NAME

glut - an introduction to the OpenGL Utility Toolkit

SYNOPSIS

#include <GL/glut.h>

DESCRIPTION

The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSI C and FORTRAN bindings for writing window system independent OpenGL programs. The toolkit supports the following functionality:

Multiple windows for OpenGL rendering.

Callback driven event processing.

Various input devices including keyboard, mouse, tablet, dial and button box, joystick, and Spaceball.

An ‘‘idle’’ routine and timers.

A simple, cascading pop-up menu facility.

Utility routines to generate various solid and wire frame objects.

Support for bitmap and stroke fonts.

Miscellaneous window management functions, including managing overlays, dynamic video resizing, and game-oriented full screen support.

An ANSI C implementation of GLUT for both the X Window System (X11) and Windows 95, 98, and NT (Win32) has been implemented by the author. Language bindings for FORTRAN and Ada are available. C++ programs can call GLUT directly through the C interface. MacOS and OS/2 implementations of GLUT are also available.

BACKGROUND

One of the major accomplishments in the specification of OpenGL was the isolation of window system dependencies from OpenGL’s rendering model. The result is that OpenGL is window system independent.

Window system operations such as the creation of a rendering window and the handling of window system events are left to the native window system to define. Necessary interactions between OpenGL and the window system such as creating and binding an OpenGL context to a window are described separately from the OpenGL specification in a window system dependent specification. For example, the GLX specification describes the standard by which OpenGL interacts with the X Window System. Likewise, the WGL interface defines how OpenGL interacts with Microsoft’s Windows operating system.

The predecessor to OpenGL is IRIS GL. Unlike OpenGL, IRIS GL does specify how rendering windows are created and manipulated. IRIS GL’s windowing interface is reasonably popular largely because it is simple to use. IRIS GL programmers can worry about graphics programming without needing to be an expert in programming the native window system. Experience also demonstrated that IRIS GL’s windowing interface was high-level enough that it could be retargeted to different window systems. Silicon Graphics migrated from NeWS to the X Window System without any major changes to IRIS GL’s basic windowing interface.

Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted to various systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs.

Unfortunately, the lack of a window system interface for OpenGL is a gap in OpenGL’s utility. Learning native window system APIs such as the X Window System’s Xlib or Motif can be daunting. Even those familiar with native window system APIs need to understand the interface that binds OpenGL to the native window system. And when an OpenGL program is written using the native window system interface, despite the portability of the program’s OpenGL rendering code, the program itself will be window system

dependent.

Testing and documenting OpenGL's functionality lead to the development of the tk and aux toolkits. The aux toolkit is used in the examples found in the OpenGL Programming Guide. Unfortunately, aux has numerous limitations and its utility is largely limited to toy programs. The tk library has more functionality than aux but was developed in an ad hoc fashion and still lacks much important functionality that IRIS GL programmers expect, like pop-up menus and overlays.

GLUT is designed to fill the need for a window system independent programming interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Features from the IRIS GL, aux, and tk interfaces are included to make it easy for programmers used to these interfaces to develop programs for GLUT.

PHILOSPHY

GLUT simplifies the implementation of programs using OpenGL rendering. The GLUT application programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT API (like the OpenGL API) is stateful. Most initial GLUT state is defined and the initial state is reasonable for simple programs.

The GLUT routines also take relatively few parameters. No pointers are returned. The only pointers passed into GLUT are pointers to character strings (all strings passed to GLUT are copied, not referenced) and opaque font handles.

The GLUT API is (as much as reasonable) window system independent. For this reason, GLUT does not return any native window system handles, pointers, or other data structures. More subtle window system dependencies such as reliance on window system dependent fonts are avoided by GLUT; instead, GLUT supplies its own (limited) set of fonts.

For programming ease, GLUT provides a simple menu sub-API. While the menuing support is designed to be implemented as pop-up menus, GLUT gives window system leeway to support the menu functionality in another manner (pull-down menus for example).

Two of the most important pieces of GLUT state are the current window and current menu. Most window and menu routines affect the current window or menu respectively. Most callbacks implicitly set the current window and menu to the appropriate window or menu responsible for the callback. GLUT is designed so that a program with only a single window and/or menu will not need to keep track of any window or menu identifiers. This greatly simplifies very simple GLUT programs.

GLUT is designed for simple to moderately complex programs focused on OpenGL rendering. GLUT implements its own event loop. For this reason, mixing GLUT with other APIs that demand their own event handling structure may be difficult. The advantage of a builtin event dispatch loop is simplicity.

GLUT contains routines for rendering fonts and geometric objects, however GLUT makes no claims on the OpenGL display list name space. For this reason, none of the GLUT rendering routines use OpenGL display lists. It is up to the GLUT programmer to compile the output from GLUT rendering routines into display lists if this is desired.

GLUT routines are logically organized into several sub-APIs according to their functionality. The sub-APIs are:

Initialization.

Command line processing, window system initialization, and initial window creation state are controlled by these routines.

Beginning Event Processing.
: This routine enters GLUT's event processing loop. This routine never returns, and it continuously calls GLUT callbacks as necessary.

Window Management.
: These routines create and control windows.

Overlay Management.
: These routines establish and manage overlays for windows.

Menu Management.
: These routines create and control pop-up menus.

Callback Registration.
: These routines register callbacks to be called by the GLUT event processing loop.

Color Index Colormap Management.
: These routines allow the manipulation of color index colormaps for windows.

Game-oriented Full Screen Mode.
: These routines allow OpenGL to be used in a full screen mode at various display resolutions and depths.

State Retrieval.
: These routines allows programs to retrieve state from GLUT.

Font Rendering.
: These routines allow rendering of stroke and bitmap fonts.

Geometric Shape Rendering.
: These routines allow the rendering of 3D geometric objects including spheres, cones, icosahedrons, and teapots.

CONVENTIONS

GLUT window and screen coordinates are expressed in pixels. The upper left hand corner of the screen or a window is (0,0). X coordinates increase in a rightward direction; Y coordinates increase in a downward direction. Note: This is inconsistent with OpenGL's coordinate scheme that generally considers the lower left hand coordinate of a window to be at (0,0) but is consistent with most popular window systems.

Integer identifiers in GLUT begin with one, not zero. So window identifiers, menu identifiers, and menu item indexes are based from one, not zero.

In GLUT's ANSI C binding, for most routines, basic types (int, char*) are used as parameters. In routines where the parameters are directly passed to OpenGL routines, OpenGL types (GLfloat) are used.

The header files for GLUT should be included in GLUT programs with the following include directive:

#include <GL/glut.h>

Because a very large window system software vendor (who will remain nameless) has an apparent inability to appreciate that OpenGL's API is independent of their window system API, portable ANSI C GLUT programs should not directly include <GL/gl.h> or <GL/glu.h>. Instead, ANSI C GLUT programs should rely on <GL/glut.h> to include the necessary OpenGL and GLU related header files.

The ANSI C GLUT library archive is typically named libglut.a on Unix systems. GLUT programs need to link with the system's OpenGL and GLUT libraries (and any libraries these libraries potentially depend on). A set of window system dependent libraries may also be necessary for linking GLUT programs. For example, programs using the X11 GLUT implementation typically need to link with Xlib, the X extension library, possibly the X Input extension library, the X miscellaneous utilities library, and the math library. An example X11/Unix compile line would look like:

cc -o foo foo.c -lglut -lGLU -lGL -lXmu -lXi -lXext -lX11 -lm

SEE ALSO

glutAddMenuEntry, glutAddSubMenu, glutAttachMenu, glutBitmapCharacter, glutBitmapLength, glutBitmapWidth, glutButtonBoxFunc, glutChangeToMenuEntry, glutChangeToSubMenu, glutCopyColormap, glutCreateMenu, glutCreateSubWindow, glutCreateWindow, glutDestroyMenu, glutDestroyWindow, glutDeviceGet, glutDialsFunc, glutDisplayFunc, glutEnterGameMode, glutEntryFunc, glutEstablishOverlay, glutExtensionSupported, glutForceJoystickFunc, glutFullScreen, glutGameModeGet, glutGameModeString, glutGet, glutGetColor, glutGetModifiers, glutIdleFunc, glutIgnoreKeyRepeat, glutInit, glutInitDisplayMode, glutInitDisplayString, glutInitWindowPosition, glutJoystickFunc, glutKeyboardFunc, glutKeyboardUpFunc, glutLayerGet, glutLeaveGameMode, glutMainLoop, glutMenuStatusFunc, glutMotionFunc, glutMouseFunc, glutOverlayDisplayFunc, glutPopWindow, glutPositionWindow, glutPostOverlayRedisplay, glutPostRedisplay, glutRemoveMenuItem, glutRemoveOverlay, glutReportErrors, glutReshapeFunc, glutReshapeWindow, glutSetColor, glutSetCursor, glutSetMenu, glutSetupVideoResizing, glutSetWindow, glutSetWindowTitle, glutShowOverlay, glutShowWindow, glutSolidCone, glutSolidCube, glutSolidDodecahedron, glutSolidIcosahedron, glutSolidOctahedron, glutSolidSphere, glutSolidTeapot, glutSolidTetrahedron, glutSolidTorus, glutSpaceballButtonFunc, glutSpaceballMotionFunc, glutSpaceballRotateFunc, glutSpecialFunc, glutSpecialUpFunc, glutStopVideoResizing, glutStrokeCharacter, glutStrokeLength, glutStrokeWidth, glutSwapBuffers, glutTabletButtonFunc, glutTabletMotionFunc, glutTimerFunc, glutUseLayer, glutVideoPan, glutVideoResizeGet, glutVideoResize, glutVisibilityFunc, glutWarpPointer, glutWindowStatusFunc

REFERENCES

Mark Kilgard, *Programming OpenGL for the X Window System*, Addison-Wesley, ISBN 0-201-48359-9, 1996. Chapters 4 is a comprehensive tutorial about using GLUT. Chapter 5 explores OpenGL with seven different GLUT examples. Appendix B is a complete functional description of the GLUT API. Appendix C documents GLUT's programmer-visible state.

Mark Kilgard, *The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3* (the official GLUT specification).

Mason Woo, Jackie Neider, Tom Davis, Dave Shriner, *OpenGL 1.2 Programming Guide, Third Edition: The Official Guide to Learning OpenGL, Version 1.2*, Addison-Wesley, ISBN 0-201-60458-2, 1999. The best all-around introduction to OpenGL uses GLUT for its examples.

Ed Angel, *Interactive Computer Graphics: A Top-Down Approach with OpenGL*, Addison-Wesley, ISBN 0-201-13859-7X, 1999. An undergraduate computer graphics textbook that uses GLUT for its programming examples.

WEB REFERENCES

Main GLUT page
http://reality.sgi.com/mjk/glut3/glut3.html

GLUT Frequently Asked Question list
http://reality.sgi.com/mjk/glut3/glut-faq.html

The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3
http://reality.sgi.com/mjk/spec3/spec3.html
http://reality.sgi.com/mjk/glut3/glut-3.spec.ps.gz

OpenGL and X: An OpenGL Toolkit article (PostScript)
http://reality.sgi.com/mjk/glut3/glut.column1.ps.gz

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutAddMenuEntry - adds a menu entry to the bottom of the current menu.

SYNTAX

void glutAddMenuEntry(char *name, int value);

ARGUMENTS

name	ASCII character string to display in the menu entry.
value	Value to return to the menu's callback function if the menu entry is selected.

DESCRIPTION

glutAddMenuEntry adds a menu entry to the bottom of the current menu. The string name will be displayed for the newly added menu entry. If the menu entry is selected by the user, the menu's callback will be called passing value as the callback's parameter.

SEE ALSO

glutAddSubMenu, glutCreateMenu, glutChangeToMenuEntry, glutRemoveMenuItem

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutAddSubMenu - adds a sub-menu trigger to the bottom of the current menu.

SYNTAX

void glutAddSubMenu(char *name, int menu);

ARGUMENTS

name	ASCII character string to display in the menu item from which to cascade the sub-menu.
menu	Identifier of the menu to cascade from this sub-menu menu item.

DESCRIPTION

glutAddSubMenu adds a sub-menu trigger to the bottom of the current menu. The string name will be displayed for the newly added sub-menu trigger. If the sub-menu trigger is entered, the sub-menu numbered menu will be cascaded, allowing sub-menu menu items to be selected.

SEE ALSO

glutAddMenuEntry, glutChangeToSubMenu, glutRemoveItem

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutAttachMenu - attaches a mouse button for the current window to the identifier of the current menu; glutDetachMenu - detaches an attached mouse button from the current window.

SYNTAX

void glutAttachMenu(int button);
void glutDetachMenu(int button);

ARGUMENTS

button The button to attach a menu or detach a menu.

DESCRIPTION

glutAttachMenu attaches a mouse button for the current window to the identifier of the current menu; glutDetachMenu detaches an attached mouse button from the current window. By attaching a menu identifier to a button, the named menu will be popped up when the user presses the specified button. button should be one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, and GLUT_RIGHT_BUTTON. Note that the menu is attached to the button by identifier, not by reference.

SEE ALSO

glutCreateMenu, glutMouseFunc, glutMenuStatusFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutBitmapCharacter - renders a bitmap character using OpenGL.

SYNTAX

void glutBitmapCharacter(void *font, int character);

ARGUMENTS

font	Bitmap font to use.
character	Character to render (not confined to 8 bits).

DESCRIPTION

Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font. The available fonts are:

GLUT_BITMAP_8_BY_13

A fixed width font with every character fitting in an 8 by 13 pixel rectangle. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-misc-fixed-medium-r-normal--13-120-75-75-C-80-iso8859-1

GLUT_BITMAP_9_BY_15

A fixed width font with every character fitting in an 9 by 15 pixel rectangle. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-misc-fixed-medium-r-normal--15-140-75-75-C-90-iso8859-1

GLUT_BITMAP_TIMES_ROMAN_10

A 10-point proportional spaced Times Roman font. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-adobe-times-medium-r-normal--10-100-75-75-p-54-iso8859-1

GLUT_BITMAP_TIMES_ROMAN_24

A 24-point proportional spaced Times Roman font. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-adobe-times-medium-r-normal--24-240-75-75-p-124-iso8859-1

GLUT_BITMAP_HELVETICA_10

A 10-point proportional spaced Helvetica font. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-adobe-helvetica-medium-r-normal--10-100-75-75-p-56-iso8859-1

GLUT_BITMAP_HELVETICA_12

A 12-point proportional spaced Helvetica font. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-adobe-helvetica-medium-r-normal--12-120-75-75-p-67-iso8859-1

GLUT_BITMAP_HELVETICA_18

A 18-point proportional spaced Helvetica font. The exact bitmaps to be used is defined by the standard X glyph bitmaps for the X font named:

-adobe-helvetica-medium-r-normal--18-180-75-75-p-98-iso8859-1

Rendering a nonexistent character has no effect. glutBitmapCharacter automatically sets the OpenGL unpack pixel storage modes it needs appropriately and saves and restores the previous modes before returning. The generated call to glBitmap will adjust the current raster position based on the width of the character.

EXAMPLE

Here is a routine that shows how to render a string of ASCII text with glutBitmapCharacter:

```
void
output(int x, int y, char *string)
{
  int len, i;

  glRasterPos2f(x, y);
  len = (int) strlen(string);
  for (i = 0; i < len; i++) {
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, string[i]);
  }
}
```

SEE ALSO

glutBitmapWidth, glutStrokeCharacter

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutBitmapWidth - returns the width of a bitmap character, glutBitmapLength returns the length of a bitmap font string.

SYNTAX

int glutBitmapWidth(void *font, int character)
int glutBitmapLength(void *font, const unsigned char *string)

ARGUMENTS

font	Bitmap font to use. For valid values, see the glutBitmapCharacter description.
character	Character to return width of (not confined to 8 bits).
string	Text string (8-bit characters), nul terminated.

DESCRIPTION

glutBitmapWidth returns the width in pixels of a bitmap character in a supported bitmap font. While the width of characters in a font may vary (though fixed width fonts do not vary), the maximum height characteristics of a particular font are fixed.

glutBitmapLength returns the length in pixels of a string (8-bit characters). This length is equivalent to summing all the widths returned by glutBitmapWidth for each character in the string.

SEE ALSO

glutBitmapCharacter, glutStrokeWidth

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutButtonBoxFunc - sets the dial & button box button callback for the current window.

SYNTAX

void glutButtonBoxFunc(void (*func)(int button, int state));

ARGUMENTS

func The new button box callback function.

DESCRIPTION

glutButtonBoxFunc sets the dial & button box button callback for the current window. The dial & button box button callback for a window is called when the window has dial & button box input focus (normally, when the mouse is in the window) and the user generates dial & button box button presses. The button parameter will be the button number (starting at one). The number of available dial & button box buttons can be determined with glutDeviceGet(GLUT_NUM_BUTTON_BOX_BUTTONS). The state is either GLUT_UP or GLUT_DOWN indicating whether the callback was due to a release or press respectively.

Registering a dial & button box button callback when a dial & button box device is not available is ineffectual and not an error. In this case, no dial & button box button callbacks will be generated.

Passing NULL to glutButtonBoxFunc disables the generation of dial & button box button callbacks. When a new window is created, no dial & button box button callback is initially registered.

SEE ALSO

glutDialsFunc, glutDeviceGet, glutSpaceballButtonFunc, glutTabletButtonFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutChangeToMenuEntry - changes the specified menu item in the current menu into a menu entry.

SYNTAX

void glutChangeToMenuEntry(int entry, char *name, int value);

ARGUMENTS

entry	Index into the menu items of the current menu (1 is the topmost menu item).
name	ASCII character string to display in the menu entry.
value	Value to return to the menu's callback function if the menu entry is selected.

DESCRIPTION

glutChangeToMenuEntry changes the specified menu entry in the current menu into a menu entry. The entry parameter determines which menu item should be changed, with one being the topmost item. entry must be between 1 and glutGet(GLUT_MENU_NUM_ITEMS) inclusive. The menu item to change does not have to be a menu entry already. The string name will be displayed for the newly changed menu entry. The value will be returned to the menu's callback if this menu entry is selected.

SEE ALSO

glutChangeToSubMenu, glutAddMenuEntry, glutRemoveMenuItem

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutChangeToSubMenu - changes the specified menu item in the current menu into a sub-menu trigger.

SYNTAX

void glutChangeToSubMenu(int entry, char *name, int menu);

ARGUMENTS

entry	Index into the menu items of the current menu (1 is the topmost menu item).
name	ASCII character string to display in the menu item to cascade the sub-menu from.
menu	Identifier of the menu to cascade from this sub-menu menu item.

DESCRIPTION

glutChangeToSubMenu changes the specified menu item in the current menu into a sub-menu trigger. The entry parameter determines which menu item should be changed, with one being the topmost item. entry must be between 1 and glutGet(GLUT_MENU_NUM_ITEMS) inclusive. The menu item to change does not have to be a sub-menu trigger already. The string name will be displayed for the newly changed sub-menu trigger. The menu identifier names the sub-menu to cascade from the newly added sub-menu trigger.

SEE ALSO

glutChangeToMenuEntry, glutAddSubMenu, glutRemoveMenuItem

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutCopyColormap - copies the logical colormap for the layer in use from a specified window to the current window.

SYNTAX

void glutCopyColormap(int win);

ARGUMENTS

win The identifier of the window to copy the logical colormap from.

DESCRIPTION

glutCopyColormap copies (lazily if possible to promote sharing) the logical colormap from a specified window to the current window's layer in use. The copy will be from the normal plane to the normal plane; or from the overlay to the overlay (never across different layers). Once a colormap has been copied, avoid setting cells in the colormap with glutSetColor since that will force an actual copy of the colormap if it was previously copied by reference. glutCopyColormap should only be called when both the current window and the win window are color index windows.

EXAMPLE

Here is an example of how to create two color index GLUT windows with their colormaps loaded identically and so that the windows are likely to share the same colormap:

```
int win1, win2;

glutInitDisplayMode(GLUT_INDEX);
win1 = glutCreateWindow("first color index win");
glutSetColor(0, 0.0, 0.0, 0.0);  /* black */
glutSetColor(1, 0.5, 0.5, 0.5);  /* gray */
glutSetColor(2, 1.0, 1.0, 1.0);  /* black */
glutSetColor(3, 1.0, 0.0, 0.0);  /* red */
win2 = glutCreateWindow("second color index win");
glutCopyColormap(win1);
```

SEE ALSO

glutSetColor, glutGetColor, glutCreateWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutCreateMenu - creates a new pop-up menu.

SYNTAX

int glutCreateMenu(void (*func)(int value));

ARGUMENTS

func The callback function for the menu that is called when a menu entry from the menu is selected. The value passed to the callback is determined by the value for the selected menu entry.

DESCRIPTION

glutCreateMenu creates a new pop-up menu and returns a unique small integer identifier. The range of allocated identifiers starts at one. The menu identifier range is separate from the window identifier range. Implicitly, the current menu is set to the newly created menu. This menu identifier can be used when calling glutSetMenu.

When the menu callback is called because a menu entry is selected for the menu, the current menu will be implicitly set to the menu with the selected entry before the callback is made.

EXAMPLE

Here is a quick example of how to create a GLUT popup menu with two submenus and attach it to the right button of the current window:

```
int submenu1, submenu2;

submenu1 = glutCreateMenu(selectMessage);
glutAddMenuEntry("abc", 1);
glutAddMenuEntry("ABC", 2);
submenu2 = glutCreateMenu(selectColor);
glutAddMenuEntry("Green", 1);
glutAddMenuEntry("Red", 2);
glutAddMenuEntry("White", 3);
glutCreateMenu(selectFont);
glutAddMenuEntry("9 by 15", 0);
glutAddMenuEntry("Times Roman 10", 1);
glutAddMenuEntry("Times Roman 24", 2);
glutAddSubMenu("Messages", submenu1);
glutAddSubMenu("Color", submenu2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

X IMPLEMENTATION NOTES

If available, GLUT for X will take advantage of overlay planes for implementing pop-up menus. The use of overlay planes can eliminate display callbacks when pop-up menus are deactivated. The SERVER_OVERLAY_VISUALS convention is used to determine if overlay visuals are available.

SEE ALSO

glutCreateWindow, glutDestroyMenu, glutSetMenu, glutAttachMenu

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutCreateSubWindow - creates a subwindow.

SYNTAX

int glutCreateSubWindow(int win,
int x, int y, int width, int height);

ARGUMENTS

win	Identifier of the subwindow's parent window.
x	Window X location in pixels relative to parent window's origin.
y	Window Y location in pixels relative to parent window's origin.
width	Width in pixels.
height	Height in pixels.

DESCRIPTION

glutCreateSubWindow creates a subwindow of the window identified by win of size width and height at location x and y within the current window. Implicitly, the current window is set to the newly created subwindow.

Each created window has a unique associated OpenGL context. State changes to a window's associated OpenGL context can be done immediately after the window is created.

The display state of a window is initially for the window to be shown. But the window's display state is not actually acted upon until glutMainLoop is entered. This means until glutMainLoop is called, rendering to a created window is ineffective. Subwindows can not be iconified.

Subwindows can be nested arbitrarily deep.

The value returned is a unique small integer identifier for the window. The range of allocated identifiers starts at one.

SEE ALSO

glutCreateWindow, glutDestroyWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutCreateWindow - creates a top-level window.

SYNTAX

int glutCreateWindow(char *name);

ARGUMENTS

name ASCII character string for use as window name.

DESCRIPTION

glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

Implicitly, the current window is set to the newly created window.

Each created window has a unique associated OpenGL context. State changes to a window's associated OpenGL context can be done immediately after the window is created.

The display state of a window is initially for the window to be shown. But the window's display state is not actually acted upon until glutMainLoop is entered. This means until glutMainLoop is called, rendering to a created window is ineffective because the window can not yet be displayed.

The value returned is a unique small integer identifier for the window. The range of allocated identifiers starts at one. This window identifier can be used when calling glutSetWindow.

X IMPLEMENTATION NOTES

The proper X Inter-Client Communication Conventions Manual (ICCCM) top-level properties are established. The WM_COMMAND property that lists the command line used to invoke the GLUT program is only established for the first window created.

SEE ALSO

glutCreateSubWindow, glutCreateMenu, glutDestroyWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutDestroyMenu - destroys the specified menu.

SYNTAX

void glutDestroyMenu(int menu);

ARGUMENTS

menu The identifier of the menu to destroy.

DESCRIPTION

glutDestroyMenu destroys the specified menu by menu. If menu was the current menu, the current menu becomes invalid and glutGetMenu will return zero.

SEE ALSO

glutCreateMenu, glutDestroyWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutDestroyWindow - destroys the specified window.

SYNTAX

void glutDestroyWindow(int win);

ARGUMENTS

win Identifier of GLUT window to destroy.

DESCRIPTION

glutDestroyWindow destroys the window specified by win and the window's associated OpenGL context, logical colormap (if the window is color index), and overlay and related state (if an overlay has been established). Any subwindows of destroyed windows are also destroyed by glutDestroyWindow. If win was the current window, the current window becomes invalid (glutGetWindow will return zero).

SEE ALSO

glutCreateWindow, glutCreateSubWindow, glutDestroyMenu

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutDeviceGet - retrieves GLUT device information represented by integers.

SYNTAX

int glutDeviceGet(GLenum info);

ARGUMENTS

info Name of device information to retrieve.

GLUT_HAS_KEYBOARD
Non-zero if a keyboard is available; zero if not available. For most GLUT implementations, a keyboard can be assumed.

GLUT_HAS_MOUSE
Non-zero if a mouse is available; zero if not available. For most GLUT implementations, a keyboard can be assumed.

GLUT_HAS_SPACEBALL
Non-zero if a Spaceball is available; zero if not available.

GLUT_HAS_DIAL_AND_BUTTON_BOX
Non-zero if a dial & button box is available; zero if not available.

GLUT_HAS_TABLET
Non-zero if a tablet is available; zero if not available.

GLUT_NUM_MOUSE_BUTTONS
Number of buttons supported by the mouse. If no mouse is supported, zero is returned.

GLUT_NUM_SPACEBALL_BUTTONS
Number of buttons supported by the Spaceball. If no Spaceball is supported, zero is returned.

GLUT_NUM_BUTTON_BOX_BUTTONS
Number of buttons supported by the dial & button box device. If no dials & button box device is supported, zero is returned.

GLUT_NUM_DIALS
Number of dials supported by the dial & button box device. If no dials & button box device is supported, zero is returned.

GLUT_NUM_TABLET_BUTTONS
Number of buttons supported by the tablet. If no tablet is supported, zero is returned.

GLUT_DEVICE_IGNORE_KEY_REPEAT
Returns true if the current window's auto repeated keys are ignored. This state is controlled by glutIgnoreKeyRepeat.

GLUT_DEVICE_KEY_REPEAT
The window system's global key repeat state. Returns either GLUT_KEY_REPEAT_OFF, GLUT_KEY_REPEAT_ON, or GLUT_KEY_REPEAT_DEFAULT. This will not necessarily return the value last passed to glutSetKeyRepeat.

GLUT_JOYSTICK_POLL_RATE
Returns the current window's joystick poll rate as set by glutJoystickFunc. If no joystick is supported, the poll rate will always be zero. The joystick poll rate also returns zero if the poll rate last specified to glutJoystickFunc is negative or a NULL callback was registered.

GLUT_HAS_JOYSTICK
Non-zero if a joystick is available; zero if not available.

GLUT_JOYSTICK_BUTTONS
Number of buttons supported by the joystick. If no joystick is supported, zero is returned.

GLUT_JOYSTICK_AXES
Number of axes supported by the joystick. If no joystick is supposrted, zero is returned.

DESCRIPTION

glutDeviceGet retrieves GLUT device information represented by integers. The info parameter determines what type of device information to return. Requesting device information for an invalid GLUT device information name returns negative one.

X IMPLEMENTATION NOTES

The current implementation uses to X Input extension to regonize SGI's Spaceball, tablet, and dial and button box devices.

WIN32 IMPLEMENTATION NOTES

The GLUT_DEVICE_KEY_REPEAT alwasy returns GLUT_KEY_REPEAT_ON.

SEE ALSO

glutGet, glutKeyboardFunc, glutMouseFunc, glutSpaceballMotion, glutTabletMotionFunc, glutTabletButtonFunc, glutDialsFunc, glutButtonBoxFunc, glutIgnoreKeyRepeat, glutSetKeyRepeat, glutJoystickFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutDialsFunc - sets the dial & button box dials callback for the current window.

SYNTAX

void glutDialsFunc(void (*func)(int dial, int value));

ARGUMENTS

func The new dials callback function.

DESCRIPTION

glutDialsFunc sets the dial & button box dials callback for the current window. The dial & button box dials callback for a window is called when the window has dial & button box input focus (normally, when the mouse is in the window) and the user generates dial & button box dial changes. The dial parameter will be the dial number (starting at one). The number of available dial & button box dials can be determined with glutDeviceGet(GLUT_NUM_DIALS). The value measures the absolute rotation in degrees. Dial values do not ‘‘roll over’’ with each complete rotation but continue to accumulate degrees (until the int dial value overflows).

Registering a dial & button box dials callback when a dial & button box device is not available is ineffectual and not an error. In this case, no dial & button box dials callbacks will be generated.

Passing NULL to glutDialsFunc disables the generation of dial & button box dials callbacks. When a new window is created, no dial & button box dials callback is initially registered.

SEE ALSO

glutButtonBoxFunc, glutDeviceGet

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutDisplayFunc - sets the display callback for the current window.

SYNTAX

void glutDisplayFunc(void (*func)(void));

ARGUMENTS

func The new display callback function.

DESCRIPTION

glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and (if no overlay display callback is registered) the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state).

GLUT determines when the display callback should be triggered based on the window's redisplay state. The redisplay state for a window can be either set explicitly by calling glutPostRedisplay or implicitly as the result of window damage reported by the window system. Multiple posted redisplays for a window are coalesced by GLUT to minimize the number of display callbacks called.

When an overlay is established for a window, but there is no overlay display callback registered, the display callback is used for redisplaying both the overlay and normal plane (that is, it will be called if either the redisplay state or overlay redisplay state is set). In this case, the layer in use is not implicitly changed on entry to the display callback.

See glutOverlayDisplayFunc to understand how distinct callbacks for the overlay and normal plane of a window may be established.

When a window is created, no display callback exists for the window. It is the responsibility of the programmer to install a display callback for the window before the window is shown. A display callback must be registered for any window that is shown. If a window becomes displayed without a display callback being registered, a fatal error occurs. Passing NULL to glutDisplayFunc is illegal as of GLUT 3.0; there is no way to ''deregister'' a display callback (though another callback routine can always be registered).

Upon return from the display callback, the normal damaged state of the window (returned by calling glutLayerGet(GLUT_NORMAL_DAMAGED) is cleared. If there is no overlay display callback registered the overlay damaged state of the window (returned by calling glutLayerGet(GLUT_OVERLAY_DAMAGED) is also cleared.

SEE ALSO

glutCreateMenu, glutPostRedisplay, glutOverlayDisplayFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutEnterGameMode, glutLeaveGameMode - enters and leaves GLUT's game mode.

SYNTAX

```
void glutEnterGameMode(void);
void glutLeaveGameMode(void);
```

DESCRIPTION

glutEnterGameMode is designed to enable high-performance fullscreen GLUT rendering, possibly at a different screen display format. Calling glutEnterGameMode creates a special fullscreen GLUT window (with its own callbacks and OpenGL rendering context state). If the game mode string describes a possible screen display format, GLUT also changes the screen display format to the one described by the game mode string. glutLeaveGameMode leaves the GLUT game mode and returns the screen display format to its default format.

When game mode is entered, certain GLUT functionality is disable to facilitate high-performance fullscreen rendering. GLUT pop-up menus are not available while in game mode. Other created windows and subwindows are not displayed in GLUT game mode. Game mode will also hide all other applications running on the computer's display screen. The intent of these restrictions is to eliminate window clipping issues, permit screen display format changes, and permit fullscreen rendering optimization such as page flipping for fullscreen buffer swaps.

After leaving game mode, the GLUT functionality disabled in game mode is available again. The game mode window (and its OpenGL rendering state) is destroyed when leaving game mode. Any windows and subwindows created before entering the game mode are displayed in their previous locations. The OpenGL state of normal GLUT windows and subwindows is not disturbed by entering and/or leaving game mode.

The following GLUT routines are ignored in game mode: glutFullScreen, glutSetWindowTitle, glutSetIconTitle, glutPositionWindow, glutReshapeWindow, glutPopWindow, glutPushWindow, glutIconifyWindow, glutShowWindow, glutHideWindow.

glutEnterGameMode can be called when already in game mode. This will destroy the previous game mode window (including any OpenGL rendering state) and create a new game mode window with a new OpenGL rendering context. Also if glutEnterGameMode is called when already in game mode and if the game mode string has changed and describes a possible screen display format, the new screen display format takes effect. A reshape callback is generated if the game mode window changes size due to a screen display format change.

Re-entering game mode provides a mechanism for changing the screen display format while already in game mode. Note though that the game mode window's OpenGL state is lost in this process and the application is responsible for re-initializing the newly created game mode window OpenGL state when re-entering game mode.

Game mode cannot be entered while pop-up menus are in use.

Note that the glutEnterGameMode and glutFullScreen routines operate differently. glutFullScreen simply makes the current window match the size of the screen. glutFullScreen does not change the screen display format and does not disable any GLUT features such as pop-up menus; glutFullScreen continues to operate in a "windowed" mode of operation. glutEnterGameMode creates a new window style, possibly changes the screen display mode, limits GLUT functionality, and hides other applications.

SEE ALSO

glutGameModeGet, glutGameModeString, glutInitDisplayString

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutEntryFunc - sets the mouse enter/leave callback for the current window.

SYNTAX

void glutEntryFunc(void (*func)(int state));

ARGUMENTS

func The new entry callback function.

DESCRIPTION

glutEntryFunc sets the mouse enter/leave callback for the current window. The state callback parameter is either GLUT_LEFT or GLUT_ENTERED depending on if the mouse pointer has last left or entered the window.

Passing NULL to glutEntryFunc disables the generation of the mouse enter/leave callback.

Some window systems may not generate accurate enter/leave callbacks. X IMPLEMENTATION NOTES An X implementation of GLUT should generate accurate enter/leave callbacks.

SEE ALSO

glutMotionFunc, glutCreateWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutEstablishOverlay - establishes an overlay (if possible) for the current window.

SYNTAX

void glutEstablishOverlay(void);

DESCRIPTION

glutEstablishOverlay establishes an overlay (if possible) for the current window. The requested display mode for the overlay is determined by the initial display mode. glutLayerGet(GLUT_OVERLAY_POSSIBLE) can be called to determine if an overlay is possible for the current window with the current initial display mode. Do not attempt to establish an overlay when one is not possible; GLUT will terminate the program.

If glutEstablishOverlay is called when an overlay already exists, the existing overlay is first removed, and then a new overlay is established. The state of the old overlay's OpenGL context is discarded.

The initial display state of an overlay is shown, however the overlay is only actually shown if the overlay's window is shown.

Implicitly, the window's layer in use changes to the overlay immediately after the overlay is established.

EXAMPLE

Establishing an overlay is a bit involved, but easy once you get the hang of it. Here is an example:

```
int overlaySupport;
int transparent, red, white;

glutInitDisplayMode(GLUT_SINGLE | GLUT_INDEX);
overlaySupport = glutLayerGet(GLUT_OVERLAY_POSSIBLE);
if (overlaySupport) {
  glutEstablishOverlay();
  glutHideOverlay();
  transparent = glutLayerGet(GLUT_TRANSPARENT_INDEX);
  glClearIndex(transparent);
  red = (transparent + 1) % glutGet(GLUT_WINDOW_COLORMAP_SIZE);
  white = (transparent + 2) % glutGet(GLUT_WINDOW_COLORMAP_SIZE);
  glutSetColor(red, 1.0, 0.0, 0.0);  /* Red. */
  glutSetColor(white, 1.0, 1.0, 1.0);  /* White. */
  glutOverlayDisplayFunc(redrawOverlay);
  glutReshapeFunc(reshape);
} else {
  printf("Sorry, no nifty overlay (try an SGI workstation)!0);
}
```

If you setup an overlay and you install a reshape callback, you need to update the viewports and possibly projection matrices of both the normal plane and the overlay. For example, your reshape callback might look like this:

```
void
reshape(int w, int h)
{
  if (overlaySupport) {
    glutUseLayer(GLUT_OVERLAY);
    /* Setup overlay to have X style coordinate system. */
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, w, 0, h);
```

```
    glScalef(1, -1, 1);
    glTranslatef(0, -h, 0);
    glMatrixMode(GL_MODELVIEW);
    glutUseLayer(GLUT_NORMAL);
  }
  glViewport(0, 0, w, h);
}
```

See the glutOverlayDisplayFunc man page for an example showing one way to write your overlay display callback.

X IMPLEMENTATION NOTES

GLUT for X uses the SERVER_OVERLAY_VISUALS convention is used to determine if overlay visuals are available. While the convention allows for opaque overlays (no transparency) and overlays with the transparency specified as a bitmask, GLUT overlay management only provides access to transparent pixel overlays.

Until RGBA overlays are better understood, GLUT only supports color index overlays.

SEE ALSO

glutUseLayer, glutRemoveLayer, glutCreateWindow, glutPostOverlayRedisplay, glutShowOverlay, glutOverlayDisplayFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutExtensionSupported - helps to easily determine whether a given OpenGL extension is supported.

SYNTAX

int glutExtensionSupported(char *extension);

ARGUMENTS

extension Name of OpenGL extension.

DESCRIPTION

glutExtensionSupported helps to easily determine whether a given OpenGL extension is supported or not. The extension parameter names the extension to query. The supported extensions can also be determined with glGetString(GL_EXTENSIONS), but glutExtensionSupported does the correct parsing of the returned string.

glutExtensionSupported returns non-zero if the extension is supported, zero if not supported.

There must be a valid current window to call glutExtensionSupported.

glutExtensionSupported only returns information about OpenGL extensions only. This means window system dependent extensions (for example, GLX extensions) are not reported by glutExtensionSupported.

EXAMPLE

Here is an example of using glutExtensionSupported:

```
if (!glutExtensionSupported("GL_EXT_texture")) {
  fprintf(stderr, "Missing the texture extension!\n");
  exit(1);
}
```

Notice that the name argument includes both the GL prefix and the extension family prefix (EXT).

SEE ALSO

glutGet, glGetString

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutForceJoystickFunc - forces current window's joystick callback to be called.

SYNTAX

void glutForceJoystickFunc(void);

DESCRIPTION

glutForceJoystickFunc forces the current window's joystick callback to be called, reporting the latest joystick state.

The joystick callback is called either due to polling of the joystick at the uniform timer interval set by glutJoystickFunc's pollInterval (specified in milliseconds) or in response to calling glutForceJoystickFunc. If the pollInterval is non-positive, no joystick polling is performed and the GLUT application must frequently (usually from an idle callback) call glutForceJoystickFunc.

The joystick callback will be called once (if one exists) for each time glutForceJoystickFunc is called. The callback is called from glutJoystickFunc. That is, when glutJoystickFunc returns, the callback will have already happened.

GLUT IMPLEMENTATION NOTES FOR X11

The current implementation of GLUT for X11 supports the joystick API, but not actual joystick input. A future implementation of GLUT for X11 may add joystick support.

GLUT IMPLEMENTATION NOTES FOR WIN32

The current implementation of GLUT for Win32 supports the joystick API and joystick input, but does so through the dated joySetCapture and joyGetPosEx Win32 Multimedia API. The current GLUT joystick support for Win32 has all the limitations of the Win32 Multimedia API joystick support. A future implementation of GLUT for Win32 may use DirectInput.

SEE ALSO

glutJoystickFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutFullScreen - requests that the current window be made full screen.

SYNTAX

void glutFullScreen(void);

DESCRIPTION

glutFullScreen requests that the current window be made full screen. The exact semantics of what full screen means may vary by window system. The intent is to make the window as large as possible and disable any window decorations or borders added the window system. The window width and height are not guaranteed to be the same as the screen width and height, but that is the intent of making a window full screen.

glutFullScreen is defined to work only on top-level windows.

The glutFullScreen requests are not processed immediately. The request is executed after returning to the main event loop. This allows multiple glutReshapeWindow, glutPositionWindow, and glutFullScreen requests to the same window to be coalesced.

Subsequent glutReshapeWindow and glutPositionWindow requests on the window will disable the full screen status of the window.

X IMPLEMENTATION NOTES

In the X implementation of GLUT, full screen is implemented by sizing and positioning the window to cover the entire screen and posting the _MOTIF_WM_HINTS property on the window requesting absolutely no decorations. Non-Motif window managers may not respond to _MOTIF_WM_HINTS.

SEE ALSO

glutReshapeWindow, glutPositionWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutGameModeGet - retrieves GLUT game mode information represented by integers.

SYNTAX

int glutGameModeGet(GLenum info);

ARGUMENTS

info Name of game mode information to retrieve.

GLUT_GAME_MODE_ACTIVE
Non-zero if GLUT's game mode is active; zero if not active. Game mode is not active initially. Game mode becomes active when glutEnterGameMode is called. Game mode becomes inactive when glutLeaveGameMode is called.

GLUT_GAME_MODE_POSSIBLE
Non-zero if the game mode string last specified to glutGameModeString is a possible game mode configuration; zero otherwise. Being "possible" does not guarantee that if game mode is entered with glutEnterGameMode that the display settings will actually changed. GLUT_GAME_MODE_DISPLAY_CHANGED should be called once game mode is entered to determine if the display mode is actually changed.

GLUT_GAME_MODE_WIDTH
Width in pixels of the screen when game mode is activated.

GLUT_GAME_MODE_HEIGHT
Height in pixels of the screen when game mode is activated.

GLUT_GAME_MODE_PIXEL_DEPTH
Pixel depth of the screen when game mode is activiated.

GLUT_GAME_MODE_REFRESH_RATE
Screen refresh rate in cyles per second (hertz) when game mode is activated. Zero is returned if the refresh rate is unknown or cannot be queried.

GLUT_GAME_MODE_DISPLAY_CHANGED
Non-zero if entering game mode actually changed the display settings. If the game mode string is not possible or the display mode could not be changed for any other reason, zero is returned.

DESCRIPTION

glutGameModeGet retrieves GLUT game mode information represented by integers. The info parameter determines what type of game mode information to return. Requesting game mode information for an invalid GLUT game mode information name returns negative one.

SEE ALSO

glutGet, glutDeviceGet, glutLayerGet, glutGameModeString, glutEnterGameMode, glutLeaveGameMode

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutGameModeString - sets the game mode configuration via a string.

SYNTAX

void glutGameModeString(const char *string);

ARGUMENTS

string ASCII string for selecting a game mode configuration.

DESCRIPTION

glutGameModeString sets the game mode configuration via an ASCII string. The game mode configuration string for GLUT's fullscreen game mode describes the suitable screen width and height in pixels, the pixel depth in bits, and the video refresh frequency in hertz. The game mode configuration string can also specify a window system dependent display mode.

The string is a list of zero or more capability descriptions seperated by spaces and tabs. Each capability description is a capability name that is followed by a comparator and a numeric value. (Unlike the display mode string specified using glutInitDisplayString, the comparator and numeric value are *not* optional.) For example, "width>=640" and "bpp=32" are both valid criteria.

The capability descriptions are translated into a set of criteria used to select the appropriate game mode configuration.

The criteria are matched in strict left to right order of precdence. That is, the first specified criteria (leftmost) takes precedence over the later criteria for non-exact criteria (greater than, less than, etc. comparators). Exact criteria (equal, not equal compartors) must match exactly so precedence is not relevant.

The numeric value is an integer that is parsed according to ANSI C's strtol(str, strptr, 0) behavior. This means that decimal, octal (leading 0), and hexidecimal values (leading 0x) are accepeted.

The valid compartors are:

= Equal.

!= Not equal.

< Less than and preferring larger difference (the least is best).

> Greater than and preferring larger differences (the most is best).

<= Less than or equal and preferring larger difference (the least is best).

>= Greater than or equal and preferring more instead of less. This comparator is useful for allocating resources like color precsion or depth buffer precision where the maximum precison is generally preferred. Contrast with the tilde (~) comprator.

~ Greater than or equal but preferring less instead of more. This compartor is useful for allocating resources such as stencil bits or auxillary color buffers where you would rather not over allocate.

The valid capability names are:

bpp Bits per pixel for the frame buffer.

height Height of the screen in pixels.

hertz Video refresh rate of the screen in hertz.

num Number of the window system depenedent display mode configuration.

width Width of the screen in pixels.

An additional compact screen resolution description format is supported. This compact description convienently encodes the screen resolution description in a single phrase. For example, "640x480:16@60" requests a 640 by 480 pixel screen with 16 bits per pixel at a 60 hertz video refresh rate. A compact screen

resolution description can be mixed with conventional capability descriptions.

The compact screen resolution description format is as follows:

[*width* "x" *height*][":" *bitsPerPixel*]["@" *videoRate*]

Unspecifed capability descriptions will result in unspecified criteria being generated. These unspecified criteria help glutGameModeString behave sensibly with terse game mode description strings.

SEE ALSO

glutGameModeGet, glutEnterGameMode, glutLeaveGameMode, glutInitDisplayString

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutGet - retrieves simple GLUT state represented by integers.

SYNTAX

int glutGet(GLenum state);

ARGUMENTS

state Name of state to retrieve.

GLUT_WINDOW_X
X location in pixels (relative to the screen origin) of the current window.

GLUT_WINDOW_Y
Y location in pixels (relative to the screen origin) of the current window.

GLUT_WINDOW_WIDTH
Width in pixels of the current window.

GLUT_WINDOW_HEIGHT
Height in pixels of the current window.

GLUT_WINDOW_BUFFER_SIZE
Total number of bits for current layer of current window's color buffer. For an RGBA window, this is the sum of GLUT_WINDOW_RED_SIZE, GLUT_WINDOW_GREEN_SIZE, GLUT_WINDOW_BLUE_SIZE, and GLUT_WINDOW_ALPHA_SIZE. For color index windows, this is the size of the color indexes.

GLUT_WINDOW_STENCIL_SIZE
Number of bits in the current layer of current window's stencil buffer.

GLUT_WINDOW_DEPTH_SIZE
Number of bits in the current layer of current window's depth buffer.

GLUT_WINDOW_RED_SIZE
Number of bits of red stored the current layer of current window's color buffer. Zero if the current layer of the current window is color index.

GLUT_WINDOW_GREEN_SIZE
Number of bits of green stored the current layer of current window's color buffer. Zero if the current layer of the current window is color index.

GLUT_WINDOW_BLUE_SIZE
Number of bits of blue stored the current layer of current window's color buffer. Zero if the current layer of the current window is color index.

GLUT_WINDOW_ALPHA_SIZE
Number of bits of alpha stored the current layer of current window's color buffer. Zero if the current layer of the current window is color index.

GLUT_WINDOW_ACCUM_RED_SIZE
Number of bits of red stored in the current layer of current window's accumulation buffer. Zero if the current layer of the current window is color index.

GLUT_WINDOW_ACCUM_GREEN_SIZE
Number of bits of green stored in the current layer of current window's accumulation buffer. Zero if the current layer of the current window is color index.

GLUT_WINDOW_ACCUM_BLUE_SIZE
Number of bits of blue stored in the current layer of current window's accumulation buffer. Zero if the current layer of the current window is color index.

GLUT_WINDOW_ACCUM_ALPHA_SIZE
Number of bits of alpha stored in the current layer of current window's accumulation buffer. Zero if the current layer of the current window is color index.

GLUT_WINDOW_DOUBLEBUFFER
One if the current layer of the current window is double buffered, zero otherwise.

GLUT_WINDOW_RGBA
One if the current layer of the current window is RGBA mode, zero otherwise (i.e., color index).

GLUT_WINDOW_PARENT
The window number of the current window's parent; zero if the window is a top-level window.

GLUT_WINDOW_NUM_CHILDREN
The number of subwindows the current window has (not counting children of children).

GLUT_WINDOW_COLORMAP_SIZE
Size of current layer of current window's color index colormap; zero for RGBA color model layers.

GLUT_WINDOW_NUM_SAMPLES
Number of samples for multisampling for the current layer of the current window.

GLUT_WINDOW_STEREO
One if the current layer of the current window is stereo, zero otherwise.

GLUT_WINDOW_CURSOR
Current cursor for the current window.

GLUT_SCREEN_WIDTH
Width of the screen in pixels. Zero indicates the width is unknown or not available.

GLUT_SCREEN_HEIGHT
Height of the screen in pixels. Zero indicates the height is unknown or not available.

GLUT_SCREEN_WIDTH_MM
Width of the screen in millimeters. Zero indicates the width is unknown or not available.

GLUT_SCREEN_HEIGHT_MM
Height of the screen in millimeters. Zero indicates the height is unknown or not available.

GLUT_MENU_NUM_ITEMS
Number of menu items in the current menu.

GLUT_DISPLAY_MODE_POSSIBLE
Whether the current display mode is supported or not.

GLUT_INIT_DISPLAY_MODE
The initial display mode bit mask.

GLUT_INIT_WINDOW_X
The X value of the initial window position.

GLUT_INIT_WINDOW_Y
The Y value of the initial window position.

GLUT_INIT_WINDOW_WIDTH
The width value of the initial window size.

GLUT_INIT_WINDOW_HEIGHT
The height value of the initial window size.

GLUT_ELAPSED_TIME
Number of milliseconds since glutInit called (or first call to glutGet(GLUT_ELAPSED_TIME)).

GLUT_WINDOW_FORMAT_ID
The window system dependent format ID for the current layer of the current window. On X11 GLUT implementations, this is the X visual ID. On Win32 GLUT implementations, this is the Win32 Pixel Format Descriptor number. This value is returned for debugging, benchmarking, and testing ease.

DESCRIPTION

glutGet retrieves simple GLUT state represented by integers. The state parameter determines what type of state to return. Where appropriate, window capability state is returned for the layer in use. GLUT state names beginning with GLUT_WINDOW_ return state for the current window. GLUT state names beginning with GLUT_MENU_ return state for the current menu. Other GLUT state names return global state. Requesting state for an invalid GLUT state name returns negative one.

SEE ALSO

glutDeviceGet, glutLayerGet, glutGetColor, glutGetWindow, glutGetMenu, glutGetModifiers, glutExtensionSupported

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutGetColor - retrieves a red, green, or blue component for a given color index colormap entry for the layer in use's logical colormap for the current window.

SYNTAX

GLfloat glutGetColor(int cell, int component);

ARGUMENTS

cell	Color cell index (starting at zero).
component	One of GLUT_RED, GLUT_GREEN, or GLUT_BLUE.

DESCRIPTION

glutGetColor retrieves a red, green, or blue component for a given color index colormap entry for the current window's logical colormap. The current window should be a color index window. cell should be zero or greater and less than the total number of colormap entries for the window. For valid color indices, the value returned is a floating point value between 0.0 and 1.0 inclusive. glutGetColor will return -1.0 if the color index specified is an overlay's transparent index, less than zero, or greater or equal to the value returned by glutGet(GLUT_WINDOW_COLORMAP_SIZE), that is if the color index is transparent or outside the valid range of color indices.

SEE ALSO

glutGet, glutSetColor, glutCopyColormap

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutGetModifiers - returns the modifier key state when certain callbacks were generated.

SYNTAX

int glutGetModifiers(void);

DESCRIPTION

glutGetModifiers returns the modifier key state at the time the input event for a keyboard, special, or mouse callback is generated. This routine may only be called while a keyboard, special, or mouse callback is being handled. The window system is permitted to intercept window system defined modifier key strokes or mouse buttons, in which case, no GLUT callback will be generated. This interception will be independent of use of glutGetModifiers.

The bitmask components of the returned integer value are:

GLUT_ACTIVE_SHIFT
Set if the Shift modifier or Caps Lock is active.

GLUT_ACTIVE_CTRL
Set if the Ctrl modifier is active.

GLUT_ACTIVE_ALT
Set if the Alt modifier is active.

SEE ALSO

glutSpecialFunc, glutKeyboardFunc, glutMouseFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutIdleFunc - sets the global idle callback.

SYNTAX

void glutIdleFunc(void (*func)(void));

ARGUMENTS

func The new idle callback function.

DESCRIPTION

glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received. If enabled, the idle callback is continuously called when events are not being received. The callback routine has no parameters. The current window and current menu will not be changed before the idle callback. Programs with multiple windows and/or menus should explicitly set the current window and/or current menu and not rely on its current setting.

The amount of computation and rendering done in an idle callback should be minimized to avoid affecting the program's interactive response. In general, not more than a single frame of rendering should be done in an idle callback.

Passing NULL to glutIdleFunc disables the generation of the idle callback.

EXAMPLE

A typical idle callback to animate a window might look like:

```
void
idle(void)
{
  time += 0.05;
  glutSetWindow(window);
  glutPostRedisplay();
}
```

Notice how the idle callback does not do any actual drawing; it only advances the time scene state global variable. That is left to the window's display callback which will be triggered by the call to glutPostRedisplay.

If you use the idle callback for animation, you should be sure to stop rendering when the window is not visible. This is easy to set up with a visibility callback. For example:

```
void
visible(int vis)
{
  if (vis == GLUT_VISIBLE)
    glutIdleFunc(idle);
  else
    glutIdleFunc(NULL);
}
```

If you do use the idle callback for animation, one thing you should *not* do is setup the idle callback before calling glutMainLoop. It is much better to use the visibility callback to install idle callback when the window first becomes visible on the screen.

SEE ALSO

glutTimerFunc, glutVisibilityFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutIgnoreKeyRepeat - determines if auto repeat keystrokes are reported to the current window

SYNTAX

void glutIgnoreKeyRepeat(int ignore);

ARGUMENTS

ignore Non-zero indicates auto repeat keystrokes should not be reported by the keyboard and special callbacks; zero indicates that auto repeat keystrokes will be reported.

DESCRIPTION

glutIgnoreKeyRepeat determines if auto repeat keystrokes are reported to the current window. The ignore auto repeat state of a window can be queried with glutDeviceGet(GLUT_DEVICE_IGNORE_KEY_REPEAT).

Ignoring auto repeated keystrokes is generally done in conjunction with using the glutKeyboardUpFunc and glutSpecialUpFunc callbacks to repeat key releases. If you do not ignore auto repeated keystrokes, your GLUT application will experience repeated release/press callbacks. Games using the keyboard will typically want to ignore key repeat.

GLUT IMPLEMENTATION NOTES FOR X11

X11 sends KeyPress events repeatedly when the window system's global auto repeat is enabled. glutIgnoreKeyRepeat can prevent these auto repeated keystrokes from being reported as keyboard or special callbacks, but there is still some minimal overhead by the X server to continually stream KeyPress events to the GLUT application. The glutSetKeyRepeat routine can be used to actually disable the global sending of auto repeated KeyPress events. Note that glutSetKeyRepeat affects the global window system auto repeat state so other applications will not auto repeat if you disable auto repeat globally through glutSetKeyRepeat.

SEE ALSO

glutSetKeyRepeat, glutDeviceGet, glutKeyboardFunc, glutKeyboardUpFunc, glutSpecialFunc, glutSpecialUpFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutInit - initialize the GLUT library.

SYNTAX

void glutInit(int *argcp, char **argv);

ARGUMENTS

argcp A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

DESCRIPTION

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options.

glutInit also processes command line options, but the specific options parse are window system dependent.

X IMPLEMENTATION NOTES

The X Window System specific options parsed by glutInit are as follows:

−display *DISPLAY*
Specify the X server to connect to. If not specified, the value of the DISPLAY environment variable is used.

−geometry *WxH+X+Y*
Determines where window's should be created on the screen. The parameter following -geometry should be formatted as a standard X geometry specification. The effect of using this option is to change the GLUT initial size and initial position the same as if glutInitWindowSize or glutInitWindowPosition were called directly.

−iconic Requests all top-level windows be created in an iconic state.

−indirect
Force the use of indirect OpenGL rendering contexts.

−direct Force the use of direct OpenGL rendering contexts (not all GLX implementations support direct rendering contexts). A fatal error is generated if direct rendering is not supported by the OpenGL implementation.

If neither -indirect or -direct are used to force a particular behavior, GLUT will attempt to use direct rendering if possible and otherwise fallback to indirect rendering.

−gldebug
After processing callbacks and/or events, check if there are any OpenGL errors by calling glGetError. If an error is reported, print out a warning by looking up the error code with gluErrorString. Using this option is helpful in detecting OpenGL run-time errors.

−sync Enable synchronous X protocol transactions. This option makes it easier to track down potential X protocol errors.

SEE ALSO

glutCreateWindow, glutInitWindowPosition, glutInitWindowSize, glutMainLoop

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutInitDisplayMode - sets the initial display mode.

SYNTAX

void glutInitDisplayMode(unsigned int mode);

ARGUMENTS

mode Display mode, normally the bitwise {\m OR}-ing of GLUT display mode bit masks. See values below:

GLUT_RGBA
Bit mask to select an RGBA mode window. This is the default if neither GLUT_RGBA nor GLUT_INDEX are specified.

GLUT_RGB
An alias for GLUT_RGBA.

GLUT_INDEX
Bit mask to select a color index mode window. This overrides GLUT_RGBA if it is also specified.

GLUT_SINGLE
Bit mask to select a single buffered window. This is the default if neither GLUT_DOUBLE or GLUT_SINGLE are specified.

GLUT_DOUBLE
Bit mask to select a double buffered window. This overrides GLUT_SINGLE if it is also specified.

GLUT_ACCUM
Bit mask to request a window with an accumulation buffer.

GLUT_ALPHA
Bit mask to request a window with an alpha component to the color buffer(s).

GLUT_DEPTH
Bit mask to request a window with a depth buffer.

GLUT_STENCIL
Bit mask to request a window with a stencil buffer.

GLUT_MULTISAMPLE
Bit mask to request a window with multisampling support. If multisampling is not available, a non-multisampling window will automatically be chosen. Note: both the OpenGL client-side and server-side implementations must support the GLX_SAMPLE_SGIS extension for multisampling to be available.

GLUT_STEREO
Bit mask to select a stereo window.

GLUT_LUMINANCE
Bit mask to select a window with a ‘‘luminance’’ color model. This model provides the functionality of OpenGL’s RGBA color model, but the green and blue components are not maintained in the frame buffer. Instead each pixel’s red component is converted to an index between zero and glutGet(GLUT_WINDOW_COLORMAP_SIZE)-1 and looked up in a per-window color map to determine the color of pixels within the window. The initial colormap of GLUT_LUMINANCE windows is initialized to be a linear gray ramp, but can be modified with GLUT’s colormap routines.

DESCRIPTION

The initial display mode is used when creating top-level windows, subwindows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

Note that GLUT_RGBA selects the RGBA color model, but it does not request any bits of alpha (sometimes called an alpha buffer or destination alpha) be allocated. To request alpha, specify GLUT_ALPHA. The same applies to GLUT_LUMINANCE.

Note that some bits "request" a capability and other bits "select" a capability. A requestable capability may be assigned to the created window even if the bit for the capability was not set. For example, GLUT may create a window with a depth buffer even though GLUT_DEPTH is not specified.

The glutInitDisplayString routine provides a more powerful way to select frame buffer capabilities for GLUT windows.

GLUT_LUMINANCE IMPLEMENTATION NOTES

GLUT_LUMINANCE is not supported on most OpenGL platforms.

SEE ALSO

glutInit, glutCreateWindow, glutInitDisplayString

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutInitDisplayString - sets the initial display mode via a string.

SYNTAX

void glutInitDisplayString(char *string);

ARGUMENTS

string Display mode description string, see below.

DESCRIPTION

The initial display mode description string is used when creating top-level windows, subwindows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

The string is a list of zero or more capability descriptions separated by spaces and tabs. Each capability description is a capability name that is optionally followed by a comparator and a numeric value. For example, "double" and "depth>=12" are both valid criteria.

The capability descriptions are translated into a set of criteria used to select the appropriate frame buffer configuration.

The criteria are matched in strict left to right order of precdence. That is, the first specified criteria (leftmost) takes precedence over the later criteria for non-exact criteria (greater than, less than, etc. comparators). Exact criteria (equal, not equal compartors) must match exactly so precedence is not relevant.

The numeric value is an integer that is parsed according to ANSI C's strtol(str, strptr, 0) behavior. This means that decimal, octal (leading 0), and hexidecimal values (leading 0x) are accepeted.

The valid compartors are:

= Equal.

!= Not equal.

< Less than and preferring larger difference (the least is best).

> Greater than and preferring larger differences (the most is best).

<= Less than or equal and preferring larger difference (the least is best).

>= Greater than or equal and preferring more instead of less. This comparator is useful for allocating resources like color precsion or depth buffer precision where the maximum precison is generally preferred. Contrast with the tilde (~) comprator.

~ Greater than or equal but preferring less instead of more. This compartor is useful for allocating resources such as stencil bits or auxillary color buffers where you would rather not over allocate.

When the compartor and numeric value are not specified, each capability name has a different default (one default is to require a a compartor and numeric value).

The valid capability names are:

alpha Alpha color buffer precision in bits. Default is ">=1".

acca Red, green, blue, and alpha accumulation buffer precision in bits. Default is ">=1" for red, green, blue, and alpha capabilities.

acc Red, green, and green accumulation buffer precision in bits and zero bits of alpha accumulation buffer precision. Default is ">=1" for red, green, and blue capabilities, and "~0" for the alpha capability.

blue Blue color buffer precision in bits. Default is ">=1".

buffer Number of bits in the color index color buffer. Default is ">=1".

conformant
Boolean indicating if the frame buffer configuration is conformant or not. Conformance information is based on GLX's EXT_visual_rating extension if supported. If the extension is not supported, all visuals are assumed conformat. Default is "=1".

depth Number of bits of precsion in the depth buffer. Default is ">=12".

double Boolean indicating if the color buffer is double buffered. Default is "=1".

green Green color buffer precision in bits. Default is ">=1".

index Boolean if the color model is color index or not. True is color index. Default is ">=1".

num A special capability name indicating where the value represents the Nth frame buffer configuration matching the description string. When not specified, glutInitDisplayString also returns the first (best matching) configuration. num requires a compartor and numeric value.

red Red color buffer precision in bits. Default is ">=1".

rgba Number of bits of red, green, blue, and alpha in the RGBA color buffer. Default is ">=1" for red, green, blue, and alpha capabilities, and "=1" for the RGBA color model capability.

rgb Number of bits of red, green, and blue in the RGBA color buffer and zero bits of alpha color buffer precision. Default is ">=1" for the red, green, and blue capabilities, and "~0" for alpha capability, and "=1" for the RGBA color model capability.

luminance
Number of bits of red in the RGBA and zero bits of green, blue (alpha not specified) of color buffer precision. Default is ">=1" for the red capabilitis, and "=0" for the green and blue capabilities, and "=1" for the RGBA color model capability, and, for X11, "=1" for the StaticGray ("xstaticgray") capability.

SGI InfiniteReality (and other future machines) support a 16-bit luminance (single channel) display mode (an additional 16-bit alpha channel can also be requested). The red channel maps to gray scale and green and blue channels are not available. A 16-bit precision luminance display mode is often appropriate for medical imaging applications. Do not expect many machines to support extended precision luminance display modes.

stencil Number of bits in the stencil buffer.

single Boolean indicate the color buffer is single buffered. Double buffer capability "=1".

stereo Boolean indicating the color buffer is supports OpenGL-style stereo. Default is "=1".

samples Indicates the number of multisamples to use based on GLX's SGIS_multisample extension (for antialiasing). Default is "<=4". This default means that a GLUT application can request multipsampling if available by simply specifying "samples".

slow Boolean indicating if the frame buffer configuration is slow or not. For the X11 implementation of GLUT, slowness information is based on GLX's EXT_visual_rating extension if supported. If the EXT_visual_rating extension is not supported, all visuals are assumed fast. For the Win32 implementation of GLUT, slowness is based on if the underlying Pixel Format Descriptor (PFD) is marked "generic" and not "accelerated". This implies that Microsoft's relatively slow software OpenGL implementation is used by this PFD. Note that slowness is a relative designation relative to other frame buffer configurations available. The intent of the slow capability is to help programs avoid frame buffer configurations that are slower (but perhaps higher precision) for the current machine. Default is ">=0" if not comparator and numeric value are provided. This default means that slow visuals are used in preference to fast visuals, but fast visuals will still be allowed.

win32pfd
Only recognized on GLUT implementations for Win32, this capability name matches the Win32 Pixel Format Descriptor by numer. win32pfd requires a compartor and numeric value.

xvisual Only recongized on GLUT implementations for the X Window System, this capability name matches the X visual ID by number. xvisual requires a compartor and numeric value.

xstaticgray
Only recongized on GLUT implementations for the X Window System, boolean indicating if the frame buffer configuration's X visual is of type StaticGray. Default is "=1".

xgrayscale
Only recongized on GLUT implementations for the X Window System, boolean indicating if the frame buffer configuration's X visual is of type GrayScale. Default is "=1".

xstaticcolor
Only recongized on GLUT implementations for the X Window System, boolean indicating if the frame buffer configuration's X visual is of type StaticColor. Default is "=1".

xpseudocolor
Only recongized on GLUT implementations for the X Window System, boolean indicating if the frame buffer configuration's X visual is of type PsuedoColor. Default is "=1".

xtruecolor
Only recongized on GLUT implementations for the X Window System, boolean indicating if the frame buffer configuration's X visual is of type TrueColor. Default is "=1".

xdirectcolor
Only recongized on GLUT implementations for the X Window System, boolean indicating if the frame buffer configuration's X visual is of type DirectColor. Default is "=1".

Unspecifed capability descriptions will result in unspecified criteria being generated. These unspecified criteria help glutInitDisplayString behave sensibly with terse display mode description strings. For example, if no "slow" capability description is provided, fast frame buffer configurations will be choosen in preference to slow frame buffer configurations, but slow frame buffer configurations will still be choosen if no better fast frame buffer configuration is available.

EXAMPLE

Here is an examples using glutInitDisplayString:

```
glutInitDisplayString("stencil~2 rgb double depth>=16 samples");
```

The above call requests a window with an RGBA color model (but requesting no bits of alpha), a depth buffer with at least 16 bits of precsion but preferring more, mutlisampling if available, and at least 2 bits of stencil (favoring less stencil to more as long as 2 bits are available).

SEE ALSO

glutInit, glutCreateWindow, glutInitDisplayMode

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutInitWindowPosition, glutInitWindowSize - set the initial window position and size respectively.

SYNTAX

```
void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);
```

ARGUMENTS

width	Width in pixels.
height	Height in pixels.
x	Window X location in pixels.
y	Window Y location in pixels.

DESCRIPTION

Windows created by glutCreateWindow will be requested to be created with the current initial window position and size.

The initial value of the initial window position GLUT state is -1 and -1. If either the X or Y component to the initial window position is negative, the actual window position is left to the window system to determine. The initial value of the initial window size GLUT state is 300 by 300. The initial window size components must be greater than zero.

The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified size or position. A GLUT program should use the window's reshape callback to determine the true size of the window.

EXAMPLE

If you would like your GLUT program to default to starting at a given screen location and at a given size, but you would also like to let the user override these defaults via a command line argument (such as -geometry for X11), call glutInitWindowSize and glutInitWindowPosition *before* your call to glutInit. For example:

```
int main(int argc, char **argv)
{
  glutInitWindowSize(500, 300);
  glutInitWindowPosition(100, 100);
  glutInit(&argc, argv);
  ...
}
```

However, if you'd like to force your program to start up at a given size, call glutInitWindowSize and glutInitWindowPosition *after* your call to glutInit. For example:

```
int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitWindowSize(500, 300);
  glutInitWindowPosition(100, 100);
  ...
}
```

SEE ALSO

glutInit, glutCreateWindow, glutCreateSubWindow, glutReshapeFunc, glutGet

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutJoystickFunc - sets the joystick callback for the current window.

SYNTAX

void glutJoystickFunc(void (*func)(unsigned int buttonMask,
int x, int y, int z), int pollInterval);

ARGUMENTS

func	The new joystick callback function.
pollInterval	Joystick polling interval in milliseconds.

DESCRIPTION

glutJoystickFunc sets the joystick callback for the current window.

The joystick callback is called either due to polling of the joystick at the uniform timer interval specified by pollInterval (in milliseconds) or in response to calling glutForceJoystickFunc. If the pollInterval is non-positive, no joystick polling is performed and the GLUT application must frequently (usually from an idle callback) call glutForceJoystickFunc.

The joystick buttons are reported by the callback's buttonMask parameter. The constants GLUT_JOYSTICK_BUTTON_A (0x1), GLUT_JOYSTICK_BUTTON_B (0x2), GLUT_JOYSTICK_BUTTON_C (0x4), and GLUT_JOYSTICK_BUTTON_D (0x8) are provided for programming convience.

The x, y, and z callback parameters report the X, Y, and Z axes of the joystick. The joystick is centered at (0,0,0). X, Y, and Z are scaled to range between -1000 and 1000. Moving the joystick left reports negative X; right reports positive X. Pulling the stick towards you reports negative Y; push the stick away from you reports positive Y. If the joystick has a third axis (rudder or up/down), down reports negative Z; up reports positive Z.

Passing a NULL func to glutJoystickFunc disables the generation of joystick callbacks. Without a joystick callback registered, glutForceJoystickFunc does nothing.

When a new window is created, no joystick callback is initially registered.

LIMITATIONS

The GLUT joystick callback only reports the first 3 axes and 32 buttons. GLUT supports only a single joystick.

GLUT IMPLEMENTATION NOTES FOR X11

The current implementation of GLUT for X11 supports the joystick API, but not joystick input. A future implementation of GLUT for X11 may add joystick support.

GLUT IMPLEMENTATION NOTES FOR WIN32

The current implementation of GLUT for Win32 supports the joystick API and joystick input, but does so through the dated joySetCapture and joyGetPosEx Win32 Multimedia API. The current GLUT joystick support for Win32 has all the limitations of the Win32 Multimedia API joystick support. A future implementation of GLUT for Win32 may use DirectInput.

GLUT IMPLEMENTATION NOTES FOR NON-ANALOG JOYSTICKS

If the connected joystick does not return (x,y,z) as a continuous range (for example, an 8 position Atari 2600 joystick), the implementation should report the most extreme (x,y,z) location. That is, if a 2D joystick is pushed to the upper left, report (-1000,1000,0).

SEE ALSO

glutForceJoystickFunc, glutMotionFunc, glutMouseFunc, glutSpaceballButtonFunc, glutSpaceballMotionFunc, glutButtonBoxFunc, glutTabletButtonFunc, glutDeviceGet

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutKeyboardFunc - sets the keyboard callback for the current window.

SYNTAX

```
void glutKeyboardFunc(void (*func)(unsigned char key,
        int x, int y));
```

ARGUMENTS

func The new keyboard callback function.

DESCRIPTION

glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

During a keyboard callback, glutGetModifiers may be called to determine the state of modifier keys when the keystroke generating the callback occurred.

Use glutSpecialFunc for a means to detect non-ASCII key strokes.

SEE ALSO

glutKeyboardUpFunc, glutSpecialFunc, glutCreateWindow, glutMouseFunc, glutSpaceballButtonFunc, glutButtonBoxFunc, glutTabletButtonFunc, glutGetModifiers

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutKeyboardUpFunc - sets the keyboard up (key release) callback for the current window.

SYNTAX

```
void glutKeyboardUpFunc(void (*func)(unsigned char key,
        int x, int y));
```

ARGUMENTS

func The new keyboard up callback function.

DESCRIPTION

glutKeyboardFunc sets the keyboard up (key release) callback for the current window. When a user types into the window, each key release matching an ASCII character will generate a keyboard up callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

During a keyboard up callback, glutGetModifiers may be called to determine the state of modifier keys when the keystroke generating the callback occurred.

To avoid the reporting of key release/press pairs due to auto repeat, use glutIgnoreKeyRepeat to ignore auto repeated keystrokes.

There is no guarantee that the keyboard press callback will match the exact ASCII character as the keyboard up callback. For example, the key down may be for a lowercase b, but the key release may report an uppercase B if the shift state has changed. The same applies to symbols and control characters. The precise behavior is window system dependent.

Use glutSpecialUpFunc for a means to detect non-ASCII key releases.

SEE ALSO

glutKeyboardFunc, glutSpecialUpFunc, glutSpecialFunc, glutCreateWindow, glutMouseFunc, glutSpaceballButtonFunc, glutButtonBoxFunc, glutTabletButtonFunc, glutGetModifiers, glutIgnoreKeyRepeat

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutLayerGet - retrieves GLUT state pertaining to the layers of the current window.

SYNTAX

int glutLayerGet(GLenum info);

ARGUMENTS

info Name of device information to retrieve.

GLUT_OVERLAY_POSSIBLE
Whether an overlay could be established for the current window given the current initial display mode. If false, glutEstablishOverlay will fail with a fatal error if called.

GLUT_LAYER_IN_USE
Either GLUT_NORMAL or GLUT_OVERLAY depending on whether the normal plane or overlay is the layer in use.

GLUT_HAS_OVERLAY
If the current window has an overlay established.

GLUT_TRANSPARENT_INDEX
The transparent color index of the overlay of the current window; negative one is returned if no overlay is in use.

GLUT_NORMAL_DAMAGED
True if the normal plane of the current window has damaged (by window system activity) since the last display callback was triggered. Calling glutPostRedisplay will not set this true.

GLUT_OVERLAY_DAMAGED
True if the overlay plane of the current window has damaged (by window system activity) since the last display callback was triggered. Calling glutPostRedisplay or glutPostOverlayRedisplay will not set this true. Negative one is returned if no overlay is in use.

DESCRIPTION

glutLayerGet retrieves GLUT layer information for the current window represented by integers. The info parameter determines what type of layer information to return.

SEE ALSO

glutEstablishOverlay, glutUseOverlay, glutCreateWindow, glutSetColor

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutMainLoop - enters the GLUT event processing loop.

SYNTAX

void glutMainLoop(void);

DESCRIPTION

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

SEE ALSO

glutInit

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutMenuStatusFunc - sets the global menu status callback.

SYNTAX

void glutMenuStatusFunc(void (*func)(int status, int x, int y));
void glutMenuStateFunc(void (*func)(int status));

ARGUMENTS

func The new menu status (or state) callback function.

DESCRIPTION

glutMenuStatusFunc sets the global menu status callback so a GLUT program can determine when a menu is in use or not. When a menu status callback is registered, it will be called with the value GLUT_MENU_IN_USE for its value parameter when pop-up menus are in use by the user; and the callback will be called with the value GLUT_MENU_NOT_IN_USE for its status parameter when pop-up menus are no longer in use. The x and y parameters indicate the location in window coordinates of the button press that caused the menu to go into use, or the location where the menu was released (may be outside the window). The func parameter names the callback function. Other callbacks continue to operate (except mouse motion callbacks) when pop-up menus are in use so the menu status callback allows a program to suspend animation or other tasks when menus are in use. The cascading and unmapping of sub-menus from an initial pop-up menu does not generate menu status callbacks. There is a single menu status callback for GLUT.

When the menu status callback is called, the current menu will be set to the initial pop-up menu in both the GLUT_MENU_IN_USE and GLUT_MENU_NOT_IN_USE cases. The current window will be set to the window from which the initial menu was popped up from, also in both cases.

Passing NULL to glutMenuStatusFunc disables the generation of the menu status callback.

glutMenuStateFunc is a deprecated version of the glutMenuStatusFunc routine. The only difference is glutMenuStateFunc callback prototype does not deliver the two additional x and y coordinates.

SEE ALSO

glutCreateMenu, glutCreateWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutMotionFunc, glutPassiveMotionFunc - set the motion and passive motion callbacks respectively for the current window.

SYNTAX

```
void glutMotionFunc(void (*func)(int x, int y));
void glutPassiveMotionFunc(void (*func)(int x, int y));
```

ARGUMENTS

func The new motion or passive motion callback function.

DESCRIPTION

glutMotionFunc and glutPassiveMotionFunc set the motion and passive motion callback respectively for the current window. The motion callback for a window is called when the mouse moves within the window while one or more mouse buttons are pressed. The passive motion callback for a window is called when the mouse moves within the window while no mouse buttons are pressed.

The x and y callback parameters indicate the mouse location in window relative coordinates.

Passing NULL to glutMotionFunc or glutPassiveMotionFunc disables the generation of the mouse or passive motion callback respectively.

SEE ALSO

glutMouseFunc, glutSpaceballMotionFunc, glutTabletMotionFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutMouseFunc - sets the mouse callback for the current window.

SYNTAX

```
void glutMouseFunc(void (*func)(int button, int state,
        int x, int y));
```

ARGUMENTS

func The new mouse callback function.

DESCRIPTION

glutMouseFunc sets the mouse callback for the current window. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON. For systems with only two mouse buttons, it may not be possible to generate GLUT_MIDDLE_BUTTON callback. For systems with a single mouse button, it may be possible to generate only a GLUT_LEFT_BUTTON callback. The state parameter is either GLUT_UP or GLUT_DOWN indicating whether the callback was due to a release or press respectively. The x and y callback parameters indicate the window relative coordinates when the mouse button state changed. If a GLUT_DOWN callback for a specific button is triggered, the program can assume a GLUT_UP callback for the same button will be generated (assuming the window still has a mouse callback registered) when the mouse button is released even if the mouse has moved outside the window.

If a menu is attached to a button for a window, mouse callbacks will not be generated for that button.

During a mouse callback, glutGetModifiers may be called to determine the state of modifier keys when the mouse event generating the callback occurred.

Passing NULL to glutMouseFunc disables the generation of mouse callbacks.

SEE ALSO

glutKeyboardFunc, glutMotionFunc, glutSpaceballButtonFunc, glutButtonBoxFunc, glutTabletButtonFunc, glutGetModifiers

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutOverlayDisplayFunc - sets the overlay display callback for the current window.

SYNTAX

void glutOverlayDisplayFunc(void (*func)(void));

ARGUMENTS

func The new overlay display callback function.

DESCRIPTION

glutDisplayFunc sets the overlay display callback for the current window. The overlay display callback is functionally the same as the window's display callback except that the overlay display callback is used to redisplay the window's overlay.

When GLUT determines that the overlay plane for the window needs to be redisplayed, the overlay display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and the layer in use is set to the overlay. The overlay display callback is called with no parameters. The entire overlay region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state).

GLUT determines when the overlay display callback should be triggered based on the window's overlay redisplay state. The overlay redisplay state for a window can be either set explicitly by calling glutPostOverlayRedisplay or implicitly as the result of window damage reported by the window system. Multiple posted overlay redisplays for a window are coalesced by GLUT to minimize the number of overlay display callbacks called.

Upon return from the overlay display callback, the overlay damaged state of the window (returned by calling glutLayerGet(GLUT_OVERLAY_DAMAGED) is cleared.

The overlay display callback can be deregistered by passing NULL to glutOverlayDisplayFunc. The overlay display callback is initially NULL when an overlay is established. See glutDisplayFunc to understand how the display callback alone is used if an overlay display callback is not registered.

SEE ALSO

glutDisplayFunc, glutPostOverlayRedisplay

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutPopWindow, glutPushWindow - change the stacking order of the current window relative to its siblings.

SYNTAX

void glutPopWindow(void);
void glutPushWindow(void);

DESCRIPTION

glutPopWindow and glutPushWindow work on both top-level windows and subwindows. The effect of pushing and popping windows does not take place immediately. Instead the push or pop is saved for execution upon return to the GLUT event loop. Subsequent push or pop requests on a window replace the previously saved request for that window. The effect of pushing and popping top-level windows is subject to the window system's policy for restacking windows.

SEE ALSO

glutShowWindow, glutIconifyWindow, glutHideWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutPositionWindow - requests a change to the position of the current window.

SYNTAX

void glutPositionWindow(int x, int y);

ARGUMENTS

x	New X location of window in pixels.
y	New Y location of window in pixels.

DESCRIPTION

glutPositionWindow requests a change in the position of the current window. For top-level windows, the x and y parameters are pixel offsets from the screen origin. For subwindows, the x and y parameters are pixel offsets from the window's parent window origin.

The requests by glutPositionWindow are not processed immediately. The request is executed after returning to the main event loop. This allows multiple glutPositionWindow, glutReshapeWindow, and glutFullScreen requests to the same window to be coalesced.

In the case of top-level windows, a glutPositionWindow call is considered only a request for positioning the window. The window system is free to apply its own policies to top-level window placement. The intent is that top-level windows should be repositioned according glutPositionWindow's parameters.

glutPositionWindow disables the full screen status of a window if previously enabled.

SEE ALSO

glutInitWindowPosition, glutReshapeWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutPostOverlayRedisplay, glutPostWindowOverlayRedisplay - marks the overlay of the current or specified window as needing to be redisplayed.

SYNTAX

void glutPostOverlayRedisplay(void);
void glutPostWindowOverlayRedisplay(int win);

DESCRIPTION

Mark the overlay of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's overlay display callback (or simply the display callback if no overlay display callback is registered) will be called to redisplay the window's overlay plane. Multiple calls to glutPostOverlayRedisplay before the next display callback opportunity (or overlay display callback opportunity if one is registered) generate only a single redisplay. glutPostOverlayRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

Logically, overlay damage notification for a window is treated as a glutPostOverlayRedisplay on the damaged window. Unlike damage reported by the window system, glutPostOverlayRedisplay will not set to true the overlay's damaged status (returned by glutLayerGet(GLUT_OVERLAY_DAMAGED).

If the window you want to post an overlay redisplay on is not already current (and you do not require it to be immediately made current), using glutPostWindowOverlayRedisplay is more efficient that calling glutSetWindow to the desired window and then calling glutPostOverlayRedisplay.

EXAMPLE

If you are doing an interactive effect like rubberbanding in the overlay, it is a good idea to structure your rendering to minimize flicker (most overlays are single-buffered). Only clear the overlay if you know that the window has been damaged. Otherwise, try to simply erase what you last drew and redraw it in an updated position. Here is an example overlay display callback used to implement overlay rubberbanding:

```
void
redrawOverlay(void)
{
  static int prevStretchX, prevStretchY;

  if (glutLayerGet(GLUT_OVERLAY_DAMAGED)) {
    /* Damage means we need a full clear. */
    glClear(GL_COLOR_BUFFER_BIT);
  } else {
    /* Undraw last rubber-band. */
    glIndexi(transparent);
    glBegin(GL_LINE_LOOP);
    glVertex2i(anchorX, anchorY);
    glVertex2i(anchorX, prevStretchY);
    glVertex2i(prevStretchX, prevStretchY);
    glVertex2i(prevStretchX, anchorY);
    glEnd();
  }
  glIndexi(red);
  glBegin(GL_LINE_LOOP);
  glVertex2i(anchorX, anchorY);
  glVertex2i(anchorX, stretchY);
  glVertex2i(stretchX, stretchY);
  glVertex2i(stretchX, anchorY);
  glEnd();
  prevStretchX = stretchX;
  prevStretchY = stretchY;
```

```
}
```

Notice how glutLayerGet(GLUT_OVERLAY_DAMAGED) is used to determine if a clear needs to take place because of damage; if a clear is unnecessary, it is faster to just draw the last rubberband using the transparent pixel.

When the application is through with the rubberbanding effect, the best way to get ride of the rubberband is to simply hide the overlay by calling glutHideOverlay.

SEE ALSO

glutPostRedisplay, glutEstablishOverlay, glutLayerGet

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutPostRedisplay, glutPostWindowRedisplay - marks the current or specified window as needing to be redisplayed.

SYNTAX

void glutPostRedisplay(void);
void glutPostWindowRedisplay(int win);

DESCRIPTION

glutPostRedisplay marks the normal plane of current window as needing to be redisplayed. glutPostWindowRedisplay works the specified window as needing to be redisplayed. After either call, the next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

Logically, normal plane damage notification for a window is treated as a glutPostRedisplay on the damaged window. Unlike damage reported by the window system, glutPostRedisplay will not set to true the normal plane's damaged status (returned by glutLayerGet(GLUT_NORMAL_DAMAGED).

If the window you want to post a redisplay on is not already current (and you do not require it to be immediately made current), using glutPostWindowRedisplay is more efficient that calling glutSetWindow to the desired window and then calling glutPostRedisplay.

SEE ALSO

glutPostOverlayRedisplay, glutDisplayFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutRemoveMenuItem - remove the specified menu item.

SYNTAX

void glutRemoveMenuItem(int entry);

ARGUMENTS

entry Index into the menu items of the current menu (1 is the topmost menu item).

DESCRIPTION

glutRemoveMenuItem remove the entry menu item regardless of whether it is a menu entry or sub-menu trigger. entry must be between 1 and glutGet(GLUT_MENU_NUM_ITEMS) inclusive. Menu items below the removed menu item are renumbered.

SEE ALSO

glutAddMenuEntry, glutAddSubMenu, glutChangeToMenuEntry, glutChangeToSubMenu

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutRemoveOverlay - removes the overlay (if one exists) from the current window.

SYNTAX

void glutRemoveOverlay(void);

DESCRIPTION

glutRemoveOverlay removes the overlay (if one exists). It is safe to call glutRemoveOverlay even if no overlay is currently established--it does nothing in this case. Implicitly, the window's layer in use changes to the normal plane immediately once the overlay is removed.

If the program intends to re-establish the overlay later, it is typically faster and less resource intensive to use glutHideOverlay and glutShowOverlay to simply change the display status of the overlay.

SEE ALSO

glutEstablishOverlay, glutDestroyWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutReportErrors - for debugging purposes; prints out OpenGL run-time errors.

SYNTAX

void glutReportErrors(void);

DESCRIPTION

This routine prints out any OpenGL run-time errors pending and clears the errors. This routine typically should only be used for debugging purposes since calling it will slow OpenGL programs. It is provided as a convenience; all the routine does is call *glGetError* until no more errors are reported. Any errors detected are reported with a GLUT warning and the corresponding text message generated by *gluErrorString.*

Calling glutReportErrors repeatedly in your program can help isolate OpenGL errors to the offending OpenGL command. Remember that you can use the *-gldebug* option to detect OpenGL errors in any GLUT program.

SEE ALSO

glutInit, glutCreateWindow, glutInitDisplayMode, gluErrorString, glGetError

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutReshapeFunc - sets the reshape callback for the current window.

SYNTAX

void glutReshapeFunc(void (*func)(int width, int height));

ARGUMENTS

func The new reshape callback function.

DESCRIPTION

glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped.

If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply call glViewport(0,0,width,height) on the normal plane (and on the overlay if one exists).

If an overlay is established for the window, a single reshape callback is generated. It is the callback's responsibility to update both the normal plane and overlay for the window (changing the layer in use as necessary).

When a top-level window is reshaped, subwindows are not reshaped. It is up to the GLUT program to manage the size and positions of subwindows within a top-level window. Still, reshape callbacks will be triggered for subwindows when their size is changed using glutReshapeWindow.

SEE ALSO

glutDisplayFunc, glutReshapeWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutReshapeWindow - requests a change to the size of the current window.

SYNTAX

void glutReshapeWindow(int width, int height);

ARGUMENTS

width	New width of window in pixels.
height	New height of window in pixels.

DESCRIPTION

glutReshapeWindow requests a change in the size of the current window. The width and height parameters are size extents in pixels. The width and height must be positive values.

The requests by glutReshapeWindow are not processed immediately. The request is executed after returning to the main event loop. This allows multiple glutReshapeWindow, glutPositionWindow, and glutFullScreen requests to the same window to be coalesced.

In the case of top-level windows, a glutReshapeWindow call is considered only a request for sizing the window. The window system is free to apply its own policies to top-level window sizing. The intent is that top-level windows should be reshaped according glutReshapeWindow's parameters. Whether a reshape actually takes effect and, if so, the reshaped dimensions are reported to the program by a reshape callback.

glutReshapeWindow disables the full screen status of a window if previously enabled.

SEE ALSO

glutPositionWindow, glutReshapeFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSetColor - sets the color of a colormap entry in the layer of use for the current window.

SYNTAX

```
void glutSetColor(int cell,
                  GLfloat red, GLfloat green, GLfloat blue);
```

ARGUMENTS

cell	Color cell index (starting at zero).
red	Red intensity (clamped between 0.0 and 1.0 inclusive).
green	Green intensity (clamped between 0.0 and 1.0 inclusive).
blue	Blue intensity (clamped between 0.0 and 1.0 inclusive).

DESCRIPTION

Sets the cell color index colormap entry of the current window's logical colormap for the layer in use with the color specified by red, green, and blue. The layer in use of the current window should be a color index window. cell should be zero or greater and less than the total number of colormap entries for the window. If the layer in use's colormap was copied by reference, a glutSetColor call will force the duplication of the colormap. Do not attempt to set the color of an overlay's transparent index.

SEE ALSO

glutGetColor, glutCopyColormap, glutInitDisplayMode

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSetCursor - changes the cursor image of the current window.

SYNTAX

void glutSetCursor(int cursor);

ARGUMENTS

cursor Name of cursor image to change to. Possible values follow:

GLUT_CURSOR_RIGHT_ARROW
Arrow pointing up and to the right.

GLUT_CURSOR_LEFT_ARROW
Arrow pointing up and to the left.

GLUT_CURSOR_INFO
Pointing hand.

GLUT_CURSOR_DESTROY
Skull & cross bones.

GLUT_CURSOR_HELP
Question mark.

GLUT_CURSOR_CYCLE
Arrows rotating in a circle.

GLUT_CURSOR_SPRAY
Spray can.

GLUT_CURSOR_WAIT
Wrist watch.

GLUT_CURSOR_TEXT
Insertion point cursor for text.

GLUT_CURSOR_CROSSHAIR
Simple cross-hair.

GLUT_CURSOR_UP_DOWN
Bi-directional pointing up & down.

GLUT_CURSOR_LEFT_RIGHT
Bi-directional pointing left & right.

GLUT_CURSOR_TOP_SIDE
Arrow pointing to top side.

GLUT_CURSOR_BOTTOM_SIDE
Arrow pointing to bottom side.

GLUT_CURSOR_LEFT_SIDE
Arrow pointing to left side.

GLUT_CURSOR_RIGHT_SIDE
Arrow pointing to right side.

GLUT_CURSOR_TOP_LEFT_CORNER
Arrow pointing to top-left corner.

GLUT_CURSOR_TOP_RIGHT_CORNER
Arrow pointing to top-right corner.

GLUT_CURSOR_BOTTOM_RIGHT_CORNER
Arrow pointing to bottom-left corner.

GLUT_CURSOR_BOTTOM_LEFT_CORNER
Arrow pointing to bottom-right corner.

GLUT_CURSOR_FULL_CROSSHAIR
Full-screen cross-hair cursor (if possible, otherwise GLUT_CURSOR_CROSSHAIR).

GLUT_CURSOR_NONE
Invisible cursor.

GLUT_CURSOR_INHERIT
Use parent's cursor.

DESCRIPTION

glutSetCursor changes the cursor image of the current window. Each call requests the window system change the cursor appropriately. The cursor image when a window is created is GLUT_CURSOR_INHERIT. The exact cursor images used are implementation dependent. The intent is for the image to convey the meaning of the cursor name. For a top-level window, GLUT_CURSOR_INHERIT uses the default window system cursor.

X IMPLEMENTATION NOTES

GLUT for X uses SGI's _SGI_CROSSHAIR_CURSOR convention to access a full-screen cross-hair cursor if possible.

SEE ALSO

glutCreateWindow, glutCreateSubWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSetKeyRepeat - sets the key repeat mode

SYNTAX

int glutSetKeyRepeat(int repeatMode);

ARGUMENTS

repeatMode Mode for setting key repeat to.

GLUT_KEY_REPEAT_OFF
Disable key repeat for the window system on a global basis if possible.

GLUT_KEY_REPEAT_ON
Enable key repeat for the window system on a global basis if possible.

GLUT_KEY_REPEAT_DEFAULT
Reset the key repeat mode for the window system to its default state if possible.

DESCRIPTION

glutSetKeyRepeat sets the key repeat mode for the window system on a global basis if possible. If supported by the window system, the key repeat can either be enabled, disabled, or set to the window system's default key repeat state.

GLUT IMPLEMENTATION NOTES FOR X11

X11 sends KeyPress events repeatedly when the window system's global auto repeat is enabled. glutIgnoreKeyRepeat can prevent these auto repeated keystrokes from being reported as keyboard or special callbacks, but there is still some minimal overhead by the X server to continually stream KeyPress events to the GLUT application. The glutSetKeyRepeat routine can be used to actually disable the global sending of auto repeated KeyPress events. Note that glutSetKeyRepeat affects the global window system auto repeat state so other applications will not auto repeat if you disable auto repeat globally through glutSetKeyRepeat.

GLUT applications using the X11 GLUT implemenation should disable key repeat with glutSetKeyRepeat to disable key repeats most efficiently.

GLUT applications that change the key repeat state with glutSetKeyRepeat are responsible for explicitly restoring the default key repeat state on exit.

GLUT IMPLEMENTATION NOTES FOR WIN32

The Win32 implementation of glutSetKeyRepeat does nothing. The glutIgnoreKeyRepeat routine can be used in the Win32 GLUT implementation to ignore repeated keys on a per-window basis without changing the global window system key repeat.

SEE ALSO

glutIgnoreKeyRepeat, glutKeyboardFunc, glutSpecialFunc, glutKeyboardUpFunc, glutSpecialUpFunc, glutDeviceGet

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSetMenu - sets the current menu; glutGetMenu - returns the identifier of the current menu.

SYNTAX

void glutSetMenu(int menu);
int glutGetMenu(void);

ARGUMENTS

menu The identifier of the menu to make the current menu.

DESCRIPTION

glutSetMenu sets the current menu; glutGetMenu returns the identifier of the current menu. If no menus exist or the previous current menu was destroyed, glutGetMenu returns zero.

SEE ALSO

glutCreateMenu, glutSetWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSetWindow - sets the current window; glutGetWindow - returns the identifier of the current window.

SYNTAX

```
void glutSetWindow(int win);
int glutGetWindow(void);
```

ARGUMENTS

win Identifier of GLUT window to make the current window.

DESCRIPTION

glutSetWindow sets the current window; glutGetWindow returns the identifier of the current window. If no windows exist or the previously current window was destroyed, glutGetWindow returns zero. glutSetWindow does not change the layer in use for the window; this is done using glutUseLayer.

SEE ALSO

glutCreateWindow, glutSetMenu

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSetWindowTitle, glutSetIconTitle - change the window or icon title respectively of the current top-level window.

SYNTAX

```
void glutSetWindowTitle(char *name);
void glutSetIconTitle(char *name);
```

ARGUMENTS

name ASCII character string for the window or icon name to be set for the window.

DESCRIPTION

These routines should be called only when the current window is a top-level window. Upon creation of a top-level window, the window and icon names are determined by the name parameter to glutCreateWindow. Once created, glutSetWindowTitle and glutSetIconTitle can change the window and icon names respectively of top-level windows. Each call requests the window system change the title appropriately. Requests are not buffered or coalesced. The policy by which the window and icon name are displayed is window system dependent.

SEE ALSO

glutCreateWindow, glutIconifyWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutShowOverlay, glutHideOverlay - shows or hides the overlay of the current window

SYNTAX

```
void glutShowOverlay(void);
void glutHideOverlay(void);
```

DESCRIPTION

glutShowOverlay shows the overlay of the current window; glutHideOverlay hides the overlay. The effect of showing or hiding an overlay takes place immediately. Note that glutShowOverlay will not actually display the overlay unless the window is also shown (and even a shown window may be obscured by other windows, thereby obscuring the overlay). It is typically faster and less resource intensive to use these routines to control the display status of an overlay as opposed to removing and re-establishing the overlay.

SEE ALSO

glutEstablishOverlay, glutShowWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutShowWindow, glutHideWindow, glutIconifyWindow - change the display status of the current window.

SYNTAX

```
void glutShowWindow(void);
void glutHideWindow(void);
void glutIconifyWindow(void);
```

DESCRIPTION

glutShowWindow will show the current window (though it may still not be visible if obscured by other shown windows). glutHideWindow will hide the current window. glutIconifyWindow will iconify a top-level window, but GLUT prohibits iconification of a subwindow. The effect of showing, hiding, and iconifying windows does not take place immediately. Instead the requests are saved for execution upon return to the GLUT event loop. Subsequent show, hide, or iconification requests on a window replace the previously saved request for that window. The effect of hiding, showing, or iconifying top-level windows is subject to the window system's policy for displaying windows.

SEE ALSO

glutPopWindow, glutPushWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidCone, glutWireCone - render a solid or wireframe cone respectively.

SYNTAX

```
void glutSolidCone(GLdouble base, GLdouble height,
        GLint slices, GLint stacks);
void glutWireCone(GLdouble base, GLdouble height,
        GLint slices, GLint stacks);
```

ARGUMENTS

base	The radius of the base of the cone.
height	The height of the cone.
slices	The number of subdivisions around the Z axis.
stacks	The number of subdivisions along the Z axis.

DESCRIPTION

glutSolidCone and glutWireCone render a solid or wireframe cone respectively oriented along the Z axis. The base of the cone is placed at Z = 0, and the top at Z = height. The cone is subdivided around the Z axis into slices, and along the Z axis into stacks.

SEE ALSO

glutSolidSphere, glutSolidCube, glutSolidTorus, glutSolidDodecahedron, glutSolidOctahedron, glutSolidTetrahedron, glutSolidIcosahedron, glutSolidTeapot

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidCube, glutWireCube - render a solid or wireframe cube respectively.

SYNTAX

void glutSolidCube(GLdouble size);
void glutWireCube(GLdouble size);

ARGUMENTS

size Length of each edge.

DESCRIPTION

glutSolidCube and glutWireCube render a solid or wireframe cube respectively. The cube is centered at the modeling coordinates origin with sides of length size.

SEE ALSO

glutSolidSphere, glutSolidCone, glutSolidTorus, glutSolidDodecahedron, glutSolidOctahedron, glutSolidTetrahedron, glutSolidIcosahedron, glutSolidTeapot

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidDodecahedron, glutWireDodecahedron - render a solid or wireframe dodecahedron (12-sided regular solid) respectively.

SYNTAX

void glutSolidDodecahedron(void);
void glutWireDodecahedron(void);

DESCRIPTION

glutSolidDodecahedron and glutWireDodecahedron render a solid or wireframe dodecahedron respectively centered at the modeling coordinates origin with a radius of sqrt(3).

SEE ALSO

glutSolidSphere, glutSolidCube, glutSolidCone, glutSolidTorus, glutSolidOctahedron, glutSolidTetrahedron, glutSolidIcosahedron, glutSolidTeapot

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidIcosahedron, glutWireIcosahedron - render a solid or wireframe icosahedron (20-sided regular solid) respectively.

SYNTAX

void glutSolidIcosahedron(void);
void glutWireIcosahedron(void);

DESCRIPTION

glutSolidIcosahedron and glutWireIcosahedron render a solid or wireframe icosahedron respectively. The icosahedron is centered at the modeling coordinates origin and has a radius of 1.0.

SEE ALSO

glutSolidSphere, glutSolidCube, glutSolidCone, glutSolidTorus, glutSolidDodecahedron, glutSolidOctahedron, glutSolidTetrahedron, glutSolidTeapot

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidOctahedron, glutWireOctahedron - render a solid or wireframe octahedron (8-sided regular solid) respectively.

SYNTAX

void glutSolidOctahedron(void);
void glutWireOctahedron(void);

DESCRIPTION

glutSolidOctahedron and glutWireOctahedron render a solid or wireframe octahedron respectively centered at the modeling coordinates origin with a radius of 1.0.

SEE ALSO

glutSolidSphere, glutSolidCube, glutSolidCone, glutSolidTorus, glutSolidDodecahedron, glutSolidTetrahedron, glutSolidIcosahedron, glutSolidTeapot

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidSphere, glutWireSphere - render a solid or wireframe sphere respectively.

SYNTAX

```
void glutSolidSphere(GLdouble radius,
        GLint slices, GLint stacks);
void glutWireSphere(GLdouble radius,
        GLint slices, GLint stacks);
```

ARGUMENTS

radius	The radius of the sphere.
slices	The number of subdivisions around the Z axis (similar to lines of longitude).
stacks	The number of subdivisions along the Z axis (similar to lines of latitude).

DESCRIPTION

Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

SEE ALSO

glutSolidCube, glutSolidCone, glutSolidTorus, glutSolidDodecahedron, glutSolidOctahedron, glutSolidTetrahedron, glutSolidIcosahedron, glutSolidTeapot

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidTeapot, glutWireTeapot - render a solid or wireframe teapot respectively.

SYNTAX

void glutSolidTeapot(GLdouble size);
void glutWireTeapot(GLdouble size);

ARGUMENTS

size Relative size of the teapot.

DESCRIPTION

glutSolidTeapot and glutWireTeapot render a solid or wireframe teapot respectively. Both surface normals and texture coordinates for the teapot are generated. The teapot is generated with OpenGL evaluators.

BUGS

The teapot is greatly over-tesselated; it renders way too slow.

OpenGL's default glFrontFace state assumes that front facing polygons (for the purpose of face culling) have vertices that wind counter clockwise when projected into window space. This teapot is rendered with its front facing polygon vertices winding clockwise. For OpenGL's default back face culling to work, you should use:

```
glFrontFace(GL_CW);
glutSolidTeapot(size);
glFrontFace(GL_CCW);
```

Both these bugs reflect issues in the original aux toolkit's teapot rendering routines (GLUT used the same teapot rendering routine).

SEE ALSO

glutSolidSphere, glutSolidCube, glutSolidCone, glutSolidTorus, glutSolidDodecahedron, glutSolidOctahedron, glutSolidTetrahedron, glutSolidIcosahedron

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidTetrahedron, glutWireTetrahedron - render a solid or wireframe tetrahedron (4-sided regular solid) respectively.

SYNTAX

void glutSolidTetrahedron(void);
void glutWireTetrahedron(void);

DESCRIPTION

glutSolidTetrahedron and glutWireTetrahedron render a solid or wireframe tetrahedron respectively centered at the modeling coordinates origin with a radius of sqrt(3).

SEE ALSO

glutSolidSphere, glutSolidCube, glutSolidCone, glutSolidTorus, glutSolidDodecahedron, glutSolidOctahedron, glutSolidIcosahedron, glutSolidTeapot

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSolidTorus, glutWireTorus - render a solid or wireframe torus (doughnut) respectively.

SYNTAX

```
void glutSolidTorus(GLdouble innerRadius,
                    GLdouble outerRadius,
                    GLint nsides, GLint rings);
void glutWireTorus(GLdouble innerRadius,
                   GLdouble outerRadius,
                   GLint nsides, GLint rings);
```

ARGUMENTS

innerRadius	Inner radius of the torus.
outerRadius	Outer radius of the torus.
nsides	Number of sides for each radial section.
rings	Number of radial divisions for the torus.

DESCRIPTION

glutSolidTorus and glutWireTorus render a solid or wireframe torus (doughnut) respectively centered at the modeling coordinates origin whose axis is aligned with the Z axis.

SEE ALSO

glutSolidSphere, glutSolidCube, glutSolidCone, glutSolidDodecahedron, glutSolidOctahedron, glutSolidTetrahedron, glutSolidIcosahedron, glutSolidTeapot

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSpaceballButtonFunc - sets the Spaceball button callback for the current window.

SYNTAX

void glutSpaceballButtonFunc(void (*func)(int button, int state));

ARGUMENTS

func The new spaceball button callback function.

DESCRIPTION

glutSpaceballButtonFunc sets the Spaceball button callback for the current window. The Spaceball button callback for a window is called when the window has Spaceball input focus (normally, when the mouse is in the window) and the user generates Spaceball button presses. The button parameter will be the button number (starting at one). The number of available Spaceball buttons can be determined with glutDeviceGet(GLUT_NUM_SPACEBALL_BUTTONS). The state is either GLUT_UP or GLUT_DOWN indicating whether the callback was due to a release or press respectively.

Registering a Spaceball button callback when a Spaceball device is not available is ineffectual and not an error. In this case, no Spaceball button callbacks will be generated.

Passing NULL to glutSpaceballButtonFunc disables the generation of Spaceball button callbacks. When a new window is created, no Spaceball button callback is initially registered.

SEE ALSO

glutSpaceballMotionFunc, glutSpaceballRotateFunc, glutMouseFunc, glutButtonBoxFunc, glutTabletButtonFunc, glutJoystickFunc, glutDeviceGet

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSpaceballMotionFunc - sets the Spaceball motion callback for the current window.

SYNTAX

void glutSpaceballMotionFunc(void (*func)(int x, int y, int z));

ARGUMENTS

func The new entry callback function.

DESCRIPTION

glutSpaceballMotionFunc sets the Spaceball motion callback for the current window. The Spaceball motion callback for a window is called when the window has Spaceball input focus (normally, when the mouse is in the window) and the user generates Spaceball translations. The x, y, and z callback parameters indicate the translations along the X, Y, and Z axes. The callback parameters are normalized to be within the range of -1000 to 1000 inclusive.

Registering a Spaceball motion callback when a Spaceball device is not available has no effect and is not an error. In this case, no Spaceball motion callbacks will be generated.

Passing NULL to glutSpaceballMotionFunc disables the generation of Spaceball motion callbacks. When a new window is created, no Spaceball motion callback is initially registered.

SEE ALSO

glutSpaceballRotateFunc, glutSpaceballButtonFunc, glutMotionFunc, glutTabletMotionFunc, glutJoystick-Func, glutDeviceGet

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSpaceballRotateFunc - sets the Spaceball rotation callback for the current window.

SYNTAX

void glutSpaceballRotateFunc(void (*func)(int x, int y, int z));

ARGUMENTS

func The new spaceball rotate callback function.

DESCRIPTION

glutSpaceballRotateFunc sets the Spaceball rotate callback for the current window. The Spaceball rotate callback for a window is called when the window has Spaceball input focus (normally, when the mouse is in the window) and the user generates Spaceball rotations. The x, y, and z callback parameters indicate the rotation along the X, Y, and Z axes. The callback parameters are normalized to be within the range of -1800 to 1800 inclusive.

Registering a Spaceball rotate callback when a Spaceball device is not available is ineffectual and not an error. In this case, no Spaceball rotate callbacks will be generated.

Passing NULL to glutSpaceballRotateFunc disables the generation of Spaceball rotate callbacks. When a new window is created, no Spaceball rotate callback is initially registered.

SEE ALSO

glutSpaceballMotionFunc, glutSpaceballButtonFunc, glutMotionFunc, glutTabletMotionFunc, glutJoystick-Func, glutDeviceGet

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSpecialFunc - sets the special keyboard callback for the current window.

SYNTAX

void glutSpecialFunc(void (*func)(int key, int x, int y));

ARGUMENTS

func The new special callback function.

DESCRIPTION

glutSpecialFunc sets the special keyboard callback for the current window. The special keyboard callback is triggered when keyboard function or directional keys are pressed. The key callback parameter is a GLUT_KEY_* constant for the special key pressed. The x and y callback parameters indicate the mouse in window relative coordinates when the key was pressed. When a new window is created, no special callback is initially registered and special key strokes in the window are ignored. Passing NULL to glutSpecialFunc disables the generation of special callbacks.

During a special callback, glutGetModifiers may be called to determine the state of modifier keys when the keystroke generating the callback occurred.

An implementation should do its best to provide ways to generate all the GLUT_KEY_* special keys. The available GLUT_KEY_* values are:

GLUT_KEY_F1
F1 function key.

GLUT_KEY_F2
F2 function key.

GLUT_KEY_F3
F3 function key.

GLUT_KEY_F4
F4 function key.

GLUT_KEY_F5
F5 function key.

GLUT_KEY_F6
F6 function key.

GLUT_KEY_F7
F7 function key.

GLUT_KEY_F8
F8 function key.

GLUT_KEY_F9
F9 function key.

GLUT_KEY_F10
F10 function key.

GLUT_KEY_F11
F11 function key.

GLUT_KEY_F12
F12 function key.

GLUT_KEY_LEFT
Left directional key.

GLUT_KEY_UP
Up directional key.

GLUT_KEY_RIGHT
Right directional key.

GLUT_KEY_DOWN
Down directional key.

GLUT_KEY_PAGE_UP
Page up directional key.

GLUT_KEY_PAGE_DOWN
Page down directional key.

GLUT_KEY_HOME
Home directional key.

GLUT_KEY_END
End directional key.

GLUT_KEY_INSERT
Inset directional key.

Note that the escape, backspace, and delete keys are generated as an ASCII character.

SEE ALSO

glutSpecialUpFunc, glutKeyboardFunc, glutMouseFunc, glutSpaceballButtonFunc, glutButtonBoxFunc, glutTabletButtonFunc, glutGetModifiers

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSpecialUpFunc - sets the special keyboard up (key release) callback for the current window.

SYNTAX

void glutSpecialUpFunc(void (*func)(int key, int x, int y));

ARGUMENTS

func The new special callback function.

DESCRIPTION

glutSpecialUpFunc sets the special keyboard up (key release) callback for the current window. The special keyboard up callback is triggered when keyboard function or directional keys are released. The key callback parameter is a GLUT_KEY_* constant for the special key pressed. The x and y callback parameters indicate the mouse in window relative coordinates when the key was pressed. When a new window is created, no special up callback is initially registered and special key releases in the window are ignored. Passing NULL to glutSpecialUpFunc disables the generation of special up callbacks.

During a special up callback, glutGetModifiers may be called to determine the state of modifier keys when the key release generating the callback occurred.

To avoid the reporting of key release/press pairs due to auto repeat, use glutIgnoreKeyRepeat to ignore auto repeated keystrokes.

An implementation should do its best to provide ways to generate all the GLUT_KEY_* special keys. The available GLUT_KEY_* values are:

GLUT_KEY_F1
F1 function key.

GLUT_KEY_F2
F2 function key.

GLUT_KEY_F3
F3 function key.

GLUT_KEY_F4
F4 function key.

GLUT_KEY_F5
F5 function key.

GLUT_KEY_F6
F6 function key.

GLUT_KEY_F7
F7 function key.

GLUT_KEY_F8
F8 function key.

GLUT_KEY_F9
F9 function key.

GLUT_KEY_F10
F10 function key.

GLUT_KEY_F11
F11 function key.

GLUT_KEY_F12
F12 function key.

GLUT_KEY_LEFT
Left directional key.

GLUT_KEY_UP
Up directional key.

GLUT_KEY_RIGHT
Right directional key.

GLUT_KEY_DOWN
Down directional key.

GLUT_KEY_PAGE_UP
Page up directional key.

GLUT_KEY_PAGE_DOWN
Page down directional key.

GLUT_KEY_HOME
Home directional key.

GLUT_KEY_END
End directional key.

GLUT_KEY_INSERT
Inset directional key.

Note that the escape, backspace, and delete keys are generated as an ASCII character.

SEE ALSO

glutSpecialFunc, glutKeyboardFunc, glutKeyboardUpFunc, glutMouseFunc, glutSpaceballButtonFunc, glutButtonBoxFunc, glutTabletButtonFunc, glutGetModifiers, glutIgnoreKeyRepeat

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutStrokeCharacter - renders a stroke character using OpenGL.

SYNTAX

void glutStrokeCharacter(void *font, int character);

ARGUMENTS

font	Stroke font to use.
character	Character to render (not confined to 8 bits).

DESCRIPTION

Without using any display lists, glutStrokeCharacter renders the character in the named stroke font. The available fonts are:

GLUT_STROKE_ROMAN

A proportionally spaced Roman Simplex font for ASCII characters 32 through 127. The maximum top character in the font is 119.05 units; the bottom descends 33.33 units.

GLUT_STROKE_MONO_ROMAN

A mono-spaced spaced Roman Simplex font (same characters as GLUT_STROKE_ROMAN) for ASCII characters 32 through 127. The maximum top character in the font is 119.05 units; the bottom descends 33.33 units. Each character is 104.76 units wide.

Rendering a nonexistent character has no effect. A glTranslatef is used to translate the current model view matrix to advance the width of the character.

EXAMPLE

Here is a routine that shows how to render a string of ASCII text with glutStrokeCharacter:

```
void
output(GLfloat x, GLfloat y, char *text)
{
  char *p;

  glPushMatrix();
  glTranslatef(x, y, 0);
  for (p = text; *p; p++)
    glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
  glPopMatrix();
}
```

If you want to draw stroke font text using wide, antialiased lines, use:

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
glEnable(GL_LINE_SMOOTH);
glLineWidth(2.0);
output(200, 225, "This is antialiased.");
```

SEE ALSO

glutBitmapCharacter, glutStrokeWidth

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutStrokeWidth returns the width of a stroke character, glutStrokeLength returns the length of a stroke font string.

SYNTAX

int glutStrokeWidth(void *font, int character);
int glutStrokeLength(void *font, const unsigned char *string);

ARGUMENTS

font	Stroke font to use. For valid values, see the glutStrokeWidth description.
character	Character to return width of (not confined to 8 bits).
string	Text string (8-bit characters), nul terminated.

DESCRIPTION

glutStrokeWidth returns the width in modeling units of a stroke character in a supported stroke font. While the width of characters in a font may vary (though fixed width fonts do not vary), the maximum height characteristics of a particular font are fixed.

glutStrokeLength returns the length in modeling units of a string (8-bit characters). This length is equivalent to summing all the widths returned by glutStrokeWidth for each character in the string.

SEE ALSO

glutStrokeCharacter, glutBitmapWidth

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutSwapBuffers - swaps the buffers of the current window if double buffered.

SYNTAX

void glutSwapBuffers(void);

DESCRIPTION

Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called.

An implicit glFlush is done by glutSwapBuffers before it returns. Subsequent OpenGL commands can be issued immediately after calling glutSwapBuffers, but are not executed until the buffer exchange is completed.

If the layer in use is not double buffered, glutSwapBuffers has no effect.

SEE ALSO

glutPostRedisplay, glutDisplayFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutTabletButtonFunc - sets the special keyboard callback for the current window.

SYNTAX

```
void glutTabletButtonFunc(void (*func)(int button, int state,
                int x, int y));
```

ARGUMENTS

func The new tablet button callback function.

DESCRIPTION

glutTabletButtonFunc sets the tablet button callback for the current window. The tablet button callback for a window is called when the window has tablet input focus (normally, when the mouse is in the window) and the user generates tablet button presses. The button parameter will be the button number (starting at one). The number of available tablet buttons can be determined with glutDeviceGet(GLUT_NUM_TABLET_BUTTONS). The state is either GLUT_UP or GLUT_DOWN indicating whether the callback was due to a release or press respectively. The x and y callback parameters indicate the window relative coordinates when the tablet button state changed.

Registering a tablet button callback when a tablet device is not available is ineffectual and not an error. In this case, no tablet button callbacks will be generated.

Passing NULL to glutTabletButtonFunc disables the generation of tablet button callbacks. When a new window is created, no tablet button callback is initially registered.

SEE ALSO

glutTabletMotionFunc, glutDeviceGet, glutMotionFunc, glutSpaceballMotionFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutTabletMotionFunc - sets the special keyboard callback for the current window.

SYNTAX

void glutTabletMotionFunc(void (*func)(int x, int y));

ARGUMENTS

func The new entry callback function.

DESCRIPTION

glutTabletMotionFunc sets the tablet motion callback for the current window. The tablet motion callback for a window is called when the window has tablet input focus (normally, when the mouse is in the window) and the user generates tablet motion. The x and y callback parameters indicate the absolute position of the tablet ‘‘puck’’ on the tablet. The callback parameters are normalized to be within the range of 0 to 2000 inclusive.

Registering a tablet motion callback when a tablet device is not available is ineffectual and not an error. In this case, no tablet motion callbacks will be generated.

Passing NULL to glutTabletMotionFunc disables the generation of tablet motion callbacks. When a new window is created, no tablet motion callback is initially registered.

SEE ALSO

glutTabletButtonFunc, glutDeviceGet, glutMotionFunc, glutSpaceballMotionFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutTimerFunc - registers a timer callback to be triggered in a specified number of milliseconds.

SYNTAX

```
void glutTimerFunc(unsigned int msecs,
                   void (*func)(int value), value);
```

ARGUMENTS

msecs	Number of milliseconds to pass before calling the callback.
func	The timer callback function.
value	Integer value to pass to the timer callback.

DESCRIPTION

glutTimerFunc registers the timer callback func to be triggered in at least msecs milliseconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc. Multiple timer callbacks at same or differing times may be registered simultaneously.

The number of milliseconds is a lower bound on the time before the callback is generated. GLUT attempts to deliver the timer callback as soon as possible after the expiration of the callback's time interval.

There is no support for canceling a registered callback. Instead, ignore a callback based on its value parameter when it is triggered.

SEE ALSO

glutIdleFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutUseLayer - changes the layer in use for the current window.

SYNTAX

void glutUseLayer(GLenum layer);

ARGUMENTS

layer Either GLUT_NORMAL or GLUT_OVERLAY, selecting the normal plane or overlay respectively.

DESCRIPTION

glutUseLayer changes the per-window layer in use for the current window, selecting either the normal plane or overlay. The overlay should only be specified if an overlay exists, however windows without an overlay may still call glutUseLayer(GLUT_NORMAL). OpenGL commands for the window are directed to the current layer in use.

To query the layer in use for a window, call glutLayerGet(GLUT_LAYER_IN_USE).

SEE ALSO

glutEstablishOverlay, glutSetWindow

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutVideoResizeGet - retrieves GLUT video resize information represented by integers.

SYNTAX

int glutVideoResizeGet(GLenum param);

ARGUMENTS

param Name of video resize information to retrieve.

GLUT_VIDEO_RESIZE_POSSIBLE

Non-zero if video resizing is supported by the underlying system; zero if not supported. If this is zero, the other video resize GLUT calls do nothing when called. See the Implementation Notes sections below.

GLUT_VIDEO_RESIZE_IN_USE

GLUT_VIDEO_RESIZE_X_DELTA

GLUT_VIDEO_RESIZE_Y_DELTA

GLUT_VIDEO_RESIZE_WIDTH_DELTA

GLUT_VIDEO_RESIZE_HEIGHT_DELTA

GLUT_VIDEO_RESIZE_X

GLUT_VIDEO_RESIZE_Y

GLUT_VIDEO_RESIZE_WIDTH

GLUT_VIDEO_RESIZE_HEIGHT

DESCRIPTION

glutVideoResizeGet retrieves GLUT video resizing information represented by integers. The param parameter determines what type of video resize information to return.

X IMPLEMENTATION NOTES

The current implementation uses the SGIX_video_resize GLX extension. This extension is currently supported on SGI's InfiniteReality-based systems.

WIN32 IMPLEMENTATION NOTES

The current implementation never reports that video resizing is possible.

SEE ALSO

glutGet, glutSetupVideoResizing, glutStopVideoResizing, glutVideoResize, glutVideoPan

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutVisibilityFunc - sets the visibility callback for the current window.

SYNTAX

void glutVisibilityFunc(void (*func)(int state));

ARGUMENTS

func The new visibility callback function.

DESCRIPTION

glutVisibilityFunc sets the visibility callback for the current window. The visibility callback for a window is called when the visibility of a window changes. The state callback parameter is either GLUT_NOT_VISIBLE or GLUT_VISIBLE depending on the current visibility of the window. GLUT_VISIBLE does not distinguish a window being totally versus partially visible. GLUT_NOT_VISIBLE means no part of the window is visible, i.e., until the window's visibility changes, all further rendering to the window is discarded.

GLUT considers a window visible if any pixel of the window is visible or any pixel of any descendant window is visible on the screen.

Passing NULL to glutVisibilityFunc disables the generation of the visibility callback.

If the visibility callback for a window is disabled and later re-enabled, the visibility status of the window is undefined; any change in window visibility will be reported, that is if you disable a visibility callback and re-enable the callback, you are guaranteed the next visibility change will be reported.

Setting the visibility callback for a window disables the window status callback set for the window (and vice versa). The window status callback is set with glutWindowStatusFunc. glutVisibilityFunc is deprecated in favor of the more informative glutWindowStatusFunc.

SEE ALSO

glutCreateWindow, glutPopWindow, glutWindowStatusFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutWarpPointer warps the pointer's location.

SYNTAX

void glutWarpPointer(int x, int y);

ARGUMENTS

x	X offset relative to the current window's origin (upper left).
y	Y offset relative to the current window's origin (upper left).

DESCRIPTION

glutWarpPointer warps the window system's pointer to a new location relative to the origin of the current window. The new location will be offset *x* pixels on the X axis and *y* pixels on the Y axis. These parameters may be negative. The warp is done immediately.

If the pointer would be warped outside the screen's frame buffer region, the location will be clamped to the nearest screen edge. The window system is allowed to further constrain the pointer's location in window system dependent ways.

The following is good advice that applies to glutWarpPointer: ''There is seldom any reason for calling this function. The pointer should normally be left to the user.'' (from Xlib's XWarpPointer man page.)

SEE ALSO

glutMouseFunc, glutMotionFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)

NAME

glutWindowStatusFunc - sets the window status callback for the current window.

SYNTAX

void glutWindowStatusFunc(void (*func)(int state));

ARGUMENTS

func The new window status callback function.

DESCRIPTION

glutWindowStatusFunc sets the window status callback for the current window. The window status callback for a window is called when the window status (visibility) of a window changes. The state callback parameter is one of GLUT_HIDDEN, GLUT_FULLY_RETAINED, GLUT_PARTIALLY_RETAINED, or GLUT_FULLY_COVERED depending on the current window status of the window. GLUT_HIDDEN means that the window is either not shown (often meaning that the window is iconified). GLUT_FULLY_RETAINED means that the window is fully retained (no pixels belonging to the window are covered by other windows). GLUT_PARTIALLY_RETAINED means that the window is partially retained (some but not all pixels belonging to the window are covered by other windows). GLUT_FULLY_COVERED means the window is shown but no part of the window is visible, i.e., until the window's status changes, all further rendering to the window is discarded.

GLUT considers a window visible if any pixel of the window is visible or any pixel of any descendant window is visible on the screen.

GLUT applications are encouraged to disable rendering and/or animation when windows have a status of either GLUT_HIDDEN or GLUT_FULLY_COVERED.

Passing NULL to glutWindowStatusFunc disables the generation of the window status callback.

If the window status callback for a window is disabled and later re-enabled, the window status of the window is undefined; any change in window window status will be reported, that is if you disable a window status callback and re-enable the callback, you are guaranteed the next window status change will be reported.

Setting the window status callback for a window disables the visibility callback set for the window (and vice versa). The visibility callback is set with glutVisibilityFunc. glutVisibilityFunc is deprecated in favor of the more informative glutWindowStatusFunc.

SEE ALSO

glutCreateWindow, glutPopWindow, glutVisibilityFunc

AUTHOR

Mark J. Kilgard (mjk@nvidia.com)