAR *Text*: **ARRAY [1..***k***] of CHAR**
): INTEGER [PUBLIC];

Description

The **gstext** subroutine writes the number of characters indicated by the parameters, starting at the specified baseline position and according to the relevant attributes. This subroutine is to be used only with annotated text.

The relevant attributes are:

- Color map
- Plane mask
- Text font
- Code page
- Baseline direction
- Text color index.

Parameters

x, y	Define the baseline position for writing the text.
Number	Indicates the number of bytes to write from the *Text* string.

Text Contains as an array the ASCII codes for the characters to write.

For any ASCII value between 0 and 31 (decimal), no graphic is written. For any other ASCII value combination that does not result in a valid graphic, a dash is written.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length; that is, the *k* in the routine declaration must be a constant.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_CORD Coordinate not valid

GS_FBUF Frame buffer overflow

GS_INAC Virtual terminal inactive

GS_SUCC Successful.

Example

1. To write annotated text, the **curs.c** C language program uses the **gstext** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gstatt** subroutine.

gsulns Subroutine

Purpose

Sets the user line style.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsulns_ (*Pattern*, *Length*, *Begin*)
int **Pattern*, **Length*, **Begin*;

FORTRAN Syntax

INTEGER function gsulns (*Pattern*, *Length*, *Begin*)
INTEGER *Pattern*, *Length*, *Begin*

Pascal Syntax

FUNCTION gsulns_ (
VAR *Pattern*, *Length*, *Begin*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsulns** subroutine establishes the user line style.

Parameters

Pattern Defines the pixel pattern used for the line style. A 1 bit indicates that the XGSL draws a pixel; a 0 bit means that it does not.

Length Defines the number of bits (starting with the most significant) of the *Pattern* parameter used for line drawing. The bits are repeated for the Length of the line.

The *Length* parameter is a value not less than 2 or greater than 32.

Begin Indicates the length of the starting run of bits set to 1 in the pattern. It is used to adjust the beginning and ending runs of the noncontinuous line styles.

For proper appearance, the application-supplied line pattern begins with a run of bits set to 1 and ends with a run of bits set to 0.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_LENG Length not valid.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsunlk Subroutine

Purpose

Provides compatibility for GSL applications that use the **gsunlk** subroutine.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

void gsunlk_ ( )

FORTRAN Syntax

subroutine gsunlk ( )

Pascal Syntax

PROCEDURE gsunlk_ ( ) [PUBLIC];

Description

The **gsunlk** subroutine is provided for compatibility with existing GSL applications. Under the current implementation of XGSL, the **gsunlk** subroutine always returns **GS_USUC**, indicating that the subroutine was unsuccessful.

Return Value

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return value:

GS_USUC Unsuccessful.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsvgrn Subroutine

Purpose

Sets the valuator granularity.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsvgrn_ (*Valuators*, *Granularity*)
**char ****Valuators*, ******Granularity*;

FORTRAN Syntax

INTEGER function gsvgrn (*Valuators*, *Granularity*)
CHARACTER *Valuators*, *Granularity*

Pascal Syntax

FUNCTION gsvgrn_ (
VAR *Valuators*, *Granularity*: **CHAR**
): INTEGER [PUBLIC];

Description

The **gsvgrn** subroutine sets the resolution of input events generated by the valuators, that is, the number of events per turn of the valuator dial.

Parameters

Valuators Specifies which valuators to set to the indicated granularity. Each bit in the *Valuators* parameter corresponds to one of the valuator dials, with the most significant bit indicating that valuator 0 (zero) is to be set, the next most significant bit indicating that valuator 1 (one) is to be set, and so on.

Granularity Specifies the desired resolution for the valuators indicated. The *Granularity* parameter must have a value of 2 through 8 and indicates a resolution of 4, 8, 16, 32, 64, 128, or 256 points per revolution, respectively. The default value is 4, for a resolution of 16.

If an attempt to set the valuator granularity is unsuccessful, the device may not be attached.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC Successful

GS_USUC Unsuccessful

GS_VALG Granularity not valid.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

gsxblt Subroutine

Purpose

Moves a rectangular block of pixels in system or display adapter memory from one location to another.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

int gsxblt_ (*SrcPix*, *DstPix*, *MskPix*, *Width*, *Height*, *Logop*)
int **SrcPix*, **DstPix*, **MskPix*;
int **Width*, **Height*, **Logop*;

FORTRAN Syntax

INTEGER function gsxblt_ (*SrcPix*, *DstPix MskPix*, *Width*, *Height*, *Logop*)
INTEGER *SrcPix* (*), *DstPix*(*), *MskPix*(*)
INTEGER *Width*, *Height*, *Logop*

Pascal Syntax

FUNCTION gsxblt_ (
VAR *SrcPix*, *DstPix*, *MskPix*: **ARRAY [32] of INTEGER;**
VAR *Width*, *Height*, *Logop* : **INTEGER**
): **INTEGER [PUBLIC];**

Description

The **gsxblt** subroutine moves a rectangular block of pixels from one memory location to another, either in system memory or in the frame buffer.

For FORTRAN-specific address information, see the **gsxptr** subroutine.

The **gsxblt** subroutine is used to support windowing operations, such as overlays and movement around the screen. The source rectangle and the destination rectangle can be in either system or adapter pixel memory. The **gsxblt** subroutine is also used for user-defined cursors and for the save and restore of a pixel map for applications such as pop-up menus.

The mask operation provided by the **gsxblt** subroutine controls which pixels in the destination rectangle can be modified.

The relevant attributes are:

- Plane mask
- Color map.

Parameters

SrcPix	Contains the address of the source pixel map.
DstPix	Contains the address of the destination pixel map.

MskPix Contains the address of the mask operation pixel map. This parameter should equal 0 if there is no bit mask operator to apply. For FORTRAN applications, a valid *MskPix* array must always be defined. If no masking is required, the address field of the array, *MskPix[9]*, must be initialized to 0.

The *MskPix* pixel map consists of only 1 bit per pixel, and the mask rectangle is the same size as the source and destination rectangles. In the mask rectangle, a 1 bit means that the corresponding pixel in the destination rectangle can be modified, while a 0 bit means the destination pixel is not to be modified.

Width Defines the width of the rectangular area to be transferred.

Height Defines the height of the rectangular area to be transferred.

Logop Indicates the logical operation to perform between the source pixel map and the destination pixel map.

In the following table, please note:

- The source or tile (a special type of source) pixels represent bits of data to be merged in some way with the corresponding bits of data in the destination rectangle.
- The first three columns of the table specify the operations you can perform, and the Code column contains the corresponding value you should specify for the *Logop* parameter.
- There are two unique codes for each logical operation, to be used depending on how the tiling bit in the source pixel map is set. Codes 0 to 15 are used when the tiling bit is not set, while codes 16 to 31 are used when the tiling bit is set.
- A ~ (tilde) represents the logical INVERSE.

Logical Operation Table **Part 1 of 2**

Type of Source	Logical Operation	Destination Type	Code
		Destination clear	0
		Set destination	15
	No operation	Destination	5
		~Destination	10
Source	REPLACE	Destination	3
Source	AND	Destination	1
Source	AND	~Destination	2
Source	Exclusive-OR	Destination	6
Source	OR	Destination	7
Source	OR	~Destination	11
~Source	REPLACE	Destination	12
~Source	AND	Destination	4
~Source	AND	~Destination	8
~Source	Exclusive-OR	Destination	9

Logical Operation Table			Part 2 of 2
Type of Source	**Logical Operation**	**Destination Type**	**Code**
~Source	OR	Destination	13
~Source	OR	~Destination	14
		Destination clear	16
	No operation	Destination	21
		~Destination	26
Tile	REPLACE	Destination	19
Tile	AND	Destination	17
Tile	AND	~Destination	18
Tile	Exclusive-OR	Destination	22
Tile	OR	Destination	23
Tile	OR	~Destination	27
~Tile	REPLACE	Destination	28
~Tile	AND	Destination	20
~Tile	AND	~Destination	24
~Tile	Exclusive-OR	Destination	25
~Tile	OR	Destination	29
~Tile	OR	~Destination	30

A pixel map is a 32-bit array of integers that contains the following bit fields:

0 Device ID (0 for memory).

1 Flags.

In the following explanations, bit 0 is the low-order bit:

- Plane (XY) format is selected when bit 0 is set and bits 1 and 2 are not set. Pixel (Z) format is selected when bits 0, 1, and 2 are not set.
- A repetitive tile is specified when bit 3 is set, while no tile is specified when bit 3 is not set.

  If the repetitive tile bit is set in the *SrcPix*, pixel map, then the Device ID field in that pixel map must be set to 0. The tile data must be in memory.
- Bit 4 selects the lower left coordinate system when it is set and the upper left coordinate system when it is not set.

2 Height (in pixels).

3 Width (in pixels).

This value must be an even multiple of 16 pixels for all pixel maps, which means that all pixel maps must be at least 16 pixels wide.

4 Number of bits per pixel.

5 Pixels per byte, right-justified.

6 Bytes per pixel.

7 x offset.

8	y offset.
9	Address of upper left corner of data.
10	Foreground color index.
11	Background color index.
12 to 31	Reserved.

Definitions of pixel map terms include:

Device ID This is a required parameter for all pixel map definitions. If the pixel map being defined is a display adapter, this field must contain the Device ID of that display adapter. If the pixel map resides in system memory, then this field must equal 0.

Pixel format Data stored in this format has all bits for a pixel stored together. The data starts with the origin and increases first in the x direction, then in the y direction.

As an example using the upper left coordinate system, a pixel map with 4 bits per pixel and 1 pixel per byte stores the 4 bits for the pixel at location (0,0) in the first byte of the data area, right-justified in the byte. The 4 bits for the pixel at location (1,0) are stored in the second byte, followed by the rest of the pixel values in that row. When the end of the row is reached, the next byte contains the 4 bits for the pixel at location (0,1), followed by the rest of the pixel values in that row, and so on for the entire image.

Plane format Plane format indicates that each of the bits that make up a pixel is stored in a separate, consecutive plane in memory. The most significant bit is first, followed by the next significant, and so on to the least significant bit, which is last. The bits within a plane are packed together 8 bits per byte. Therefore, using the upper-left coordinate system as an example, a pixel map with 4 bits per pixel would consist of four separate planes of data with the first bit value being the one for location (0,0) and increasing first in the x direction, then in the y direction.

Repetitive tiling operation
This operation consists of repeatedly copying a 16-pixel-wide by 16-pixel-high tile rectangle pointed to by the tile pixel map data address to fill a rectangular area of a size specified by the *Height* and *Weight* parameters of this call. The format of the tile data is determined by the format defined in the `flags` field of the tile pixel map structure.

Upper left coordinate system
This indicates that the upper left corner of the pixel map is used as the origin of the coordinate system, with increasing values of x moving to the right and increasing values of y moving down. The x offset and y offset are set to the upper left corner of the rectangle when using this coordinate system.

Lower left coordinate system
This indicates that the lower left corner of the pixel map is used as the origin of the coordinate system, with increasing values of x moving to the right and increasing values of y moving up. The x offset and the y offset are set to the lower-left corner of the rectangle when using this coordinate system. However, the data address specified in the pixel map structure must always point to the upper-left corner of the data area no matter which coordinate system is defined.

Number of bits per pixel
This field identifies the number of data bits required to define a pixel value. For example, a simple monochrome display requires only 1 bit per pixel, while a color display may require 4 bits of information to define a pixel.

Number of pixels per byte
If the number of bits per pixel is fewer than 8, this field defines how many pixels are stored in each byte of pixel map data. A pixel map with only 1 bit per pixel must always store 8 pixels per byte. It is strongly recommended that for 2 through 7 bits per pixel, you store data with only 1 pixel per byte.

Bytes per pixel
If the number of bits per pixel is greater than 8, this field defines how many bytes are used to store each pixel. It is strongly recommended that for 9 through 16 bits per pixel, you store data 2 bytes per pixel. For 17 through 32 bits per pixel, data should be stored 4 bytes per pixel.

Foreground color index
This specifies the color index value to use for a value of 1 in the source pixel map during a color expansion operation.

Background color index
This specifies the color index value to use for a value of 0 in the source pixel map during a color expansion operation.

A color expansion operation takes place automatically when the source pixel map data area contains only 1 bit per pixel and the destination pixel map data area is a color display adapter frame buffer defined to have more than 1 bit per pixel. In this case, when a 1 is specified in the source pixel map data area, the foreground color index value specified in the destination pixel map (*DstPix*) is written to the destination data area. When a 0 is specified in the source pixel map data area, the background color index value specified in the *DstPix* parameter is written to the destination data area.

The foreground color index and the background color index must be initialized in the destination pixel map before calling this operation, but do not need to be initialized in the source or mask pixel maps.

Not all logical operations are supported for a color expansion operation. The following table shows which operations are supported. In this table, a ~ (tilde) represents the logical INVERSE.

Source Pixel Maps		Tile Pixel Maps	
Type of Operation	**Code**	**Type of Operation**	**Code**
Destination clear	0	Destination clear	16
Set destination	15	Set destination	31
Destination	5	Destination	21
~Destination	10	~Destination	26
Source	3	Source	19
~Source	12	~Source	28

If a source or destination pixel map structure defines the active display adapter, you do not need to initialize all the fields of that pixel map structure. Device-dependent information, such as height, width, pixels per byte, bytes per pixel, and address of data, is supplied automatically. You must initialize the fields for device ID, bits per pixel, flags (except for the

data format bits), x offset, and y offset. Also, the foreground color index and the background color index must be initialized if appropriate for this adapter.

When initializing a pixel map structure to use as the *MskPix* parameter:

1. The flags field equals a value of 0x01 if the upper left coordinate system is used or 0x11 if the lower left coordinate system is used.
2. The number of bits per pixel equals 1.
3. The number of pixels per byte equals 8.

The XGSL plane mask attribute applies to all **gsxblt** subroutine operations that use the display adapter as the source or destination pixel map.

The XGSL color map attribute applies to all **gsxblt** subroutine operations that use the display adapter as the destination pixel map.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful.
GS_IWID	Width specification not valid. The x offset plus the *Width* parameter of one of the pixel maps exceeds the total width of that pixel map.
GS_IHEI	Height specification not valid. The y offset plus the *Height* parameter of one of the pixel maps exceeds the total height of that pixel map.
GS_NPLF	Source and destination data formats do not match.
GS_INAC	Virtual terminal inactive.
GS_CORD	Coordinate not valid. Coordinate specified placed the origin of the source, destination, or mask rectangle outside its pixel map.
GS_IBP	Value not valid for bits per pixel in the source pixel map.
GS_CEX	Color expansion operation attempted, but the destination pixel map was not a display adapter.
GS_PWID	The width of one of the pixel maps is not an even multiple of 16 pixels.

Example

1. To move a block of pixels from a memory storage area, the **blit.c** C language program uses the **gsxblt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsxptr** subroutine.

gsxcnv Subroutine

Purpose

Converts pixel map data organization.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsxcnv_ (*Inppix*, *Outpix*)
int **Inppix*, **Outpix*;

FORTRAN Syntax

INTEGER function gsxcnv (*Inppix*, *Outpix*)
INTEGER *Inppix* (*), *Outpix* (*)

Pascal Syntax

FUNCTION gsxcnv_ (
VAR *Inppix*, *Outpix*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsxcnv** subroutine converts pixel map data to and from planes. That is, the **gsxcnv** subroutine converts XY form to and from pixels (Z form).

Both the *Inppix* and *Outpix* parameters contain the address of a pixel map. The fields of each pixel map must be completely initialized before calling this subroutine. Both pixel maps must point to data areas that reside in system memory, not in a display adapter frame buffer.

The *Inppix* and *Outpix* pixel maps do not have to specify the same number of bits per pixel. If there are more input bits per pixel, the least significant bits are truncated. If there are fewer input bits per pixel than required to fill out the destination, the most significant bits are filled with zeros.

The **gsxcnv** subroutine only supports pixel maps defined to have 8 bits per pixel or less. If a pixel-format pixel map is defined with less than 8 bits per pixel, the data must be arranged 1 (one) byte per pixel, right-justified in that byte.

The widths of the two data areas must be identical. The heights of the two data areas must also be identical.

Parameters

Inppix Points to the address of the pixel map whose data area is to be converted.

Outpix Points to the address of the pixel map that contains the address of where to put the converted data.

Note: The calling process must allocate enough storage in the area pointed to by the *Outpix* pixel map to contain all of the converted data.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_INPF	Data format specified in *Inppix* pixel map structure not valid
GS_OUTF	Data format specified in *Outpix* pixel map structure not valid
GS_BMAX	Pixel map defines data of more than 8 bits per pixel.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **gsxblt** subroutine.

gsxptr Subroutine

Purpose

Creates FORTRAN pointer-type variables for addressing data.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**).

C Syntax

None

FORTRAN Syntax

INTEGER function gsxptr (*Intptr*, *Datptr*)
INTEGER *Intptr*(*), *Datptr*(*)

Pascal Syntax

None

Description

The **gsxptr** subroutine is used in FORTRAN applications to place the address of a variable into another variable so that the data can be addressed with a pointer-type addressing.

You must call the **gsxptr** subroutine before the **gsxblt** subroutine so that the data address field of a pixel map structure can be initialized.

Parameters

Intptr	Contains the data that must be addressed by a pointer.
Datptr	Contains, on return, the address of the data area. This variable can then be used as a pointer.

Return Value

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return value:

GS_SUCC	Successful.

Example

1. To specify the path of a user-defined font file as a pointer in the **gsmfld** subroutine, the **fontld.for** example FORTRAN program uses the **gsxptr** subroutine.

Related Information

The **gsxblt** subroutine.

gsxtat Subroutine

Purpose

Sets the text attributes for **Xtext**.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsxtat_ (*Foregrnd*, *Backgrnd*, *Logop*, *Font*, *Clipbox*)
int **Foregrnd*, **Backgrnd*,
int **Logop*, **Font*, **Clipbox*;

FORTRAN Syntax

INTEGER function gsxtat (*Foregrnd*, *Backgrnd*, *Logop*, *Font*, *Clipbox*)
INTEGER *Foregrnd*, *Backgrnd*, *Logop*, *Font*, *Clipbox*

Pascal Syntax

FUNCTION gsxtat_ (
VAR *Foregrnd*, *Backgrnd*, *Logop*, **: INTEGER;**
VAR *Font*: **ARRAY [1..***k***] of INTEGER;**
VAR *Clipbox*: **ARRAY [1..***l***] of INTEGER;**
): INTEGER [PUBLIC];

Description

The **gsxtat** subroutine defines the attributes to be used when drawing text with a font in the **Xtext** format.

Parameters

Foregrnd — Defines an entry in the color map to use for the foreground color (bits set to 1) in the font raster for each character. A value of –1 indicates no change for this attribute.

Backgrnd — Defines an entry in the color map to use for the background color (bits set to 0) in the font raster for each character. A value of –1 indicates no change for this attribute.

Logop — Indicates the logical operation to perform between the font raster and the display destination.

In the following table, please note:

- The source pixels represent bits of data from the font raster to be merged in some way with the corresponding bits of data in the destination rectangle.
- The first three columns of the table specify the operations you can perform, and the Code column contains the corresponding value you should specify for the *logop* parameter.
- A ~ (tilde) represents the logical INVERSE.

Logical Operation Table			
Type of Source	**Logical Operation**	**Destination Type**	**Code**
		Destination clear	0
		Set destination	15
	No operation	Destination	5
		~Destination	10
Source	REPLACE	Destination	3
Source	AND	Destination	1
Source	AND	~Destination	2
Source	Exclusive-OR	Destination	6
Source	OR	Destination	7
Source	OR	~Destination	11
~Source	REPLACE	Destination	12
~Source	AND	Destination	4
~Source	AND	~Destination	8
~Source	Exclusive-OR	Destination	9
~Source	OR	Destination	13
~Source	OR	~Destination	14

A value of –1 for this parameter indicates no change in the current logical operation.

Font

Points to an **Xtext** font file that contains the font header and raster definitions for all characters defined in the font. The calling process is responsible for either mapping the font file or copying it into a memory area in order to obtain a pointer to the data area.

Setting the value of this pointer to 0 indicates no change to the current font file.

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

The XGSL supports only a subset of the different forms that the **Xtext** format allows. Specifically, the XGSL supports any combination of the following font formats:

- Fixed width and height
- Variable width or height, or both
- Halfword alignment or fullword alignment
- Glyphs in raster format only
- Index character array width of 4 bytes
- All individual glyph character bounds for variable width and height fonts, except negative left or right bearings.

The XGSL does not support any formats for **Xtext** files other than those listed above. If the font file specified is not in a supported format, then the XGSL returns the GS_FFMT return code.

Clipbox Specifies an array of integers that correspond to a rectangular area on the display screen. When the **gsxtxt** subroutine is used to draw text, any full or partial characters that fall outside this area are clipped. The elements of the area to clip are as follows:

For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *l* in the routine declaration must be a constant.

First element	Reserved. This value is always 1.
Second element	Specifies, in pixels, the x coordinate of the lower left corner of the clip box.
Third element	Specifies, in pixels, the y coordinate of the lower left corner of the clip box.
Fourth element	Specifies, in pixels, the height of the clip box.
Fifth element	Specifies, in pixels, the width of the clip box.

The bottom and left edges of the clip box are inclusive, while the top and right edges are exclusive.

This parameter is a pointer to the clip box array, which is not copied into any XGSL data structure, allowing the calling process to modify the elements of the array without calling the **gsxtat** subroutine. If the values for the clip box are changed between calls to the **gsxtxt** subroutine, the new clip box is used for all text drawing until another change is made.

Setting the value of this pointer to 0 indicates no change.

Note:

1. Since the XGSL subroutines that use the **Xtext** format are designed for high-performance text drawing, no verification is made of the validity of the clip box. The calling process must ensure that the entire clip box resides inside the physical size of the display. Using a clip box that is not entirely within the screen produces unpredictable results.
2. When the XGSL is installed from diskette, an attempt is made to convert the 14 precompiled annotated text fonts into **Xtext** format. The **vrm2rtfont** command is used on the 14 annotated text fonts in the **/etc/vtm** directory, and the resulting **Xtext** fonts are stored in the **/usr/lpp/fonts** directory. The following list shows the **Xtext** format files stored in **/usr/lpp/fonts**:

   All of these fonts have fixed width and height and are halfword aligned.

Return Values

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return values:

GS_SUCC	Successful
GS_FFMT	Font format not valid
GS_LONS	Logical operation not valid.

Example

1. To set the text attributes for **Xtext**, the **xtex.c** C language program uses the **gsxtat** subroutine.

Files

/usr/include/rtfont.h	Header file for the **Xtext** format.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Related Information

The **vrm2rtfont** command.

The **gsxtxt** subroutine.

gsxtxt Subroutine

Purpose

Writes **Xtext**.

Library

The AIXwindows Graphics Support Library (**libxgsl.a**)

C Syntax

int gsxtxt_ (*x*, *y*, *Number*, *Text*)
int **x*, **y*, **Number*;
char *Text*[ ];

FORTRAN Syntax

INTEGER function gsxtxt (*x*, *y*, *Number*, *Text*)
INTEGER *x*, *y*, *Number*
CHARACTER*n *Text*

Pascal Syntax

FUNCTION gsxtxt_ (
VAR *x*, *y*, *Number*: **INTEGER;**
VAR *Text*: **ARRAY [1..***k***] of CHAR**
): INTEGER [PUBLIC];

Description

The **gsxtxt** subroutine displays the specified text string by using the **Xtext** format. Only those full or partial characters within the clip box specified by the **gsxtat** subroutine are displayed. Because there is no default **Xtext** defined for use by the AIXwindows Graphics Support Library (XGSL), the **gsxtat** subroutine must be called to set all relevant attributes before the first call to the **gsxtxt** subroutine.

The relevant attributes are:

- Xtext foreground color
- Xtext background color
- Xtext logical operation
- Xtext clip box
- Current Xtext font
- Plane mask
- Color map.

Parameters

x, *y*	Define the baseline position for writing the text.
Number	Indicates the number of bytes to write from the *Text* string.
Text	Contains the ASCII codes for the characters to write.
	For FORTRAN syntax, the *n* in the routine declaration must be a constant.
	For Pascal syntax, the application must declare that the passed array is fixed-length and that the routine accepts an array of that length. That is, the *k* in the routine declaration must be a constant.

Return Value

Using the preprocessor **include** statement to incorporate the **gslerrno.h** header file in your program, you can compare the integer returned by this subroutine to the following return value:

GS_SUCC Successful.

Note: Because the text is either displayed or clipped in any case, the **gsxtxt** subroutine always completes execution with a successful return value.

Example

1. To write text using the **Xtext** font format, the **xtex.c** C language program uses the **gsxtxt** subroutine.

Implementation Specifics

This subroutine is part of XGSL in the AIXwindows environment.

Files

/usr/include/rtfont.h	Header file for the **Xtext** format.

Chapter 4. XGSL Example Programs

arc1.c Example C Language Program

```
/*
arc1.c

This program uses the gsline subroutine to draw two intersecting
lines.  It then uses the gseara subroutine to draw several ellipses
around the point where the lines intersect.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   int ring[128], save, outdev;
   int ring_size, rc, lcolor, lstyle, lop;
   int cx, cy, start_ang, end_ang;
   int major, minor, ang;
   int x1, y1, x2, y2;
   int one=1, three=3;
   int data[33];

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */


 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                    frame buffer */
 outdev = -1;                   /* Send output to a
                                    new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsevds_(&three);               /* Disable mouse input */
 gsevwt_( &zero, data );        /* Flush ring buffer data,
                                   if any */

 gsclrs_();                     /* Clear screen to color zero */

 lcolor = 1;                    /* Use color number one */
 lstyle = -1;                   /* Leave line style alone */
 gslatt_( &lcolor, &lstyle );

 lop = 3;                       /* Set operation to Replace */
 gslop_( &lop );
```

```
 /* Draw intersecting lines at (x1,y1) */
 x1 = 360;
 y1 = 256;
 x2 = x1 - 200;
 y2 = y1 + 200;
 gsline_( &x1, &y1, &x2, &y2 );
 x2 = x1 + 200;
 y2 = y1 +200;
 gsline_( &x1, &y1, &x2, &y2 );
 x2 = x1 - 200;
 y2 = y1 - 200;
 gsline_( &x1, &y1, &x2, &y2 );
 x2 = x1 + 200;
 y2 = y1 - 200;
 gsline_( &x1, &y1, &x2, &y2 );

 /* Draw several ellipses around center point (cx,cy) */
 cx = 360;
 cy = 256;
 major = 200;                    /* Semi-major axis */
 minor = 100;                    /* Semi-minor axis */
 for ( ang = 0; ang <= 1800; ang += 200 )
 {
  end_ang = 450;
  for( start_ang = 0; start_ang <= 2700+450; start_ang += 450 )
  {
   rc = gseara_( &cx, &cy, &major, &minor,
    &ang, &start_ang, &end_ang );
   ck_rc( "gseara", rc );        /* Check return code */
   end_ang += 450;
  }
 }

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */

/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

arc2.c Example C Language Program

```
/*
arc2.c

This program uses the gsearc subroutine to draw several ellipses
at different angles around a center point.  Before each new
ellipse is drawn, the last ellipse is drawn a second time.  The
program uses the gslop subroutine to set the logical operation to
Exclusive OR so that when each ellipse is drawn the second time,
it erases the ellipse from the frame buffer.  This gives the
illusion that a single ellipse is spinning around the center
point.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   int ring[128], save, outdev;
   int ring_size, rc, lcolor, lstyle, lop;
   int cx, cy, start_x, start_y, end_x, end_y, ang;
   int major, minor, rot;
   int one=1, three=3;
   int data[33];

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                   frame buffer */
 outdev = -1;                   /* Send output to a new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsevds_(&three);

 gsclrs_();

 lcolor = 2;                    /* Use color number two */
 lstyle = -1;                   /* Leave line style alone */
 gslatt_( &lcolor, &lstyle );

 lop = 6;                       /* Set logical operation to
                                   Exclusive OR */
 gslop_( &lop );
```

```
 /* Draw several ellipses around center point (cx,cy) */
 cx = 360;
 cy = 256;
 major = 200;                    /* Semi-major axis */
 minor = 100;                    /* Semi-minor axis */
 start_x = cx + major;
 start_y = cy;
 end_x = start_x;
 end_y = cy;
 rot = 0;                        /* No rotational transformation */

 /* Rotate the semi-major axis for the ellipses 360 degrees */
 for( ang = 0; ang <=3600; ang += 100 )
 {  rc = gsearc_( &cx, &cy, &major, &minor, &ang,
   &start_x, &start_y, &end_x, &end_y, &rot );
  ck_rc( "gsearc", rc );         /* Check return code */

  /* If this is not the last ellipse, redraw the ellipse.
     Since the logical operation is set to Exclusive OR,
     redrawing the ellipse will erase it.*/
  if( ang < 3600 )
  {
   rc = gsearc_( &cx, &cy, &major, &minor, &ang,
    &start_x, &start_y, &end_x, &end_y, &rot );
   ck_rc( "gsearc", rc );        /* Check return code */
  }
 }

 gsevds_(&three);

 gseven_(&one);

 gsevwt_(&one, data );

 gsterm_();
} /* end mainline */

/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions
   of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

arc3.c Example C Language Program

```
/*
arc3.c

This program uses the gscarc subroutine to draw several quarter
circles.  It then uses the gslop subroutine to set the logical
operation to Exclusive OR and redraws the same circles.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[128],dummy;
   int ring_size, initrc, trrc, rc, lcolor, lstyle, lop;
   int tcolor,tpage,tbase,tfont;
   int *bit, n,m,h,w,xo,yo,per,loopcnt, fd;
   int arcx, arcy, start_x, start_y, end_x, end_y, r;
   int major, minor, rot;
   int x2,y2,x3,y3;
   char buf[100];
   int x[128],y[128];
   int one=1, three=3;
   int data[33];
   int ncolors, redtab[16], greentab[16], bluetab[16];

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 256;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                   frame buffer */
 outdev = -1;                   /* Send output to a new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsevds_(&three);               /* Disable mouse input */
 gsevwt_( &zero, data );        /* Flush ring buffer data, if any */


 gsclrs_();                     /* Clear screen to color zero */

 lop = 3;                       /* Set operation to Replace */
 gslop_( &lop );
```

```
/* Draw several arcs in different colors */
lcolor = 5;                    /* Start with color five */
lstyle = -1;                   /* Leave line style at default */
arcx = 719;
arcy = 511;
for( r = 1; r < 511; r += 2 )
{
 if ( lcolor > 16 )
  lcolor = 1;
 gslatt_( &lcolor, &lstyle );
 lcolor++;
 start_x = arcx - r;
 start_y = arcy;
 end_x = arcx;
 end_y = arcy - r;
 rc = gscarc_( &arcx, &arcy, &r, &start_x, &start_y,
  &end_x, &end_y );
 ck_rc( "gscarc", rc );        /* Check return code */
}

/* Wait for keystroke event */
gseven_(&one);
gsevwt_( &one, data );

lop = 6;                       /* Set operation to Exclusive OR */
gslop_( &lop );

/* Draw the same arcs with the Exclusive OR operation */
lcolor = 5;                    /* Start with color five */
lstyle = -1;                   /* Leave line style at default */
for( r = 1; r < 511; r += 2 )
{
 if ( lcolor > 16 )
  lcolor = 1;
 gslatt_( &lcolor, &lstyle );
 lcolor++;
 start_x = arcx - r;
 start_y = arcy;
 end_x = arcx;
 end_y = arcy - r;
 rc = gscarc_( &arcx, &arcy, &r, &start_x, &start_y,
  &end_x, &end_y );
 ck_rc( "gscarc", rc );        /* Check return code */
}

/* Wait for keystroke event, then exit */
gseven_(&one);
gsevwt_( &one, data );

gsterm_();
} /* end of main */
```

```
/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

arc4.c Example C Language Program

```
/*
arc4.c

This program uses the gscmap subroutine to make a color table with
four colors.  The gsclrs subroutine clears the screen and changes
it to the background color (color 0).  It then draws several
circular arcs in different colors with the gscrca subroutine.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[128],dummy;
   int ring_size, initrc, trrc, rc, lcolor, lstyle, lop;
   int tcolor,tpage,tbase,tfont;
   int *bit, n,m,h,w,xo,yo,per,loopcnt, fd;
   int arcx, arcy, start_x, start_y, end_x, end_y, r;
   int major, minor, rot;
   int x2,y2,x3,y3;
   char buf[100];
   int x[128],y[128];
   int one=1, three=3;
   int data[33];
   int ncolors, redtab[16], greentab[16], bluetab[16];

 sync();
 signal( SIGQUIT, die );          /* Catch QUIT signal */
 signal( SIGINT , die );          /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 256;                 /* Input ring buffer size */
 save = -1;                       /* Save and restore the
                                     frame buffer */
 outdev = -1;                     /* Send output to a new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsevds_(&three);                 /* Disable mouse input */
 gsevwt_( &zero, data );          /* Flush ring buffer data, if any */
```

```
 /* Set up a color table with the gscmap subroutine */
 ncolors = 4;                    /* Define 4 colors */
 redtab[0] = 0;                  /*   color 0 is blue,
                                     (the background color) */
 greentab[0] = 0;
 bluetab[0] = 0x3fff;
 redtab[1] = 0x3fff;             /*   color 1 is red */
 greentab[1] = 0;
 bluetab[1] = 0;
 redtab[2] = 0x3fff;             /*   color 2 is white */
 greentab[2] = 0x3fff;
 bluetab[2] = 0x3fff;
 redtab[3] = 0x3fff;             /*   color 3 is yellow */
 greentab[3] = 0x3fff;
 bluetab[3] = 0;
 gscmap_( &ncolors, redtab, greentab, bluetab );

 gsclrs_();                      /* Clear screen to color zero */

 lop = 6;                        /* Set operation to Exclusive OR */
 gslop_( &lop );

/* Draw a full circle by drawing several circular
    arcs in different colors using the gscrca subroutine */
 lcolor = 1;                     /* Start with color one */
 lstyle = -1;                    /* Use default line style */
 arcx = 360;                     /* Center of circle */
 arcy = 256;
 r = 250;                        /* Radius of circle */

 /* Loop through 360 degrees in ten degree arcs */
 for( start_x = 0; start_x <= 3600; start_x += 100 )
 {
  if ( lcolor > 3 )              /* If new color is greater
                                     than color table defined, */
   lcolor = 1;                   /*  go back to first color */
  gslatt_( &lcolor, &lstyle );   /* Set new color */
  lcolor++;                      /* Change color for next arc */
  end_x = 1800;                  /* Define the end angle */
  /* Draw the arc */
  rc = gscrca_( &arcx, &arcy, &r, &start_x, &end_x );
  ck_rc( "gscrca", rc );         /* Check return code */
 }

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */
```

```
/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

arc5.c Example C Language Program

```
/*
arc5.c

This program defines an elliptical arc in terms of its center,
semi-major and semi-minor axes, and beginning and ending points.
It uses the gsecnv subroutine to convert the arc definition into a
set of points that are then used by the gspoly subroutine to draw
the arc.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>
#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[128], dummy;
   int indat[13];
   int one=1, three=3;
   int ring_size, rc, lcolor, lstyle;
   int *display;
   int w, h, cx, cy;
   int start_x, start_y, end_x, end_y, r, rot, pre, len;
   int major,minor;
   int x[600],y[600];

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                   frame buffer */
 outdev = -1;                   /* Send output to a  new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsevds_(&three);               /* Disable mouse input */
 gsevwt_( &zero, data );        /* Flush ring buffer data, if any */

 gsclrs_();                     /* Clear screen to color zero */
```

```
/* Retrieve information about the display */
 display = (int *)malloc(sizeof(int)*32);
 gsqdsp_(display);
 display += 1;
 w = *display++;                    /* Display width in pixels */
 h = *display++;                    /* Display height in pixels */
 cx = w / 2;                        /* Center of display */
 cy = h / 2;

 /* Define an elliptical arc */
 major = cx - 10;                   /* Semi-major axis */
 minor = cy - 20;                   /* Semi-minor axis */
 r = 0;
 start_x = cx + major;              /* Specify a quarter of
                                       an ellipse */
 start_y = cy;
 end_x = cx;
 end_y = cy + minor;
 rot = 0;
 len = 599;
 pre = 512;

/* Convert the arc into a set of points */
 rc = gsecnv_( &cx, &cy, &major, &minor, &r,
  &start_x, &start_y, &end_x, &end_y,
  &rot, &len, x, y, &pre );
 ck_rc( "gsecnv", rc );             /* Check return code */

 rc = gspoly_( &len, x, y );
 ck_rc( "gspoly", rc );             /* Check return code */

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */


/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

blit.c Example C Language Program

```
/*
blit.c

This program uses the gsbply subroutine and the gseply subroutine to
 designate groups of shapes to fill.  It draws shapes with other
(non-fill) XGSL subroutines.  The program then uses the gsblit
subroutine to draw 15 colored rectangles.
*/
#include <stdio.h>

#define dim 4
int xpts1[dim]  = { 400, 100, 100, 400 };
int ypts1[dim]  = { 100, 100, 400, 400 };
main()
{
   int (*routine)() = NULL ;
   int buffer[512] ;
   int *display;
   int outdev;
   int sav_res = 0;
   int rsvd = 0;
   int colr;
   int hatch;
   int numvtx = 4;
   int err;
   int plmask;
   int zero = 3;
   int size = 512;
   int devid;
   int i,j;
   int one=1, three=3;
   int data[33];
   int srcpix[32];
   int dstpix[32];
   int wid;
   int hgt;
   FILE *fl_ptr ;
   unsigned char pixdata[1024];
   unsigned char pixaray[1024];

 display = (int *)malloc(sizeof(int)*32); /* Reserve memory
                                             for display
                                             characteristics */
 fl_ptr = fopen("out.file","w") ;       /* Open a file to save
                                             the display
                                             characteristics */

 outdev = -1;
 /* Initialize XGSL */
 if ((err = gsinit_(buffer, &size, &sav_res,
  routine, routine, &outdev)) == 0)
 {
```

```
gsqdsp_(display);
devid = *display;
fprintf(fl_ptr,"Display ID = %d\n",*display++) ;
fprintf(fl_ptr,"Viewable width of frame buffer (pixels) = %d\n",
 *display++) ;
fprintf(fl_ptr,"Viewable height of frame buffer (pixels) = %d\n",
*display++) ;
fprintf(fl_ptr,"Physical width of display (mm) = %d\n",
 *display++) ;
fprintf(fl_ptr,"Physical height of display (mm) = %d\n",
 *display++) ;
fprintf(fl_ptr,"Number of bitplanes = %d\n",*display++) ;
fprintf(fl_ptr,
"Frame buffer orginization (0 = plane, 1 = pixel) = %X\n",
 *display++) ;
fprintf(fl_ptr,"Number of bits for Red DAC = %d\n",
 *display++) ;
fprintf(fl_ptr,"Number of bits for Green DAC = %d\n",
 *display++) ;
fprintf(fl_ptr,"Number of bits for Blue DAC = %d\n",
 *display++) ;
fprintf(fl_ptr,"Minimum cursor width(pixels) = %d\n",
 *display++) ;
fprintf(fl_ptr,"Minimum cursor height(pixels) = %d\n",
 *display++) ;
fprintf(fl_ptr,"Maximum cursor width(pixels) = %d\n",
 *display++) ;
fprintf(fl_ptr,"Maximum cursor height(pixels) = %d\n",
 *display++) ;
fprintf(fl_ptr,"Color table size = %d\n",*display++) ;
fprintf(fl_ptr,"Font class (1 = compressed , 2 = not) = %d\n",
 *display++);

/* Set the fill characteristics */
colr = 1;
hatch = 3;
gsfatt_(&colr,&hatch,&rsvd);

/* Draw a filled polygon */
gsbply_();                      /* Define beginning of polygon */
gspoly_(&numvtx,xpts1,ypts1); /* Draw a shape */
cx = 400;
cy = 250;
cr = 150;
bx = 400;
by = 400;
ex = 400;
ey = 100;
gscarc_(&cx,&cy,&cr,&bx,&by,&ex,&ey);
gseply_();                      /* Complete the polygon,
                                    and fill it */

/* Set new fill characteristics */
colr = 3;
hatch = 2;
gsfatt_(&colr,&hatch,&rsvd);
```

```
  /* Draw another filled polygon */
  gsbply_();                    /* Define beginning of polygon */
  cx = 400;                     /* Draw a circle */
  cy = 250;
  cr = 140;
  bx = 400;
  by = 110;
  ex = 400;
  ey = 110;
  gscarc_(&cx,&cy,&cr,&bx,&by,&ex,&ey);
  gseply_();;                   /* Complete the polygon,
                                    and fill it */

  /* Wait for keystroke event */
  gseven_(&one);
  gsevwt_( &one, data );
  gsclrs_();

  /* Do a block transfer from memory to the frame buffer */
  /* Set up source & destination pixel maps */
  srcpix[0] = 0;                /* From memory */
  dstpix[0] = devid;            /* To frame buffer */
  srcpix[1] = 16;               /* Pixel format, lower-left
                                   coordinate system */

  dstpix[1] = 16;
  srcpix[2] = 32;
  srcpix[3] = 32;
  srcpix[4] = 8;
  srcpix[9] = (int)pixaray;
  dstpix[7] = 10;
  dstpix[8] = 20;
  wid =  31;
  hgt =  32;

/* Draw 15 rectangles */
  for(i=1;i<16;i++)
  {
   for (j=0;j<1024;j++)
    pixaray[j] = i;             /* Set the bit-planes for each
                                    pixel of each rectangle */

   /* Do a block transfer from memory (pixaray),
        to the frame buffer */
   err = gsxblt_(srcpix,dstpix,NULL,&wid,&hgt,&zero);
   dstpix[7] += 32;
   dstpix[8] += 32;
  }

  /* Wait for keystroke event */
  gseven_(&one);
  gsevwt_( &one, data );

  gsterm_();
  fclose(fl_ptr);
 }
 else
 {
  fprintf(fl_ptr,"error = %d\n",err);
  fclose(fl_ptr) ;
 }
} /* end of main */
```

cir1.c Example C Language Program

```
/*
cir1.c

This program uses the gscir subroutine to draw several concentric
circles in the center of the screen.  It uses the gsqdsp
subroutine to get the display's screen size for calculating the
center of the screen.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[128], save, outdev;
   int one=1, three=3;
   int ring_size, rc, lcolor, lstyle;
   int *display;
   int w, h, cx, cy, n;

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                     frame buffer */
 outdev = -1;                   /* Send output to a new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 display = (int *)malloc(sizeof(int)*32);
 gsqdsp_(display);

 display += 1;
 w = *display++;
 h = *display++;

 cx = w / 2;
 cy = h / 2;

 lcolor = 1;
 lstyle = -1;
```

```
 for( n = 1; n < 255; n = n + 10 )
 {
  if ( lcolor > 16 )
   lcolor = 1;
  gslatt_( &lcolor, &lstyle );
  lcolor++;

  rc = gscir_( &cx, &cy, &n );
  ck_rc( "gscir", rc );
  while(1 <= 0);
 }

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */

/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

cir2.c Example C Language Program

```
/*
cir2.c

This program uses the gsfci subroutine to draw several filled
circles in the center of the screen.  The program uses the
gsevds subroutine to disable mouse input and the gsevwt
subroutine to flush the input ring buffer.  The program also
uses the gsevwt subroutine to wait for a keystroke after it
enables the keyboard with the gseven subroutine.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[100], save, outdev;
   int ring_size, rc, lcolor, lstyle, lop;
   int *display;
   int n, cx, cy;

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                    frame buffer */
 outdev = -1;                   /* Send output to a new window */
 gsinit_( &ring[0], &ring_size, save, 0, 0, &outdev);

 gsevds_(&three);               /* Disable mouse input */
 gsevwt_( &zero, data );        /* Flush ring buffer data, if any */

 gsclrs_();

 display = (int *)malloc(sizeof(int)*32);
 gsqdsp_(display);

 display += 1;
 w = *display++;
 h = *display++;

 cx = w / 2;
 cy = h / 2;
```

```
 lcolor = 2;
 lstyle = 0;

 for( n = 1; n < 255; n = n + 10 )
 {
  if ( lcolor > 15 )
   lcolor = 1;
  if ( lstyle > 5 )
   lstyle = 0;
  gsfatt_( &lcolor, &lstyle );
  lcolor++;
  lstyle++;

  rc = gsfci_( &cx, &cy, &n );
  ck_rc( "gsfci", rc );
 }

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */

/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

curs.c Example C Language Program

```
/*
curs.c

This program uses the gscatt subroutine to set the attributes for
the single-colored cursor and to select the single-colored cursor
for subsequent cursor operations. The cursor operations include
making the cursor visable with the gsmcur subroutine, and making
the cursor invisible with the gsecur subroutine.

Then the program uses the gsmcat subroutine to set the attributes
for the multicolored cursor and to select the multicolored cursor
for subsequent cursor operations. The colors for the multicolored
cursor are initialized with the gscmap subroutine.

The program also uses the gstatt subroutine to set the attributes
for annotated text and the gstext subroutine to draw the text
characters.  The program then uses the gsqfnt subroutine to get
the character size which it uses to calculate the size of a box
to draw the text in.
*/

#include <stdio.h>
#include <gslerrno.h>
#include <fcntl.h>

#define  TRUE    1
#define  FALSE   0
```

```
/* Cursor pattern 1 is a cross */
int   pattern1[32] =
      {
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000
      };
```

```
/* Cursor pattern 2 is a diamond-shaped crosshair pattern */
int   pattern2[32] =
      {
          0x000ff000,
          0x001ff800,
          0x003dbc00,
          0x00799e00,
          0x00f18f00,
          0x01e18780,
          0x03c183c0,
          0x078181e0,
          0x0f0180f0,
          0x1e018078,
          0x3c01803c,
          0x7801801e,
          0xf001800f,
          0xe0018007,
          0xc0018003,
          0xffffffff,
          0xffffffff,
          0xc0018003,
          0xe0018007,
          0xf001800f,
          0x7801801e,
          0x3c01803c,
          0x1e018078,
          0x0f0180f0,
          0x078181e0,
          0x03c183c0,
          0x01e18780,
          0x00f18f00,
          0x00799e00,
          0x003dbc00,
          0x001ff800,
          0x000ff000
      };
```

```
/* Cursor pattern 3 is a checkerboard */
int    pattern3[32] =
       {
           0xf0f0f0f1,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0xf0f0f0f0,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0x0f0f0f0f,
           0x9f0f0f0f
       };
```

```
/* Cursor pattern 4 is multicolored */
int   pattern4[32] =
      {
          0xe007f0f0,
          0x9009f0f0,
          0x8811f0f0,
          0x4422f0f0,
          0x2244f0f0,
          0x1188f0f0,
          0x0990f0f0,
          0x0660f0f0,
          0x0660f0f0,
          0x0990f0f0,
          0x1188f0f0,
          0x2244f0f0,
          0x4422f0f0,
          0x8811f0f0,
          0x9009f0f0,
          0xe007f0f0,
          0xe007f0f0,
          0x9009f0f0,
          0x8811f0f0,
          0x4422f0f0,
          0x2244f0f0,
          0x1188f0f0,
          0x0990f0f0,
          0x0660f0f0,
          0x0660f0f0,
          0x0990f0f0,
          0x1188f0f0,
          0x2244f0f0,
          0x4422f0f0,
          0x8811f0f0,
          0x9009f0f0,
          0xe007f0f0
      };
```

```
/* This is a mask for cursor pattern 4 */
int   p4mask[32] =
      {
          0xe007ffff,
          0xf00fffff,
          0xf81fffff,
          0x7c3effff,
          0x3e7cffff,
          0x1ff8ffff,
          0x0ff0ffff,
          0x07e0ffff,
          0x07e0ffff,
          0x0ff0ffff,
          0x1ff8ffff,
          0x3e7cffff,
          0x7c3effff,
          0xf81fffff,
          0xf00fffff,
          0xe007ffff,
          0xe007ffff,
          0xf00fffff,
          0xf81fffff,
          0x7c3effff,
          0x3e7cffff,
          0x1ff8ffff,
          0x0ff0ffff,
          0x07e0ffff,
          0x07e0ffff,
          0x0ff0ffff,
          0x1ff8ffff,
          0x3e7cffff,
          0x7c3effff,
          0xf81fffff,
          0xf00fffff,
          0xe007ffff
        };

main()
{
   int   ring[128];
   int   ring_size, save, outdev;
   int   zero = 0, one = 1, m_one = -1, three = 3;
   int   i, j, k;
   int   data[33];
   int   cx, cy, xmax, ymax;
   int   ncolors, redtab[16], greentab[16], bluetab[16];
   int   xbl, ybl, xtr, ytr;
   int   fildes;
   int   x, y;
   int   wid, ht;
   int   fg, bg, xo, yo, lop;

 /* Initialize the XGSL and open a new window */
 ring_size = 128;            /* Input ring buffer size */
 save = -1;                  /* Save and restore the frame buffer */
 outdev = -1;                /* Send output to a new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsevds_(&three);            /* Disable mouse input */
 gsevwt_( &zero, data );     /* Flush ring buffer data, if any */
```

```
ncolors = 4;                    /* Set up a 4 color table: */
redtab[0] = 0;                  /*   color 0 is blue, */
greentab[0] = 0;
bluetab[0] = 0x3fff;
redtab[1] = 0x3fff;             /*   color 1 is red, */
greentab[1] = 0;
bluetab[1] = 0;
redtab[2] = 0x3fff;             /*   color 2 is white, */
greentab[2] = 0x3fff;
bluetab[2] = 0x3fff;
redtab[3] = 0x3fff;             /*   color 3 is yellow. */
greentab[3] = 0x3fff;
bluetab[3] = 0;

gscmap_( &ncolors, redtab, greentab, bluetab );

gsclrs_();

/*   Establish first font   */
i = 1;                          /* red text */
j = 6;                          /* large font */
gstatt_( &i, &m_one, &m_one, &j, "" );

gsqfnt_( data );                /* Get current font
                                     characteristics: */
cx = data[7];                   /* char width */
cy = data[8];                   /* char height */

/*   Draw a color filled box    */
gsqdsp_( data );                /* Get display characteristics: */
xmax = data[1];                 /*  screen width  */
ymax = data[2];                 /*  screen height */
xbl = xmax/2 - 17*cx/2;         /* Coordinates of lower left
                                    corner of box */
ybl = ymax/2 - 5*cy/2;
xtr = xbl + 17*cx;              /* Coordinatess of top right
                                    corner of box */
ytr = ybl + 5*cy;

i = 3;
gsfatt_( &i, &m_one, &zero );  /* Set up for yellow box */
gsfrec_( &xbl, &ybl, &xtr, &ytr ); /* Draw filled box */

/*   Write "HELLO" in the box   */
i = xbl + 6*cx;                 /* Coordinates for beginning
                                    of text */
j = ybl + 2*cy;
k = 5;
gstext_( &i, &j, &k, "HELLO" );

/*   Write prompting message    */
i = 2;                          /* White text */
j = 1;                          /* Normal font */
gstatt_( &i, &m_one, &m_one, &j, "" );
gsqfnt_( data );                /* Get current font
                                     characteristics: */
cx = data[7];                   /* char width */
cy = data[8];                   /* char height */
xbl = xmax/2 - 11*cx;           /* Coordinates for beginning
                                    of text */
ybl = 4*cy;
k = 26;                         /* Text length */
gstext_( &xbl, &ybl, &k, "Press any key to continue." );
```

```
/* Set cursor attributes  */
i = 1; wid = 32; ht = 32;
gscatt_(&i, &wid, &ht, pattern1, &zero, &zero);

/* Display first cursor */
gsmcur_();

/* Wait for keystroke event */
gseven_(&one);
gsevwt_( &one, data );

/* Hide the cursor */
gsecur_();

/* Change to new cursor pattern */
i = 2; wid = 32; ht = 32;
gscatt_(&i, &wid, &ht, pattern2, &zero, &zero);
gsmcur_();

/* Wait for keystroke event */
gseven_(&one);
gsevwt_( &one, data );

/* Hide the cursor */
gsecur_();

/* Change to new cursor pattern */
i = 3; wid = 32; ht = 32;
gscatt_(&i, &wid, &ht, pattern3, &zero, &zero);
gsmcur_();

/* Wait for keystroke event */
gseven_(&one);
gsevwt_( &one, data );

/* Hide the cursor */
gsecur_();

/* Change to multicolored cursor */
fg = 1; bg = 2;
wid = 32; ht = 32;
xo = 0; yo = 0;
lop = 6;
gsmcat_( &fg, &bg, &wid, &ht, pattern4, p4mask, &xo, &yo, &lop );

gsmcur_();

/* Wait for keystroke event, then exit */
gseven_(&one);
gsevwt_( &one, data );

gsterm_();
} /* end of main */
```

djpoly.c Example C Language Program

```
/*
djpoly.c

This program uses the gsdjply subroutine to draw several
disjointed polylines on the screen.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"
#include <math.h>;

struct
{
    int fclass;
    int fid;
    int fstyle;
    int fattb;
    int fchar;
    int fbase;
    int fcaps;
    int fwidth;
    int fheight;
    int fustop;
    int fusbot;
    int fresv[21];
} q_font;

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main(argc, argv)

int      argc;
char     *argv[];
{
   extern int errno;
   int ring[200], dummy;
   char stng[100];
   int zero, ml, rc, count, wt;
   int ring_size, initrc, color, lstyle, lop, x, y;
   int npoly, kpoly[50];
   int xs[50], ys[50];
   int q_disp[200], width, height, w2, h2;
   int ii, jj, kk, ll;
   int xmin, xmax, ymin, ymax, index;
   unsigned int nn;
   int one=1, three=3;
   int data[33];
```

```
sync();
signal( SIGQUIT, die );          /* Catch QUIT signal */
signal( SIGINT , die );          /* Catch INTERUPT signal */

/* Initialize the XGSL and open a new window */
ring_size = 128;                 /* Input ring buffer size */
save = -1;                       /* Save and restore the
                                    frame buffer */
outdev = -1;                     /* Send output to a
                                    new window */
gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

gsevds_(&three);                 /* Disable mouse input */
gsevwt_( &zero, data );          /* Flush ring buffer data,
                                    if any */

gsclrs_();                       /* Clear screen to color zero */

/*  display test name  */
gsqdsp_( q_disp );
width = q_disp[1];
w2 = width >> 1;
height = q_disp[2];
h2 = height >> 1;

/* Get font size                                  */
if ((rc = gsqfnt_( &q_font.fclass) ) != 0)
 perror("gsqfnt: ",rc);

sprintf(stng,"Disjoint Polylines \n");
count = strlen(stng);
write(2,stng,count);
x = w2 - ((count >> 1) * q_font.fwidth);
y = h2;

if ((rc = gstext_( &x, &y, &count, stng ) ) != 0 )
 perror("gstext: ",rc);

npoly = 2;
kpoly[0] = 5;
xs[0] = 0;                       ys[0] = 0;
xs[1] = width - 1;               ys[1] = 0;
xs[2] = width - 1;               ys[2] = height - 1;
xs[3] = 0;                       ys[3] = height - 1;
xs[4] = 0;                       ys[4] = 0;
kpoly[1] = 5;
xs[5] = 10;                       ys[5] = 10;
xs[6] = width - 10;               ys[6] = 10;
xs[7] = width - 10;               ys[7] = height - 10;
xs[8] = 10;                       ys[8] = height - 10;
xs[9] = 10;                       ys[9] = 10;

/* Draw the polylines */
if ((rc = gsdjply_( &npoly, kpoly, xs, ys ) ) != GS_SUCC )
 perror("gsdjpoly: ",rc);

/* Wait for keystroke event */
gseven_(&one);
gsevwt_( &one, data );

gsclrs_();
```

```
 /* draw lots of polylines */
 count = 0;
 index = 0;
 xmin = ymin = 0;
 xmax = width - 1;
 ymax = height - 1;
 for( ll=0; ll < 5; ll++ )
 {
  jj = ymax;
  kk = xmax;
  kpoly[ll] = 0;
  ii = ll * 10;
  while( ii < w2 )
  {
   kpoly[ll] += 4;
   xs[index] = ii;          ys[index] = ii;
   xs[index+1] = kk;        ys[index+1] = ii;
   xs[index+2] = kk;        ys[index+2] = jj;
   xs[index+3] = ii+200;    ys[index+3] = jj;

   ii += 200;   jj -= 200; kk -= 200;
   index +=4;
  }
  xmax -= (ll * 10);   ymax -= (ll * 10);
 }
 npoly = 5;

 /*  - line patterns operations                */
 ll = -1;
 for( nn = 0; nn < 3; nn++)
 {
  if ((rc = gslatt_( &ll, &nn ) ) != GS_SUCC )
   perror("gslatt: ",rc);
  if ((rc = gsdjply_( &npoly, kpoly, xs, ys ) ) != GS_SUCC )
   perror("gsdjpoly: ",rc);
  sleep(25);
 }

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */

perror( text, rc )
char *text;
int rc;
{
   char buff[100];

 sprintf(buff,"gsl: %s gslerrno = %d \n",text,rc);
 write(2,buff,strlen(buff));
 return(0);
}
```

ell1.c Example C Language Program

```
/*
ell1.c

This program uses the gsell subroutine to draw several ellipses
on the screen.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[128], save, outdev;
   int ring_size, rc, lcolor, lstyle;
   int n, m, h, w, cx, cy;
   int *display;

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                   frame buffer */
 outdev = -1;                   /* Send output to a
                                   new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsclrs_();

 display = (int *)malloc(sizeof(int)*32);
 gsqdsp_(display);

 display += 1;
 w = *display++;
 h = *display++;

 cx = w / 2;
 cy = h / 2;

 m = 60;
 n = 15;
 lstyle = 6;
 gslop_( &lstyle );
 lcolor = 1;
 lstyle = -1;
```

```
 for ( h = 0; h <= 3600; h = h + 100)
 {
  if ( lcolor > 15 )
   lcolor = 1;
  gslatt_( &lcolor, &lstyle );

  rc = gsell_( &cx, &cy, &m, &n, &h );
  ck_rc( "gsell", rc );

  gsclrs_();
  m += 7;
  n += 4;
  lcolor++;
 }                          /* end for h */

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */

/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

ell2.c Example C Language Program

```
/*
ell2.c

This program uses the gsfell subroutine to draw several filled
ellipses on the screen.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[128], save, outdev;
   int ring_size, rc, lcolor, lstyle;
   int n, m, h, w, cx, cy;
   int *display;

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                   frame buffer */
 outdev = -1;                   /* Send output to a
                                   new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsclrs_();

 display = (int *)malloc(sizeof(int)*32);
 gsqdsp_(display);

 display += 1;
 w = *display++;
 h = *display++;

 cx = w / 2;
 cy = h / 2;

 m = 75;
 n = 30;
 lstyle = 6;
 gslop_( &lstyle );
 lcolor = 1;
 lstyle = -1;
```

```
 for ( h = 0; h <= 15; h++)
 {
  if ( lcolor > 16 )
   lcolor = 1;
  gsfatt_( &lcolor, &lstyle );

  rc = gsfell_( &xo, &yo, &m, &n, &h, &per );
  ck_rc( "gsfell", rc );

  if ( h < 20 )
  {
   /* Wait for keystroke event */
   gseven_(&one);
   gsevwt_( &one, data );
  }

  gsclrs_();
  m += 8;
  lcolor++;
 }                                  /* end for h */

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */

/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

fontld.for Example FORTRAN program

```
C The FORTRAN program fontld.for loads 2 fonts using the gstatt
C subroutine.  First a user-defined font and then a system font.
C After loading each font, the program displays a text string
C with the gstext subroutine.
C
C The program uses the gsxptr subroutine to put the pathname of
C the user-defined font into an integer array.  The array
C contains two integers for each font to be loaded.  The first
C integer contains the font ID; the second contains the address
C of the font's path.  In addition, each character string that
C specifies a font path is terminated with a null (0) character.
C
      program fontld.for
      integer buffer(128)
      integer count, size, len, savres, fildes, x, y, rc
      character*25 a
      character*80 msg
      integer fontray(2,2)
      external gsinit_, gsxptr_, gstatt_, gstext_, gsterm_
      msg = 'XGSL Font Demonstration'
C Initialize the XGSL
      size = 0
      savres = -1
      fildes = -1
      call gsinit_ (buffer,size,savres,0,0,fildes)
C Define the number of fonts to be loaded.
      count = 2
C Use 0 as the font ID for any user-defined fonts.
      fontray (1,1) = 0
C Define the path of the user-defined font.
C Put a 0 at the end of the string.
      a = '/yourpath/font1\0'

C Put the path of the user-defined font into the array.
      call gsxptr_ (a,fontray (2,1))
C Specify the font ID of the desired precompiled font.
      fontray (1,2) = 12
C Ensure that the path for the predefined font is NULL
      fontray (2,2) = 0

C Display each of the 2 fonts.
      y = 30
      len = 23
      count = 1
      do 11 x = 50, 110, 20
       y = y + 20
       if (count .eq. 1) then

C       Load the user-defined font.
        rc = gstatt_ (savres,savres,savres,fontray(1,1,),a)
       else if (count .eq. 2) then
C       Load the predefined system font.
        rc = gstatt_ (savres,savres,savres,
     +         fontray(1,2),fontray(2,2))
       endif
```

```
C       Dsiplay the text in the current font.
        rc = gstext_ (x,y,len,msg)
        count = count + 1
        if  (count .eq. 3) then
         count = 1
        endif
11     continue
C Terminate the XGSL and exit.
       rc = gsterm_ ()
       stop
       end
```

gtex.c Example C Language Program

```
/*
gtex.c

This program uses the gsgtat subroutine to specify a geometric-
text font and the relevant attributes, and the gsgtxt subroutine
to draw several geometric-text characters.
*/
#include <signal.h>
#include <stdio.h>
#include <fcntl.h>
#include <termio.h>

#define GSPFONT1_0    "/etc/vtm/nrmMP1.9x20"  /* Normal font
                                                 9 x 20 pels*/
#define GSPFONT2_0    "/etc/vtm/itlMP1.9x20"  /* Italic font
                                                 9 x 20 pels*/
#define GSPFONT3_0    "/etc/vtm/bldMP1.9x20"  /* Bold font
                                                 9 x 20 pels  */
#define GSPFONT4_0    "/etc/vtm/nrmMP1.8x14"  /* Normal font
                                                 8 x 14 pels*/
#define GSPFONT5_0    "/etc/vtm/nrmMP1.4x8"   /* Normal font
                                                 4 x 8  pels*/
#define GSPFONT6_0    "/etc/vtm/nrmMP1.18x40" /* Normal font
                                                 18 x 40 pels*/
#define GSPFONT7_0    "/etc/vtm/nrmMP1.12x30" /* Normal font
                                                 12 x 30 pels*/
#define GSPFONT8_0    "/etc/vtm/ergMP1.9x20"  /* Ergon font
                                                 9 x 20  pels*/
#define GSPGEOM_01    "/etc/vtm/geofont.mp"   /* Geometric font
                                                 for test */

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   int ring[100];
   int ring_size, txrc, rc, lncolor, lnpatrn;
   int *bit, cmap, n,m, loopcnt;
   char buf[100];
   int m1 = -1;
   int zero = 0;
```

```
   int txt_col;                  /* TEXT ATTRIBUTES      */
   int txt_dir;
   int txt_pre;
   int txt_expan;
   int txt_spac;
   int txt_tran;
   int txt_2byte;
   int txt_high;
   int txt_upvx;
   int txt_upvy;
   int txt_algnhz;
   int txt_algnvt;
   int font_id;

#define  fonttmp  "/etc/vtm/geofont.mp"
   int reds[256],greens[256],blues[256];
   int x1,x2,y1,y2, i, j;
   short xs[100],ys[100];

 sync();
 signal( SIGQUIT, die );         /* Catch QUIT signal */
 signal( SIGINT , die );         /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;                /* Input ring buffer size */
 save = -1;                      /* Save and restore the
                                     frame buffer */
 outdev = -1;                    /* Send output to a
                                     new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsevds_(&three);                /* Disable mouse input */
 gsevwt_( &zero, data );         /* Flush ring buffer data, if any */

 gsclrs_();                      /* Clear screen to color zero */

 /* Paint text in a loop */
 x1 = 50;
 y1 =  800;
 x2 = 30;

 txt_col = 14;
 txt_dir  =  -1;
 txt_pre  = 2;
 txt_expan  = 0x80000000;
 txt_spac  = 0x80000000;
 txt_high = -1;

 txt_upvx = 0x80000000;
 txt_upvy = 0x80000000;
 txt_algnhz = -1;
 txt_algnvt = -1;

 font_id = 0x0409;

 txrc = gsgtat_( &txt_col, &txt_dir, &txt_pre, &txt_expan,
  &txt_spac, &txt_high, &txt_upvx, &txt_upvy, &txt_algnhz,
  &txt_algnvt, &font_id,"/u/beau/lpp/geofont.mp");
 ck_rc("gsgtat",txrc);

 for( y1 = 400; y1 >= 300; y1 -= 30)
 {
```

```
  txrc = gsgtxt_(&x1,&y1,&x2,"1234567890123456789 1234567890");
  ck_rc("gsgtxt",txrc);

  txt_col ++;
  if ( txt_col > 15)
   txt_col = 1;
  font_id = -1;
  txrc = gsgtat_( &txt_col, &txt_dir, &txt_pre, &txt_expan,
   &txt_spac, &txt_high, &txt_upvx, &txt_upvy, &txt_algnhz,
   &txt_algnvt, &font_id,"/u/beau/lpp/geofont.mp");
  ck_rc("gsgtat",txrc);
 }

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */

/* ck_rc */
/*
Display non-zero XGSL return codes.
 ertext Name of XGSL subroutine
 erc    Return code from XGSL subroutine
*/
ck_rc(ertext,erc)
int erc;
char *ertext;
{
   char buff[256];
 if (erc != 0)
 {
  sprintf(buff,"gsl: %s: gslerrno = %d\n", ertext, erc);
  write(2,buff,strlen(buff));
  sprintf(buff,"\tSee gslerrno.h for definitions of gslerrno.\n");
  write(2,buff,strlen(buff));
  return(0);
 }
} /* end ck_rc */
```

mark.c Example C Language Program

```
/*
mark.c

This program uses the gsmatt subroutine to set marker attributes,
and the gsplym subroutine to draw markers at several points.
*/
#include <signal.h>
#include <stdio.h>
#include <math.h>
#include <fcntl.h>;
#include "gslerrno.h"

/* Marker is a cross */
int   marker[32] =
      {
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0xffffffff,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000,
          0x000ff000
      };
```

```
/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[128], save, outdev;
   int ring_size, rc, color, style, n;
   int w, h, xo, yo;
   int x[10], y[10];
   int zero=0, one=1, three=3;
   int data[33];

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */


 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                    frame buffer */
 outdev = -1;                   /* Send output to a
                                    new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);


 gsevds_(&three);               /* Disable mouse input */
 gsevwt_( &zero, data );        /* Flush ring buffer data,
                                   if any */

 gsclrs_();                     /* Clear screen to color zero */

 color = 3;                     /* Color of first line */
 style = 5;                     /* Style 5 is an X */
 gsmatt_(&color, &style);

 for (n = 0; n < 10; n++)       /* Plot 10 points */
 {
  x[n] = n * 50 + 10 ;  y[n] = x[n] / 2 + 5;
 }
 n = 10;
 gsplym_(&n, x, y);

 color = 5;                     /* Color of second line */
 style = 4;                     /* Style 4 is a small circle */
 gsmatt_(&color, &style);
 for (n = 0; n < 10; n++)       /* Plot 10 points */
 {
  x[n] = n * 50 + 10 ;  y[n] = x[n] * x[n] / 1020 + 5 ;
 }
 n = 10;
 gsplym_(&n, x, y);
```

```
 color = 7;                          /* Color of third line */
 style = 1;                          /* Style 1 is a small
                                        solid circle */
 gsmatt_(&color, &style);
 for (n = 0; n < 10; n++)            /* Plot 10 points */
 {
  x[n] = n * 50 + 10 ;  y[n] = sqrt( (float) x[n] ) * 12 - 5;
 }
 n = 10;
 gsplym_(&n, x, y);

 color = 15;                         /* Color of end-marker */
 style = 0;                          /* Style 0 is user-defined */
 w = 32; h = 32;                     /* Marker size */
 xo = 15; yo = 15;
 gsmatt_(&color, &style, &w, &h, marker, &xo, &yo);
 n = 1; x[0] = 496; y[0] = 240; /* Mark last point */
 gsplym_(&n, x, y);

 /* Wait for keystroke event, then exit */
 gseven_(&one);
 gsevwt_( &one, data );

 gsterm_();
} /* end of main */
```

pix.c Example C Language Program

```
/*
pix.c

This program uses the gsrsav subroutine to save a block of
pixels on the screen (actually from the frame buffer) as a bitmap
in memory.  Then it uses the gsrrst subroutine to restore the
block of pixels.  The program uses the gsfatt subroutine to set
the fill attributes for the gsfrec subroutine, which draws filled
rectangles, and the gsfply subroutine, which draws filled
polygons.  The program also uses the gsmult subroutine to draw
several line segments.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>;
#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   int ring[128], save, outdev;
   int ring_size, rc, lcolor, lstyle, lop;
   int zero=0, one=1, three=3;
   int data[33];
   int pi1, pi2, pi3, pi4;
   int ar1[256], ar2[256], ar3[256];
   char bitmap[ 50000 ];
   int ncolors, redtab[16], greentab[16], bluetab[16];

 sync();
 signal( SIGQUIT, die );          /* Catch QUIT signal */
 signal( SIGINT , die );          /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;                 /* Input ring buffer size */
 save = -1;                       /* Save and restore the
                                     frame buffer */
 outdev = -1;                     /* Send output to a
                                     new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);

 gsevds_(&three);                 /* Disable mouse input */
 gsevwt_( &zero, data );          /* Flush ring buffer data,
                                     if any */

 gsclrs_();                       /* Clear screen to color zero */
```

```
ncolors = 4;                    /* set up a 4 color table */
redtab[0] = 0;                  /*   color 0 is blue */
greentab[0] = 0;
bluetab[0] = 0x3fff;
redtab[1] = 0x3fff;             /*   color 1 is red */
greentab[1] = 0;
bluetab[1] = 0;
redtab[2] = 0x3fff;             /*   color 2 is white */
greentab[2] = 0x3fff;
bluetab[2] = 0x3fff;
redtab[3] = 0x3fff;             /*   color 3 is yellow */
greentab[3] = 0x3fff;
bluetab[3] = 0;
gscmap_( &ncolors, redtab, greentab, bluetab );

gsclrs_();                      /* Clear to blue */

/* Draw a background for the clock face */
pi1=2; pi2=0; pi3=0;            /* Solid, white */
gsfatt_(&pi1,&pi2,&pi3);        /* Set fill attribute */

pi1= 558; pi2= 0;               /* Lower left corner */
pi3= 719; pi4= 161;             /* Upper right corner */
gsfrec_( &pi1, &pi2, &pi3, &pi4 );  /* Draw a filled rectangle */

/* Draw a clock face */
lcolor = 1; lstyle = 0;         /* Red, solid lines */
gslatt_( &lcolor, &lstyle );
ar1[0]= 639; ar2[0]= 145;       /* Define tic marks */
ar1[1]= 639; ar2[1]= 153;
ar1[2]= 646; ar2[2]= 150;
ar1[3]= 646; ar2[3]= 152;
ar1[4]= 653; ar2[4]= 149;
ar1[5]= 654; ar2[5]= 151;
ar1[6]= 660; ar2[6]= 147;
ar1[7]= 661; ar2[7]= 149;
ar1[8]= 667; ar2[8]= 144;
ar1[9]= 668; ar2[9]= 146;
ar1[10]= 671; ar2[10]= 136;
ar1[11]= 675; ar2[11]= 143;
ar1[12]= 680; ar2[12]= 137;
ar1[13]= 681; ar2[13]= 139;
ar1[14]= 686; ar2[14]= 132;
ar1[15]= 687; ar2[15]= 134;
ar1[16]= 691; ar2[16]= 127;
ar1[17]= 693; ar2[17]= 128;
ar1[18]= 696; ar2[18]= 121;
ar1[19]= 698; ar2[19]= 122;
ar1[20]= 695; ar2[20]= 112;
ar1[21]= 702; ar2[21]= 116;
ar1[22]= 703; ar2[22]= 108;
ar1[23]= 705; ar2[23]= 109;
ar1[24]= 706; ar2[24]= 101;
ar1[25]= 708; ar2[25]= 102;
ar1[26]= 708; ar2[26]= 94;
ar1[27]= 710; ar2[27]= 95;
ar1[28]= 709; ar2[28]= 87;
ar1[29]= 711; ar2[29]= 87;
ar1[30]= 704; ar2[30]= 80;
ar1[31]= 712; ar2[31]= 80;
ar1[32]= 709; ar2[32]= 73;
```

```
ar1[33]= 711; ar2[33]= 73;
ar1[34]= 708; ar2[34]= 66;
ar1[35]= 710; ar2[35]= 65;
ar1[36]= 706; ar2[36]= 59;
ar1[37]= 708; ar2[37]= 58;
ar1[38]= 703; ar2[38]= 52;
ar1[39]= 705; ar2[39]= 51;
ar1[40]= 695; ar2[40]= 48;
ar1[41]= 702; ar2[41]= 44;
ar1[42]= 696; ar2[42]= 39;
ar1[43]= 698; ar2[43]= 38;
ar1[44]= 691; ar2[44]= 33;
ar1[45]= 693; ar2[45]= 32;
ar1[46]= 686; ar2[46]= 28;
ar1[47]= 687; ar2[47]= 26;
ar1[48]= 680; ar2[48]= 23;
ar1[49]= 681; ar2[49]= 21;
ar1[50]= 671; ar2[50]= 24;
ar1[51]= 675; ar2[51]= 17;
ar1[52]= 667; ar2[52]= 16;
ar1[53]= 668; ar2[53]= 14;
ar1[54]= 660; ar2[54]= 13;
ar1[55]= 661; ar2[55]= 11;
ar1[56]= 653; ar2[56]= 11;
ar1[57]= 654; ar2[57]= 9;
ar1[58]= 646; ar2[58]= 10;
ar1[59]= 646; ar2[59]= 8;
ar1[60]= 639; ar2[60]= 15;
ar1[61]= 639; ar2[61]= 7;
ar1[62]= 632; ar2[62]= 10;
ar1[63]= 632; ar2[63]= 8;
ar1[64]= 625; ar2[64]= 11;
ar1[65]= 624; ar2[65]= 9;
ar1[66]= 618; ar2[66]= 13;
ar1[67]= 617; ar2[67]= 11;
ar1[68]= 611; ar2[68]= 16;
ar1[69]= 610; ar2[69]= 14;
ar1[70]= 607; ar2[70]= 24;
ar1[71]= 603; ar2[71]= 17;
ar1[72]= 598; ar2[72]= 23;
ar1[73]= 597; ar2[73]= 21;
ar1[74]= 592; ar2[74]= 28;
ar1[75]= 591; ar2[75]= 26;
ar1[76]= 587; ar2[76]= 33;
ar1[77]= 585; ar2[77]= 32;
ar1[78]= 582; ar2[78]= 39;
ar1[79]= 580; ar2[79]= 38;
ar1[80]= 583; ar2[80]= 48;
ar1[81]= 576; ar2[81]= 44;
ar1[82]= 575; ar2[82]= 52;
ar1[83]= 573; ar2[83]= 51;
ar1[84]= 572; ar2[84]= 59;
ar1[85]= 570; ar2[85]= 58;
ar1[86]= 570; ar2[86]= 66;
ar1[87]= 568; ar2[87]= 65;
ar1[88]= 569; ar2[88]= 73;
ar1[89]= 567; ar2[89]= 73;
ar1[90]= 574; ar2[90]= 80;
ar1[91]= 566; ar2[91]= 80;
```

```
ar1[92]= 569; ar2[92]= 87;
ar1[93]= 567; ar2[93]= 87;
ar1[94]= 570; ar2[94]= 94;
ar1[95]= 568; ar2[95]= 95;
ar1[96]= 572; ar2[96]= 101;
ar1[97]= 570; ar2[97]= 102;
ar1[98]= 575; ar2[98]= 108;
ar1[99]= 573; ar2[99]= 109;
ar1[100]= 583; ar2[100]= 112;
ar1[101]= 576; ar2[101]= 116;
ar1[102]= 582; ar2[102]= 121;
ar1[103]= 580; ar2[103]= 122;
ar1[104]= 587; ar2[104]= 127;
ar1[105]= 585; ar2[105]= 128;
ar1[106]= 592; ar2[106]= 132;
ar1[107]= 591; ar2[107]= 134;
ar1[108]= 598; ar2[108]= 137;
ar1[109]= 597; ar2[109]= 139;
ar1[110]= 607; ar2[110]= 136;
ar1[111]= 603; ar2[111]= 143;
ar1[112]= 611; ar2[112]= 144;
ar1[113]= 610; ar2[113]= 146;
ar1[114]= 618; ar2[114]= 147;
ar1[115]= 617; ar2[115]= 149;
ar1[116]= 625; ar2[116]= 149;
ar1[117]= 624; ar2[117]= 151;
ar1[118]= 632; ar2[118]= 150;
ar1[119]= 632; ar2[119]= 152;
pi1= 120; gsmult_( &pi1, ar1, ar2 );   /* Draw clock face */

/* Save clock face */
pi1= 558; pi2= 0; pi3= 719; pi4= 161;
gsrsav_( bitmap, &pi1, &pi2, &pi3, &pi4 );

/* Draw a minute hand */
ar1[0]= 592; ar2[0]= 59;
ar1[1]= 646; ar2[1]= 77;
ar1[2]= 646; ar2[2]= 77;
ar1[3]= 642; ar2[3]= 87;
ar1[4]= 642; ar2[4]= 87;
ar1[5]= 592; ar2[5]= 59;
pi1= 6; gsmult_( &pi1, ar1, ar2 );

/* Erase lines for the hour hand */
pi1=2; pi2=0; pi3=0;                   /* Erase to white */
gsfatt_(&pi1,&pi2,&pi3);

ar1[0]=666; ar2[0]=70;
ar1[1]=636; ar2[1]=86;
ar1[2]=633; ar2[2]=77;
ar1[3]=666; ar2[3]=70;
pi1=0x4; gsfply_(&pi1,ar1,ar2);        /* Fill hour hand area */

/* Draw an hour hand */
ar1[0]= 666; ar2[0]= 70;
ar1[1]= 636; ar2[1]= 86;
ar1[2]= 636; ar2[2]= 86;
ar1[3]= 633; ar2[3]= 77;
ar1[4]= 633; ar2[4]= 77;
ar1[5]= 666; ar2[5]= 70;
pi1= 6; gsmult_( &pi1, ar1, ar2 );
```

```
/* Wait for keystroke event */
gseven_(&one);
gsevwt_(&one, data );

pi1= 558; pi2= 0; pi3= 719; pi4= 161;  /* Restore clock face */
gsrrst_(bitmap,&pi1,&pi2,&pi3,&pi4);

pi1=1; pi2=0;
gslatt_(&pi1,&pi2);

ar1[0]= 589; ar2[0]= 69;
ar1[1]= 645; ar2[1]= 76;
ar1[2]= 645; ar2[2]= 76;
ar1[3]= 643; ar2[3]= 86;
ar1[4]= 643; ar2[4]= 86;
ar1[5]= 589; ar2[5]= 69;
pi1= 6; gsmult_(&pi1,ar1,ar2);

pi1=2; pi2=0; pi3=0;
gsfatt_(&pi1,&pi2,&pi3);

ar1[0]=666; ar2[0]=69;
ar1[1]=636; ar2[1]=86;
ar1[2]=633; ar2[2]=77;
ar1[3]=666; ar2[3]=69;
pi1=0x4; gsfply_(&pi1,ar1,ar2);

ar1[0]= 666; ar2[0]= 69;
ar1[1]= 636; ar2[1]= 86;
ar1[2]= 636; ar2[2]= 86;
ar1[3]= 633; ar2[3]= 77;
ar1[4]= 633; ar2[4]= 77;
ar1[5]= 666; ar2[5]= 69;
pi1= 6; gsmult_(&pi1,ar1,ar2);

/* Wait for keystroke event, then exit */
gseven_(&one);
gsevwt_( &one, data );

gsterm_();
} /* end of main */
```

xtex.c Example C Language Program

```
/*
xtex.c

This program uses the gsxtat subroutine to specify an xtext font
and the relevant attributes, and the gsxtxt subroutine to draw
several xtext characters.
*/

#include <signal.h>
#include <stdio.h>
#include <fcntl.h>

#include <sys/types.h>;
#include <sys/ipc.h>;
#include <sys/shm.h>;

#define FONT_FILE          "/usr/lpp/fonts/Rom14.500"

extern int gsp_new_xtxt;

#include "gslerrno.h"

/* die - Stop XGSL and this process on receipt of signals */
die(sig)
int sig;
{
 gsterm_();
 kill( getpid( ), sig );
 exit(0);
}

main()
{
   extern int errno;
   int ring[100],dummy;
   int ring_size, initrc, trrc, rc, lcolor, lstyle, lop;
   int tcolor,tpage,tbase,tfont;
   int *bit, n,m,h,w,xo,yo,per,loopcnt, fd;
   int xfg,xbg,xlop,*rtxfont;
   int x1,y1,x2,y2,x3,y3;
   char buff[100];
   char *str;
   int x[128],y[128];
   int new_fdes;
   int clipbox[5];

 sync();
 signal( SIGQUIT, die );        /* Catch QUIT signal */
 signal( SIGINT , die );        /* Catch INTERUPT signal */

 /* Initialize the XGSL and open a new window */
 ring_size = 128;               /* Input ring buffer size */
 save = -1;                     /* Save and restore the
                                    frame buffer */
 outdev = -1;                   /* Send output to a new window */
 gsinit_( &ring[0], &ring_size, &save, 0, 0, &outdev);
```

```
gsp_new_xtxt = 1;
new_fdes = open(FONT_FILE,O_RDONLY);
/* Convert font file to a mapped file */
  rtxfont = (int *) shmat(new_fdes,0,SHM_MAP | SHM_RDONLY);

 xfg = 3;
 xbg = 5;
 xlop = 3;
 clipbox[1] = 0;
 clipbox[2] = 0;
 clipbox[3] = 1024;
 clipbox[4] = 1024;

 rc = gsxtat_(&xfg, &xbg, &xlop, rtxfont, clipbox);
 x1 = 0;
 str = "This is sample xtext output";
 n = 27;
 for ( m = 0; m < 20; m++ )
 {
  y1 = 1023 - 20;
  for ( h = 0; h < 25; h++ )
  {

   rc = gsxtxt_( &x1, &y1, &n, &str[0] );
    y1 -= 20;
    /* Wait for keystroke event */
    gseven_(&one);
    gsevwt_( &one, data );
   }
  }

 gsterm_();
} /* end mainline      */
```

Special Terms Used in GL

ambient light. Light that reflects off of one or more surfaces in the scene before arriving at the target surface. Ambient light is assumed to be non-directional, and is reflected uniformly in all directions by the reflecting surface. In the GL, ambient light is mocked up by use of ambient terms in the lighting equation, rather than actually computing the reflections.

aspect ratio. The ratio of the height of a primitive to its width. A rectangle of width ten inches and height five inches has an aspect ratio of 10/5 or 2.

asynchronous. Not synchronized in time. For example, input events occur at the whim of the user; the program may read them later.

attribute. A parameter that can affect the appearance of a drawing primitive. For instance, color is an attribute. If the color is set to "RED", it will remain red until changed, and everything that is drawn will be drawn in red. Color is an attribute. Other attributes include linestyle, linewidth, pattern, and font. For a list of attributes and pipeline options, see the **greset** subroutine. See also **pipeline option.**

azimuthal angle. If a primitive is sitting on the ground, with its z coordinate straight up, the azimuthal viewing angle is the angle the observer makes with the y axis in the x-y plane. If the observer walks in a circle with the primitive at the center, the azimuthal angle is the only thing that varies.

B-spline cubic curve. A cubic spline approximation to a set of four control points having the property that slope and curvature are continuous across sets of control points. See also **parametric rational cubic curve.**

backfacing polygon. A polygon whose vertices appear in clockwise order in screen space. If backface culling is enabled, such polygons are not drawn.

basis. In the GL, a curve or patch basis is a 4x4 matrix that controls the relationship between control points and the approximating spline. B-splines, Bezier curves, and Cardinal splines all differ in that they have different bases.

Bezier cubic curve. A cubic spline approximation to a set of four control points that passes through the first and fourth control points, and has a continuous slope where two spline segments meet. See also **parametric rational cubic curve.**

bitplanes. A **bitplane** supplies one bit of color information per pixel on the display. Thus, an eight bitplane system allows 2 to the eighth power different colors to be displayed at each pixel.

blit. Bit block transfer.

Boolean. A value of TRUE or FALSE. TRUE=1 and FALSE=0.

bounding box. A rectangle (2D) that bounds a primitive. A bounding box can be used to determine whether the primitive lies inside a clipping region. See **clipping.**

button. Buttons include those on the keyboard, mouse, lightpen, or buttons on the dial and button box.

Cardinal cubic spline curve. A cubic spline whose endpoints are the second and third of four control points. A series of cardinal splines will have a continuous slope, and will pass through all but the first and last control points. See also **parametric rational cubic curve.**

clipping. If a primitive overlaps the boundaries of a window, it is *clipped.* The part of a primitive that appears in the window is displayed and the rest is ignored. There are several types of clipping that occur in the system. Three-D drawing primitives are clipped to the boundaries of a frustum (for perspective transformations) or to a rhombohedron (for orthographic projections). This 3-D clipping applies as well to the origin of character strings, but not to the characters themselves. A 2-D clipping is also performed; all drawing is clipped to the boundaries of the AIXwindow. The area of 2-D clipping can be controlled with the screenmask. See **clipping planes, fine clipping, gross clipping, screenmask, transformation, window.**

clipping planes. Before clipping occurs, primitive space is mapped to normalized device coordinates. The clipping planes x=±w; y=±w; or z=±w correspond to the left, right, top, bottom, near, and far planes bounding the viewing frustum. See **frustum**.

color map. A lookup table that translates color indexes into RGB triplets. The lookup table is sandwiched between the frame buffer and the digital-to-analog converters (DACs) and serves to translate the color index value stored in the frame buffer into the red, green, and blue values required by the DACs. On most hardware configurations, the color map is either 8 or 12 bits deep, allowing the simultaneous display of 256 or 4096 colors. On most hardware configurations, the DACs have an 8-bit per color accuracy, allowing the user to choose among 16,777,216 colors.

color map mode. A configuration of the hardware that passes the values stored in the frame buffer through a color lookup table (color map), from which the red, green, and blue values are obtained for display. Entries in the color map are referred to as color indexes. In color map mode, the values stored in the frame buffer are treated as color map indexes. See **RGB mode**.

color ramp. A progression of colors in a color map. Most color ramps are smooth, and have only a small number, if any, of dixcontinuities. For instance, if the full set of colors of the rainbow were loaded into the color map, that would constitute a color ramp.

concatenation. In the GL, concatenation refers to combining a series of geometric transformations; rotations, translations, and scaling. Concatenation of transformations corresponds to matrix multiplication.

concave and convex polygons. A polygon is convex if the line segment joining any two points in the figure is completely contained within the figure. Nonconvex polygons are sometimes called concave. Algorithms that render only convex polygons are much simpler than those that can render both convex and concave polygons.

control points. Points in real space that control the shape of a spline curve. The system provides hardware support for wire frame rational cubic splines, and for NURBS surfaces, the specifications of which require four control points.

culling. If a primitive is smaller than the minimum size specified in the command, it is *culled*: no further commands in the primitive are interpreted. See **clipping, pruning**.

current character position. The two-dimensional screen coordinates where the next character string or pixel read/write will occur.

current color. The color that is employed to color all subsequent drawing primitives. All drawing primitives are drawn with this color until it is changed.

current graphics position. The homogeneous three-dimensional point from which geometric drawing commands will draw. The current graphics position is not necessarily visible.

current window. The window to which the system directs the output from graphics routines. See also **window**.

current transformation matrix. The transformation matrix on top of the matrix stack. All points passed through the graphics pipeline are multiplied by the current transformation matrix before being passed on. The current transformation matrix is a concatenation of the current modeling and viewing matrices. See **transformation**.

cursor. A primitive such as an arrowhead which can be moved about the screen by means of an input device (typically a mouse).

cursor glyph. A 16x16 or 32x32 raster pattern (bitmap that determines the shape of the cursor. A GL cursor glyph can be one or two bits deep; thus, a GL cursor can use up to three colors. Color 0 is always transparent.

depth-cueing. Varying the intensity of a line with z-depth. Typically, the points on the line further from the eye are darker, so the line seems to fade into the distance.

device. A valuator, button, or the keyboard. Buttons have values 0 or 1 (up or down); valuators (mouse, dials) return values in a range, and the keyboard returns ASCII values.

dial and switch box. An I/O device with 8 dials (valuators) and 32 switches. The switch box is also called a "button box" or the "lighted programmable function keys (LPFKs)."

digital-to-analog converter (DAC). A highly specialized chip that converts the digital values coming out of the fram buffer into the rapidly varying analog voltage levels that are required by the monitor.

display list (object). Also called an object. It is a sequence of drawing commands that have been compiled into a unit. Conceptually, a display list is like a macro: it can be invoked multiple times simply by referring to its name. The object can be instantiated at different locations, sizes, and orientations by appropriate use of the transformation matrices. For instance, series of polygons arranged in the shape of a bolt can be compiled into an object. The bolt can then be drawn multiple times by invoking its display list.

dithering. A technique of interleaving dark and light pixels so that the resulting image looks smoothly shaded when viewed from a distance.

double buffer mode. A mode in which two buffers are alternately displayed and updated. A new image can be drawn into the back buffer while the front buffer (containing the previous image) is displayed. See **single buffer mode**.

event queue. A queue that records changes in input devices: buttons, valuators, and the keyboard. The event queue provides a time-ordered list of input events.

eye coordinates, eye space. The coordinate system in which the viewer's eye is located at the origin, and thus all distances are measured with respect to the eye. Viewing transformations map from world coordinates into eye coordinates, and projection transformations map from eye coordinates to normalized device coordinates. Also called viewing coordinates or viewer coordinates. See **modeling coordinates**, **world coordinates**, **screen coordinates**, **transformation**.

field of view. The extent of the area which is under view. The field of view is defined by the **viewing matrix** in use.

fine clipping. Fine clipping masks all drawing commands to a rectangular region of the screen. It would be unnecessary except for the case of character strings. The origin of a character string after transformation may be clipped out by gross, or 3-D, clipping, and the string would not be drawn. By doing gross clipping with the viewport and fine clipping with the screen masks, strings can be moved smoothly off the screen to the left or bottom. See **gross clipping**.

font. A set of characters in a particular style. See **raster font**, **primitive font**.

forward difference matrix. A 4x4 matrix that is iterated by adding each row to the next and the bottom row is output as the next point. Points so generated generally fall on a rational cubic curve.

frame buffer. A quantity of video RAM (VRAM) that is used to store the image displayed on the monitor.

The frame buffer is the electronic canvas on which every drawing primitive is drawn. It is one of the last stops in the graphics pipeline, where the final image resides in the form of digitally coded intensities and brightnesses. These are converted into analog voltage signals 60 times a second and sent to the electron guns of the monitor.

The dimensions of the frame buffer can be changed with GL; typically, the main frame buffer might be 1024 vertical by 1280 horizontal by 8 color bits. The overlay planes might by 1024x1280x2. The z-buffer is considered a frame buffer, although it is not directly visible from the monitor. (There is no direct means of displaying the contents of the z-buffer, although this can be done indirectly.) The size of the z-buffer is typically 1024x1280x24. The cursor is a very specialized form of a frame buffer; one which can move around. The typical cursor is 32x32x2 in size.

front and back buffers. In double buffer mode, the main frame buffer bitplanes are separated into two sets: the front and back buffers. Bits in the front buffer planes are visible and those in the back buffer are not. Typically, an application draws into the back buffer and views the front buffer for dynamic graphics.

frustum. A truncated, four-sided pyramid; that is, a pyramid with the point cut off. In a perspective projection, the shape of the clipping volume is a frustum. The bottom of the frustum is referred to as the far clipping plane, the top of the frustum is the near clipping plane, and the sides are respectively the top, left, bottom, and right clipping planes. In an orthographic projection, the clipping volume is a parallelepiped.

gamma correction. A logarithmic assignment of intensities to lookup table entries for shading applications. This is required since the human eye perceives intensities logarithmically rather than linearly.

gamma ramp. A set of three lookup tables, one for each of the colors red, green, and blue, attached to the electron guns of the monitor. Entries in the gamma lookup table can be set to adjust for variations in the phosphor quality between different brands of monitors. Usually, a logarithmic curve is loaded into the gamma lookup tables. (See **gamma correction**.) The gamma lookup tables are not a subset of the color map tables, but a separate entity.

Gouraud shading. A method of shading polygons smoothly based on the intensities at their vertices. The color is uniformly interpolated along each edge, and then the edge values are uniformly interpolated along each scan line. For realistic shading, colors should be gamma corrected.

graphics pipeline. The sequence of steps that a graphics primitive goes through before it becomes visible on the screen:

- transformation from model coordinates to NDC coordinates
- 3-D clipping (if out of bounds)
- perspective division
- determination of color through lighting equations or depth-cueing
- transformation NDC coordinates to screen coordinates
- 2-D clipping (by the screenmask)
- rasterization (drawing into the frame buffer)
- display of frame buffer.

gross clipping. Also known as 3-D clipping. This is the clipping that occurs in normalized device coordinates, against the sides of the perspective frustum. All 3-D primitives undergo this clipping; in particular, the origin of text strings (but not inidvidual letters) are clipped in this way. See **clipping planes**, **fine clipping**.

hidden surface. A surface of a geometric primitive that is not visible because it is obscured by other surfaces. See **z-buffering**.

hit. Also called pick hit or select hit. A hit occurs whenever a drawing primitive draws within the picking or selecting region. A hit is reported back to the user only if the name stack has changed since the last hit. In other words, multiple hits may occur although only one pick/select even is reported. See **name stack**, **picking**, **selecting**.

homogeneous coordinates. A four-dimensional method of representing three-dimensional space. A point (x, y, z, w) in homogeneous coordinates is used to represent a point (X, Y, Z) in three-dimensional space by taking X=x/w, Y=y/w, Z=z/w.

immediate mode. In this mode, graphics commands are executed immediately rather than being compiled into a display list.

instantiate. To make an instance of. To replicate.

linear interpolation. A method of approximating data values by assuming that they lie along a straight line. Typically, the two end data points are known. For example, if A is the value at a, and B is the value at b, and a<t<b, then the value C at t is (from the two-point formula):

$$C(t) = \frac{(B - A)}{(b - a)} \times (t - d) + A$$

linestyle. The pattern used to draw a line. A linestyle might be solid or broken into a pattern of dashes.

linewidth. The width of a line in pixels.

matrix stack. A stack of matrices with hardware and software support. The top matrix on the stack is the current transformation matrix, and all points passed through the graphics pipeline are multiplied by that matrix. It is a concatenation of the current modeling and viewing transformations. See **current transformation matrix**.

mirroring. The creation of a mirror image of a primitive.

modeling coordinates, modeling space. The coordinate system in which all drawing primitives do their drawing. The user can select the position and orientation of the modeling space with regard to the world space by means of translations, rotations, scales, or generalized transformations. The relation between modeling coordinates and world coordinates is determined by the modeling matrix. Modeling coordinates are a useful conceptual device when drawing complex or repetitive scenes. For instance, a paper clip can be defined once in modeling coordinates, and then drawn hundreds of times by moving the modeling coordinates around in world space. See **eye coordinates, screen coordinates, world coordinates, transformation.**

name stack. A stack of 16-bit integers, controllable by the user, used to establish what drawing primitive caused a pick or select event. The name stack is written into the pick/select event buffer every time a pick or select event occurs. The entire event buffer is returned to the user at the end of the pick/select episode.

normalized device coordinates (NDC). Coordinates in the range from –1 to 1. All primitives that draw within the unit cube are visible on the screen (unless masked by the screenmask). See **transformation, unit cube.**

NTSC. A video display and timing format that is the American broadcast standard. Most video tape recorders record and play back NTSC signals. Specialized hardware is required to convert from RGB monitor outputs to an NTSC signal.

null-terminated. Having a zero byte at the end. In the C language, character strings are stored this way internally.

NURBS. (Non-Uniform Rational B-spline). A parametric surface that can be trimmed with non-uniform rational B-spline curves and piecewise linear curves.

object. Also called a display list. It is a sequence of drawing commands that have been compiled into a unit. Conceptually, a display list is like a macro: it can be invoked multiple times simply by referring to its name. The object can be instantiated at different locations, sizes, and orientations by appropriate use of the transformation matrices. For instance, series of polygons arranged in the shape of a bolt can be compiled into an object. The bolt can then be drawn multiple times by invoking its display list.

object space. The space in which a graphics object is defined. A convenient point is chosen as the origin and the object is defined relative to this point. When an object is rendered by a call to the **callobj** subroutine, it is rendered in modeling coordinates, and the object space becomes (for that moment) the same as the modeling space.

orthographic projection. A representation in which the lines of a projection are parallel. Orthographic projections lack *perspective foreshortening* and its accompanying sense of depth realism. Because they are simple to draw, orthographic projections are often used by draftsmen. See **perspective projection.**

parametric bicubic surface. A surface defined by the equation:

$$x(u,v) = a_{11}u^3v^3 + a_{12}u^3v^3 + a_{13}u^3v + a_{14}u^3$$
$$+ a_{21}u^2v^3 + a_{22}u^2v^2 + a_{23}u^2v + a_{24}u^2$$
$$+ a_{31}uv^3 + a_{32}uv^2 + a_{33}uv + a_{34}u$$
$$+ a_{41}v^3 + a_{42}v^2 + a_{43}v + a_{44}$$

The equations for y and z are similar.

The points on a bicubic patch are defined by varying the parameters u and v from 0 to 1. If one parameter is held constant and the other is varied from 0 to 1, the result is a cubic curve. If $x(u,v)=1$ for all u,v, the bicubic surface is called "ordinary," but if $x(u,v)$ varies as a function of u,v, then the surface is called "rational." See also **homogeneous coordinates.**

parametric rational cubic curve. A curve defined by the equation:

$$x(t) = a_xt^3 + b_xt^2 + c_xt + d_x$$

$$y(t) = a_yt^3 + b_yt^2 + c_yt + d_y$$

$$z(t) = a_zt^3 + b_zt^2 + c_zt + d_z$$

$$w(t) = a_wt^3 + b_wt^2 + c_wt + d_w$$

where x, y, z, and w are cubic polynomials. The parameter t typically varies between 0 and 1. Such a curve is considered rational only if a(w), b(w), or c(w) is not equal to 0; otherwise, it is simply an ordinary parametric curve.

patch. A parametric bicubic surface.

pattern. A 16x16, 32x32, or 64x64 array of bits defining the texturing of polygons on the system display.

perspective projection. Perspective projection is a technique used to achieve realism when drawing primitives. In a perspective projection, the lines of projection meet at the viewpoint; thus the size of a primitive varies inversely with its distance from the source projection. The farther a primitive or part of a primitive is from the viewer, the smaller it will be drawn. This effect, known as *perspective foreshortening*, is similar to the effect achieved by photography and by the human visual system. See **orthographic projection**.

picking. A method for finding out what primitives are being drawn near the cursor on the display screen. See **picking region, selecting, selecting region, name stack**.

picking region. A rectangular volume around the cursor that is sensitive to picking events. If a drawing primitive draws within this volume, a pick event is reported. The width and height of the region can be set by the user. If the z-buffer is enabled, the depth of the region is the entire z-buffer. See **hit, picking, selecting region**.

piecewise linear curve. A list of coordinate pairs in the parameter space for the non-uniform rational B-spline (NURBS) surface. These points are connected with straight lines to form a path.

pipeline options. Variables that control the flow of processing in the graphics pipeline. For instance, lighting is a pipeline option: if lighting is turned on, the color of a primitive is obtained by evaluating the lighting equations, and if lighting is turned off, the last color specified is used. Other pipeline options are the backfacing flag, the shademodel flag, the depthcueing flag, the picking flag, the colormode (color index or RGB) flag, the z-buffer flag (enables or disables drawing to the z-buffer), and so on. See attributes.

pixel. A rectangular picture element. A display screen is composed of an array of pixels. In a black-and-white system, pixels are turned on and off to form images. In a color system, each pixel has three components: red, green, and blue. The intensity of each component can be controlled.

polar coordinates. A coordinate system in which positions are measured as a distance from the origin and an angle from some reference direction (usually, counterclockwise from the x-axis).

polled I/O devices. Devices (keyboard, mouse, button, dials) whose current values are read by the user process.

pre-multiplication. Matrix multiplication on the left. If a matrix M is pre-multiplied by a matrix T, the result is TM.

precision. The number of straight line segments used to approximate one segment of a spline.

primitive. A drawing command, such as **arc, line, circle, polygon,** or **charstr**. Such commands are called primitives because they are not made up of smaller parts, and because they are they basic pieces out of which more complex scenes can be composed. Also used to describe the figures created by drawing commands.

primitive font. A font in which characters are defined as primitives. Like all other primitives, primitive font characters can be scaled and rotated. See **font, raster font**.

pruning. Eliminating the drawing of parts of the display list because a bounding box test shows that the y are not visible.

queued I/O devices. Devices (keyboard, mouse, button, dials) whose changes are recorded in the event queue.

raster font. A font in which the characters are defined directly by the raster bit map. See **font, primitive font**.

refresh rate. The rate at which the monitor is refreshed. A 60 Hz monitor is redrawn 60 times per second.

relative drawing commands. Commands that draw relative to the current graphics position as opposed to being drawn at absolute locations.

repeat factor. The magnification with which the linestyle pattern is used.

RGB mode. A configuration of the hardware which allows values stored in the frame buffer to be interpreted as packed RGB values. The values found in the frame buffer are passed directly to the red, green, and blue guns of the display monitor. The values are not passed through the color map first. (However, each color is sent individually through the gamma ramp to make a final correction to its intensity.) See **color map mode**.

RGB value. The set of red, green, and blue intensities that compose a color is that color's **RGB value.**

RGBA value. The set of red, green, blue, and alpha intensities that compose a color is that color's **RGBA value.** Alpha values are available only on machines having alpha bitplanes.

right-hand rule. If the right hand is wrapped around the axis of rotation, the fingers curl in the same direction as positive rotation, and the thumb points in the same direction as the axis of rotation.. A right-handed rotation is counter-clockwise.

rotation. The transformation of a primitive by rotating it about an axis. See **transformation.**

scaling. Uniform stretching of a primitive along an axis.

screen coordinates, screen space. The coordinate system that defines the display screen. Distances are measured in units of pixels, and the origin is in the lower left-hand corner. On most systems the screen size is 1024 pixels high by 1280 pixels wide. The viewport defines the mapping from normalized device coordinates to screen coordinates. See **eye coordinates, modeling coordinates, world coordinates, transformation**.

screenmask. A rectangular area of the screen to which all drawing operations are clipped. It is normally set equal to the viewport and to the window. A screenmask is useful for character clipping.

selecting. A method for finding what primitives are being drawn in a given volume in 3-D space. See **picking, picking region, name stack, selecting region.**

selecting region. A rhomboid-shaped volume in world coordinates that is sensitive to selecting events. If a drawing primitive draws within this region, a select event is reported. See **hit, picking region, selecting, transformation.**

single buffer mode. A mode in which the frame buffer bitplanes are organized into a single large frame buffer. This frame buffer is the one currently displayed and is also the one in which all drawing occurs. See **double buffer mode.**

swap interval. The amount of elapsed time between frame buffer swaps. The system waits at least the amount of time specified by the **swapinterval** subroutine before honoring a request to exchange the front and back buffers. The swap interval is measured in units of vertical retraces, which occur every 30th of a second on most systems. The swap interval is useful in achieving smooth-flowing animation.

tag. A marker in the display list used as the location for display list editing.

textport. A region on the display screen used to present textual output from graphical or non-graphical programs.

texture. A pattern used to fill rectangles, convex polygons, arcs, and circles.

transformation. A four-by-four matrix that helps determine the location where 3D drawing will occur, the position of the viewpoint (the viewer's eye), and the amount of the scene encompassed and visible. Transformations occur at four points within the graphics pipeline:

- Modeling transformation, which maps modeling coordinates into world coordinates. All drawing primitives specify positions that are presumed to be positions in modeling coordinates. Modeling transformation can be used to move the thing being drawn.
- Viewing transformation, which maps from world coordinates to viewer coordinates. The origin of the viewer coordinate system can be thought of as the location of the viewer's "eye," and viewing transformations can be used to move the eye around in world coordinates.
- Projection transformation, which defines the boundaries of the clipping region. A projection transformation maps viewer coordinates to normalized device coordinates, and the clipping plane boundaries are at $x = \pm w$, $y = \pm w$, $z = \pm w$. Projection transformations are used to define what region of the world is visible on the screen.
- Viewport, or NDC to DC transformation. The viewport transformation is not a full-fledged four-by-four transformation matrix; only three of the diagonal elements in the matrix can be changed. The viewport determines the mapping from normalized device coordinates to screen (device) coordinates. By default viewports are the same size as the window, although this can be adjusted.

translation. The moving of a display image in a straight line from one location to another. See **transformation.**

trimming loops. A set of oriented closed curves used to set the boundaries of a NURBS surface. See **NURBS**.

twist. A rotation around the line of sight.

unit cube. A volume defined by the following planes: x = –1, x = 1, y = –1, y = 1, z = –1, z = 1. See **normalized device coordinates**.

valuator. An input/output device that returns a value in a range. For example, a mouse is logically two valuators: the x position and the y position.

vector product. Another term for the vector cross product. If a=(a1, a2, a3) and b=(b1, b2, b3) are two 3D vectors, the vector product a x b = (a2b3–b2a3, a3b1–b3a1, a1b2–b1a2).

vertical retrace. The rate at which the monitor is refreshed. A 60 Hz monitor is redrawn 60 times per second. Same as **refresh rate**.

vertical retrace period. The amount of elapsed time between retraces of the screen. All video monitors use an electron beam to sweep the phosphors at the face of the monitor. Because the phosphors glow for only a brief period of time, the entire screen must be reswept periodically by the electron beam. On most monitors, this is done 30 times per second (30 Hz). Thus, the vertical retrace period is 1/30 second.

viewing matrix. A matrix used to describe the location of the viewer (the virtual eye looking upon a scene) in relation to the world. See **transformation, world coordinates**.

viewport. The mapping from normalized device coordinates to device coordinates. The viewport maps the unit cube x/w = ±1, y/w = ±1, z/w = ±1 to the screen space, as measured in pixels. The viewport is the last transformation in the graphics pipeline. The viewport can be smaller or larger than the window, smaller or larger than the screenmask, although in most applications, it is the same size.

window. An AIXwindow. A rectangular area of the screen that can be moved about or placed on top of or pulled under other windows, or iconized by the user. All drawing inside the window is done by the GL process that created that window, and is totally under the control of that process. The drawing of the window borders, however, together with the window placement/iconization, is under the control of the window manager; for example, the AIXwindows Window Manager.

Note that for most simple GL programs, the viewport and screenmask are set to the same size as the window. Do not confuse an AIXwindow with the GL window subroutine, which defines a frustum in world space.

wire frame. A graphics surface-drawing technique in which the edges and contours of a primitive are represented by simple lines.

world coordinates, world space. The user-defined coordinate system in which an image is described. Modeling commands are used to position primitives in world space. Viewing and projection transformations define the mapping of the world space to screen space. See **modeling coordinates, screen space, transformations**.

writemask. A set of 8 or 12 bits (depending on the frame buffer configuration), one bit for each bitplane of the frame buffer. During any drawing operation, only those planes enabled by a 1 (one) in the bit mask can be altered. Planes set to 0 (zero) are marked read only.

z-buffer, z-buffering. Applies both to the device and the techniques commonly used as an aid in removing hidden lines and hidden surfaces. If z-buffering is enabled, each pixel will store a depth value as well as a color value. In simple terms, the depth can be thought of as the distance from the viewer's eye to the pixel. Whenever a drawing routine tries to update a pixel, it will first check the current pixel's "depth" or "z-value" and will only update that pixel with new values if the new pixel is closer than the current pixel. The region of memory that stores the z-values is also referred to as the z-buffer.

zoom factor. a multiplier to determine the amount of enlargement of a specified screen rectangle. The *x* zoom factor determines the enlargement in the *x* direction; the *y* zoom factor, in the *y* direction.

Index

F

G

I

K

L

M

N

O

P

Q

R

S

Z

Reader's Comment Form

AIX Calls and Subroutines Reference for IBM RISC System/6000

SC23-2198-00

Please use this form only to identify publication errors or to request changes in publications. Your comments assist us in improving our publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

☐ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page	Comments

Please contact your IBM representative or your IBM-approved remarketer to request additional publications.

Please print

Date _____

Your Name _____

Company Name _____

Mailing Address _____

_____

_____

Phone No. ( ) _____
Area Code

No postage necessary if mailed in the U.S.A

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 997
11400 Burnet Rd.
Austin, Texas 78758–3493

Fold

Fold

Cut or Fold Along Line

Fold and Tape

Please Do Not Staple

Fold and Tape

International Business Machines
Corporation
11400 Burnet Road
Austin, Texas 78758–3493

Printed in the
United States of America

SC23-2198-00