The OpenGL Utility Toolkit (GLUT) Programming Interface

API Version 3

Mark J. Kilgard
Silicon Graphics, Inc.

February 23, 1996

Contents

1 Introduction

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.
- Sophisticated input devices.
- 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.

An ANSI C implementation of GLUT for the X Window System [13] has been implemented by the author. Windows NT and OS/2 versions of GLUT are also available.

This documentation serves as both a specification and a programming guide. If you are interested in a brief introduction to programming with GLUT, look for the introductory OpenGL column [8] published in *The X Journal*.

The remainder of this section describes GLUT's design philosophy and usage model. The following sections specify the GLUT routines, grouped by functionality. The final sections discuss usage advice, the FORTRAN binding, and implementation issues. Appendix A enumerates and annotates the logical programmer visible state maintained by GLUT. Appendix B presents the ANSI C GLUT API via its header file. Appendix C presents the FORTRAN GLUT API via its header file.

1.1 Background

One of the major accomplishments in the specification of OpenGL [14, 10] 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 [3] describes the standard by which OpenGL interacts with the X Window System.

The predecessor to OpenGL is IRIS GL [15, 16]. 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 [6] or Motif [7] 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* [9]. 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.

1.2 Design Philosophy

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.

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.

1.3 API Version 2

In response to feedback from the original version of GLUT, GLUT API version 2 was developed. Additions to the original GLUT API version 1 are:

- Support for requesting stereo and multisample windows.
- New routines to query support for and provide callbacks for sophisticated input devices: the Spaceball, tablet, and dial & button box.
- New routine to register a callback for keyboard function and directional keys. In version 1, only ASCII characters could be generated.
- New queries for stereo, multisampling, and elapsed time.
- New routine to ease querying for OpenGL extension support.

GLUT API version 2 is completely compatible with version 1 of the API.

1.4 API Version 3

Further feedback lead to the development of GLUT API version 3. Additions to the GLUT API version 2 are:

- The `glutMenuStateFunc` has been deprecated in favor of the `glutMenuStatusFunc`.
- `glutFullScreen` requests full screen top-level windows.
- Three additional Helvetica bitmap fonts.
- Implementations should enforce not allowing any modifications to menus while menus are in use.
- `glutBitmapWidth` and `glutStrokeBitmap` return the widths of individual characters.
- `glutGetModifiers` called during a keyboard, mouse, or special callback returns the modifiers (Shift, Ctrl, Alt) held down when the mouse or keyboard event was generated.
- Access to per-window transparent overlays when overlay hardware is supported. The routines added are `glutEstablishOverlay`, `glutRemoveOverlay`, `glutShowOverlay`, `glutHideOverlay`, `glutUseOverlay`, `glutLayerGet`, and `glutPostOverlayRedisplay`.
- A new display mode called `GLUT_LUMINANCE` using OpenGL's RGBA color model, but that has no green or blue components. The red component is converted to an index and looked up in a writable colormap to determine displayed colors. See `glutInitDisplayMode`.

GLUT API version 3 should be largely compatible with version 2. Be aware that programs that used to (through some degree of fortuitous timing) modify menus while menus are in use will encounter fatal errors when doing so in version 3.

Another change in GLUT 3.0 that may require source code modification to pre-3.0 GLUT programs. GLUT 3.0 no longer lets a window be shown without a display callback registered. This change makes sure windows are not displayed on the screen without the GLUT application providing a way for them to be rendered. In conjunction with this change, `glutDisplayFunc` no longer allows `NULL` to deregister a display callback. While there is no longer a way to deregister a display callback, you can still change the change the display callback routine with subsequent calls to `glutDisplayFunc`.

The display mode mask parameter for `glutInitDisplayMode` and the milliseconds parameter for `glutTimerFunc` are now of type `unsigned int` (previously `unsigned long`).

1.5 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
```

1.6 Terminology

A number of terms are used in a GLUT-specific manner throughout this document. The GLUT meaning of these terms is independent of the window system GLUT is used with. Here are GLUT-specific meanings for the following GLUT-specific terms:

Callback A programmer specified routine that can be registered with GLUT to be called in response to a specific type of event. Also used to refer to a specific callback routine being called.

Colormap A mapping of pixel values to RGB color values. Use by color index windows.

Dials and button box A sophisticated input device consisting of a pad of buttons and an array of rotating dials, often used by computer-aided design programs.

Display mode A set of OpenGL frame buffer capabilities that can be attributed to a window.

Idle A state when no window system events are received for processing as callbacks and the idle callback, if one is registered, is called.

Layer in use Either the normal plane or overlay. This per-window state determines what frame buffer layer OpenGL commands affect.

Menu entry A menu item that the user can select to trigger the menu callback for the menu entry's value.

Menu item Either a menu entry or a sub-menu trigger.

Modifiers The Shift, Ctrl, and Alt keys that can be held down simultaneously with a key or mouse button being pressed or released.

Multisampling A technique for hardware antialiasing generally available only on expensive 3D graphics hardware [1]. Each pixel is composed of a number of samples (each containing color and depth information). The samples are averaged to determine the displayed pixel color value. Multisampling is supported as an extension to OpenGL.

Normal plane The default frame buffer layer where GLUT window state resides; as opposed to the *overlay*.

Overlay A frame buffer layer that can be displayed preferentially to the *normal plane* and supports transparency to display through to the *normal plane*. Overlays are useful for rubber-banding effects, text annotation, and other operations, to avoid damaging the normal plane frame buffer state. Overlays require hardware support not present on all systems.

Pop The act of forcing a window to the top of the stacking order for sibling windows.

Pop-up menu A menu that can be set to appear when a specified mouse button is pressed in a window. A pop-menu consists of multiple menu items.

Push The act of forcing a window to the bottom of the stacking order for sibling windows.

Reshape The act of changing the size or shape of the window.

Spaceball A sophisticated 3D input device that provides six degrees of freedom, three axes of rotation and three axes of translation. It also supports a number of buttons. The device is a hand-sized ball attached to a base. By cupping the ball with one's hand and applying torsional or directional force on the ball, rotations and translationsare generated.

Stereo A frame buffer capability providing left and right color buffers for creating stereoscopic renderings. Typically, the user wears LCD shuttered goggles synchronized with the alternating display on the screen of the left and right color buffers.

Sub-menu A menu cascaded from some sub-menu trigger.

Sub-menu trigger A menu item that the user can enter to cascade another pop-up menu.

Subwindow A type of window that is the child window of a top-level window or other subwindow. The drawing and visible region of a subwindow is limited by its parent window.

Tablet A precise 2D input device. Like a mouse, 2D coordinates are returned. The absolute position of the tablet "puck" on the tablet is returned. Tablets also support a number of buttons.

Timer A callback that can be scheduled to be called in a specified interval of time.

Top-level window A window that can be placed, moved, resized, etc. independently from other top-level windows by the user. Subwindows may reside within a top-level window.

Window A rectangular area for OpenGL rendering.

Window display state One of shown, hidden, or iconified. A shown window is potentially visible on the screen (it may be obscured by other windows and not actually visible). A hidden window will never be visible. An iconified window is not visible but could be made visible in response to some user action like clicking on the window's corresponding icon.

Window system A broad notion that refers to both the mechanism and policy of the window system. For example, in the X Window System both the window manager and the X server are integral to what GLUT considers the window system.

2 Initialization

Routines beginning with the `glutInit-` prefix are used to initialize GLUT state. The primary initialization routine is `glutInit` that should only be called exactly once in a GLUT program. No non-`glutInit-` prefixed GLUT or OpenGL routines should be called before `glutInit`.

The other `glutInit-` routines may be called before `glutInit`. The reason is these routines can be used to set default window initialization state that might be modified by the command processing done in `glutInit`. For example, `glutInitWindowSize(400, 400)` can be called before `glutInit` to indicate 400 by 400 is the program's default window size. Setting the *initial window size* or *position* before `glutInit` allows the GLUT program user to specify the initial size or position using command line arguments.

2.1 glutInit

`glutInit` is used to initialize the GLUT library.

Usage

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

`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` ***W*x*H*+*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.

2.2 glutInitWindowPosition, glutInitWindowSize

`glutInitWindowPosition` and `glutInitWindowSize` set the *initial window position* and *size* respectively.

Usage

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

`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.

2.3 glutInitDisplayMode

`glutInitDisplayMode` sets the *initial display mode*.

Usage

```
void glutInitDisplayMode(unsigned int mode);
```

`mode` Display mode, normally the bitwise *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 select a window with an accumulation buffer.

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

`GLUT_DEPTH` Bit mask to select a window with a depth buffer.

`GLUT_STENCIL` Bit mask to select a window with a stencil buffer.

`GLUT_MULTISAMPLE` Bit mask to select 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.

GLUT_LUMINANCE Implementation Notes

GLUT_LUMINANCE is not supported on most OpenGL platforms.

3 Beginning Event Processing

After a GLUT program has done initial setup such as creating windows and menus, GLUT programs enter the GLUT event processing loop by calling glutMainLoop.

3.1 glutMainLoop

glutMainLoop enters the GLUT event processing loop.

Usage

```
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.

4 Window Management

GLUT supports two types of windows: top-level windows and subwindows. Both types support OpenGL rendering and GLUT callbacks. There is a single identifier space for both types of windows.

4.1 glutCreateWindow

glutCreateWindow creates a top-level window.

Usage

```
int glutCreateWindow(char *name);
```

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.

4.2 glutCreateSubWindow

`glutCreateSubWindow` creates a subwindow.

Usage

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

`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.

4.3 glutSetWindow, glutGetWindow

`glutSetWindow` sets the *current window*; `glutGetWindow` returns the identifier of the *current window*.

Usage

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

`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`.

4.4 glutDestroyWindow

`glutDestroyWindow` destroys the specified window.

Usage

```
void glutDestroyWindow(int win);
```

`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).

4.5 glutPostRedisplay

`glutPostRedisplay` marks the *current window* as needing to be redisplayed.

Usage

```
void glutPostRedisplay(void);
```

Description

Mark the normal plane of *current window* as needing to be redisplayed. 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)`.

Also, see `glutPostOverlayRedisplay`.

4.6 glutSwapBuffers

`glutSwapBuffers` swaps the buffers of the *current window* if double buffered.

Usage

```
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.

4.7 glutPositionWindow

`glutPositionWindow` requests a change to the position of the *current window*.

Usage

```
void glutPositionWindow(int x, int y);
```

`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.

4.8 glutReshapeWindow

`glutReshapeWindow` requests a change to the size of the *current window*.

Usage

```
void glutReshapeWindow(int width, int height);
```

`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.

4.9 glutFullScreen

`glutFullScreen` requests that the *current window* be made full screen.

Usage

```
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`.

4.10 glutPopWindow, glutPushWindow

`glutPopWindow` and `glutPushWindow` change the stacking order of the *current window* relative to its siblings.

Usage

```
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.

4.11 glutShowWindow, glutHideWindow, glutIconifyWindow

`glutShowWindow`, `glutHideWindow`, and `glutIconifyWindow` change the display status of the *current window*.

Usage

```
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.

4.12 glutSetWindowTitle, glutSetIconTitle

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

Usage

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

`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.

4.13 glutSetCursor

`glutSetCursor` changes the cursor image of the *current window*.

Usage

```
void glutSetCursor(int cursor);
```

`cursor` Name of cursor image to change to.

`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 [4] to access a full-screen cross-hair cursor if possible.

5 Overlay Management

When overlay hardware is available, GLUT provides a set of routine for establishing, using, and removing an overlay for GLUT windows. When an overlay is established, a separate OpenGL context is also established. A window's overlay OpenGL state is kept distinct from the normal planes OpenGL state.

5.1 glutEstablishOverlay

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

Usage

```
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.

X Implementation Notes

GLUT for X uses the `SERVER_OVERLAY_VISUALS` convention [5] 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.

5.2 glutUseLayer

`glutUseLayer` changes the *layer in use* for the *current window*.

Usage

```
void glutUseLayer(GLenum layer);
```

`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)`.

5.3 glutRemoveOverlay

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

Usage

```
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.

5.4 glutPostOverlayRedisplay

`glutPostOverlayRedisplay` marks the overlay of the *current window* as needing to be redisplayed.

Usage

```
void glutPostOverlayRedisplay(void);
```

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)`.

Also, see `glutPostRedisplay`.

5.5 glutShowOverlay, glutHideOverlay

`glutShowOverlay` shows the overlay of the *current window*; `glutHideOverlay` hides the overlay.

Usage

```
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.

6 Menu Management

GLUT supports simple cascading pop-up menus. They are designed to let a user select various modes within a program. The functionality is simple and minimalistic and is meant to be that way. Do not mistake GLUT's pop-up menu facility with an attempt to create a full-featured user interface.

It is illegal to create or destroy menus, or change, add, or remove menu items while a menu (and any cascaded sub-menus) are in use (that is, popped up).

6.1 glutCreateMenu

`glutCreateMenu` creates a new pop-up menu.

Usage

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

`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.

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 [5] is used to determine if overlay visuals are available.

6.2 glutSetMenu, glutGetMenu

`glutSetMenu` sets the *current menu*; `glutGetMenu` returns the identifier of the *current menu*.

Usage

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

`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.

6.3 glutDestroyMenu

`glutDestroyMenu` destroys the specified menu.

Usage

```
void glutDestroyMenu(int menu);
```

`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.

6.4 glutAddMenuEntry

`glutAddMenuEntry` adds a menu entry to the bottom of the *current menu*.

Usage

```
void glutAddMenuEntry(char *name, int value);
```

`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.

6.5 glutAddSubMenu

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

Usage

```
void glutAddSubMenu(char *name, int menu);
```

`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.

6.6 glutChangeToMenuEntry

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

Usage

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

`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.

6.7 glutChangeToSubMenu

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

Usage

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

`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.

6.8 glutRemoveMenuItem

glutRemoveMenuItem remove the specified menu item.

Usage

```
void glutRemoveMenuItem(int entry);
```

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.

6.9 glutAttachMenu, glutDetachMenu

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*.

Usage

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

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.

7 Callback Registration

GLUT supports a number of callbacks to respond to events. There are three types of callbacks: window, menu, and global. Window callbacks indicate when to redisplay or reshape a window, when the visibility of the window changes, and when input is available for the window. The menu callback is set by the glutCreateMenu call described already. The global callbacks manage the passing of time and menu usage. The calling order of callbacks between different windows is undefined.

Callbacks for input events should be delivered to the window the event occurs in. Events should not propagate to parent windows.

X Implementation Notes

The X GLUT implementation uses the X Input extension [11, 12] to support sophisticated input devices: Spaceball, dial & button box, and digitizing tablet. Because the X Input extension does not mandate how particular types of devices are advertised through the extension, it is possible GLUT for X may not correctly support input devices that would otherwise be of the correct type. The X GLUT implementation will support the Silicon Graphics Spaceball, dial & button box, and digitizing tablet as advertised through the X Input extension.

7.1 glutDisplayFunc

`glutDisplayFunc` sets the display callback for the *current window*.

Usage

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

`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.

7.2 glutOverlayDisplayFunc

`glutOverlayDisplayFunc` sets the overlay display callback for the *current window*.

Usage

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

`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.

7.3 glutReshapeFunc

`glutReshapeFunc` sets the reshape callback for the *current window*.

Usage

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

`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`.

7.4 glutKeyboardFunc

`glutKeyboardFunc` sets the keyboard callback for the *current window*.

Usage

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

`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.

Also, see `glutSpecialFunc` for a means to detect non-ASCII key strokes.

7.5 glutMouseFunc

`glutMouseFunc` sets the mouse callback for the *current window*.

Usage

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

`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.

7.6 glutMotionFunc, glutPassiveMotionFunc

`glutMotionFunc` and `glutPassiveMotionFunc` set the motion and passive motion callbacks respectively for the *current window*.

Usage

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

`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.

7.7 glutVisibilityFunc

`glutVisibilityFunc` sets the visibility callback for the *current window*.

Usage

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

`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.

7.8 glutEntryFunc

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

Usage

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

`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.

7.9 glutSpecialFunc

`glutSpecialFunc` sets the special keyboard callback for the *current window*.

Usage

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

`func` The new entry 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.

7.10 glutSpaceballMotionFunc

`glutSpaceballMotionFunc` sets the Spaceball motion callback for the *current window*.

Usage

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

`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.

7.11 glutSpaceballRotateFunc

`glutSpaceballRotateFunc` sets the Spaceball rotation callback for the *current window*.

Usage

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

`func` The new entry 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.

7.12 glutSpaceballButtonFunc

`glutSpaceballButtonFunc` sets the Spaceball button callback for the *current window*.

Usage

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

`func` The new entry 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.

7.13 glutButtonBoxFunc

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

Usage

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

func The new entry 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.

7.14 glutDialsFunc

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

Usage

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

func The new entry 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.

7.15 glutTabletMotionFunc

`glutTabletMotionFunc` sets the special keyboard callback for the *current window*.

Usage

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

`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.

7.16 glutTabletButtonFunc

`glutTabletButtonFunc` sets the special keyboard callback for the *current window*.

Usage

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

`func` The new entry 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.

7.17 glutMenuStatusFunc

`glutMenuStatusFunc` sets the global menu status callback.

Usage

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

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.

7.18 glutIdleFunc

`glutIdleFunc` sets the global idle callback.

Usage

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

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.

7.19 glutTimerFunc

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

Usage

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

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.

8 Color Index Colormap Management

OpenGL supports both RGBA and color index rendering. The RGBA mode is generally preferable to color index because more OpenGL rendering capabilities are available and color index mode requires the loading of colormap entries.

The GLUT color index routines are used to write and read entries in a window's color index colormap. Every GLUT color index window has its own logical color index colormap. The size of a window's colormap can be determined by calling `glutGet(GLUT_WINDOW_COLORMAP_SIZE)`.

GLUT color index windows within a program can attempt to share colormap resources by copying a single color index colormap to multiple windows using `glutCopyColormap`. If possible GLUT will attempt to share the actual colormap. While copying colormaps using `glutCopyColormap` can potentially allow sharing of physical colormap resources, logically each window has its own colormap. So changing a copied colormap of a window will force the duplication of the colormap. For this reason, color index programs should generally load a single color index colormap, copy it to all color index windows within the program, and then not modify any colormap cells.

Use of multiple colormaps is likely to result in colormap installation problems where some windows are displayed with an incorrect colormap due to limitations on colormap resources.

8.1 glutSetColor

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

Usage

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

`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.

8.2 glutGetColor

`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*.

Usage

```
GLfloat glutGetColor(int cell, int component);
```

`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.

8.3 glutCopyColormap

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

Usage

```
void glutCopyColormap(int win);
```

`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.

9 State Retrieval

GLUT maintains a considerable amount of programmer visible state. Some (but not all) of this state may be directly retrieved.

9.1 glutGet

`glutGet` retrieves simple GLUT state represented by integers.

Usage

```
int glutGet(GLenum state);
```

`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 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 window*'s stencil buffer.

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

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

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

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

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

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

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

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

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

`GLUT_WINDOW_DOUBLEBUFFER` One if the *current window* is double buffered, zero otherwise.

`GLUT_WINDOW_RGBA` One if 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 window*'s color index colormap; zero for RGBA color model windows.

`GLUT_WINDOW_NUM_SAMPLES` Number of samples for multisampling for the *current window*.

`GLUT_WINDOW_STEREO` One if 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)`).

Description

`glutGet` retrieves simple GLUT state represented by integers. The `state` parameter determines what type of state to return. 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.

9.2 glutLayerGet

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

Usage

```
int glutLayerGet(GLenum info);
```

`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.

9.3 glutDeviceGet

`glutDeviceGet` retrieves GLUT device information represented by integers.

Usage

```
int glutDeviceGet(GLenum info);
```

`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.

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.

9.4 glutGetModifiers

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

Usage

```
int glutGetModifiers(void);
```

`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.

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`.

9.5 glutExtensionSupported

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

Usage

```
int glutExtensionSupported(char *extension);
```

`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.

10 Font Rendering

GLUT supports two type of font rendering: stroke fonts, meaning each character is rendered as a set of line segments; and bitmap fonts, where each character is a bitmap generated with glBitmap. Stroke fonts have the advantage that because they are geometry, they can be arbitrarily scale and rendered. Bitmap fonts are less flexible since they are rendered as bitmaps but are usually faster than stroke fonts.

10.1 glutBitmapCharacter

glutBitmapCharacter renders a bitmap character using OpenGL.

Usage

```
void glutBitmapCharacter(void *font, int character);
```

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.

10.2 glutBitmapWidth

`glutBitmapWidth` returns the width of a bitmap character.

Usage

```
int glutBitmapWidth(GLUTbitmapFont font, int character)
```

`font` Bitmap font to use.

`character` Character to return width of (not confined to 8 bits).

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.

10.3 glutStrokeCharacter

`glutStrokeCharacter` renders a stroke character using OpenGL.

Usage

```
void glutStrokeCharacter(void *font, int character);
```

`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.

10.4 glutStrokeWidth

`glutStrokeWidth` returns the width of a stroke character.

Usage

```
int glutStrokeWidth(GLUTstrokeFont font, int character)
```

`font` Stroke font to use.

`character` Character to return width of (not confined to 8 bits).

Description

`glutStrokeWidth` returns the width in pixels 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.

11 Geometric Object Rendering

GLUT includes a number of routines for generating easily recognizable 3D geometric objects. These routines reflect functionality available in the `aux` toolkit described in the *OpenGL Programmer's Guide*

and are included in GLUT to allow the construction of simple GLUT programs that render recognizable objects. These routines can be implemented as pure OpenGL rendering routines. The routines do *not* generate display lists for the objects they create.

The routines generate normals appropriate for lighting but do not generate texture coordinates (except for the teapot).

11.1 glutSolidSphere, glutWireSphere

`glutSolidSphere` and `glutWireSphere` render a solid or wireframe sphere respectively.

Usage

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

`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.

11.2 glutSolidCube, glutWireCube

`glutSolidCube` and `glutWireCube` render a solid or wireframe cube respectively.

Usage

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

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`.

11.3 glutSolidCone, glutWireCone

`glutSolidCone` and `glutWireCone` render a solid or wireframe cone respectively.

Usage

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

`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.

11.4 glutSolidTorus, glutWireTorus

`glutSolidTorus` and `glutWireTorus` render a solid or wireframe torus (doughnut) respectively.

Usage

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

`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.

11.5 glutSolidDodecahedron, glutWireDodecahedron

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

Usage

```
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 $\sqrt{3}$.

11.6 glutSolidOctahedron, glutWireOctahedron

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

Usage

```
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.

11.7 glutSolidTetrahedron, glutWireTetrahedron

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

Usage

```
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 $\sqrt{3}$.

11.8 glutSolidIcosahedron, glutWireIcosahedron

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

Usage

```
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.

11.9 glutSolidTeapot, glutWireTeapot

`glutSolidTeapot` and `glutWireTeapot` render a solid or wireframe teapot[1] respectively.

Usage

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

`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.

12 Usage Advice

There are a number of points to keep in mind when writing GLUT programs. Some of these are strong recommendations, others simply hints and tips.

- Do not change state that will affect the way a window will be drawn in a window's display callback. Your display callbacks should be idempotent.
- If you need to redisplay a window, instead of rendering in whatever callback you happen to be in, call `glutPostRedisplay` (or `glutPostRedisplay` for overlays). As a general rule, the only code that renders directly to the screen should be in called from display callbacks; other types of callbacks should not be rendering to the screen.
- If you use an idle callback to control your animation, use the visibility callbacks to determine when the window is fully obscured or iconified to determine when not to waste processor time rendering.
- Neither GLUT nor the window system automatically reshape sub-windows. If subwindows should be reshaped to reflect a reshaping of the top-level window, the GLUT program is responsible for doing this.
- Avoid using color index mode if possible. The RGBA color model is more functional, and it is less likely to cause colormap swapping effects.
- Do not call any GLUT routine that affects the *current window* or *current menu* if there is no *current window* or *current menu* defined. This can be the case at initialization time (before any windows or menus have been created) or if your destroy the *current window* or *current menu*. GLUT implementations are not obliged to generate a warning because doing so would slow down the operation of every such routine to first make sure there was a *current window* or *current menu*.
- For most callbacks, the *current window* and/or *current menu* is set appropriately at the time of the callback. Timer and idle callbacks are exceptions. If your application uses multiple windows or menus, make sure you explicitly you set the *current window* or *menu* appropriately using `glutSetWindow` or `glutSetMenu` in the idle and timer callbacks.
- If you register a single function as a callback routine for multiple windows, you can call `glutGetWindow` within the callback to determine what window generated the callback. Likewise, `glutGetMenu` can be called to determine what menu.
- By default, timer and idle callbacks may be called while a pop-up menu is active. On slow machines, slow rendering in an idle callback may compromise menu performance. Also, it may be desirable for motion to stop immediately when a menu is triggered. In this case, use the menu entry/exit callback set with `glutMenuStateFunc` to track the usage of pop-up menus.

[1] Yes, the *classic* computer graphics teapot modeled by Martin Newell in 1975 [2].

- Do not select for more input callbacks than you actually need. For example, if you do not need motion or passive motion callbacks, disable them by passing `NULL` to their callback register functions. Disabling input callbacks allows the GLUT implementation to limit the window system input events that must be processed.
- Not every OpenGL implementation supports the same range of frame buffer capabilities, though minimum requirements for frame buffer capabilities do exist. If `glutCreateWindow` or `glutCreateSubWindow` are called with an *initial display mode* not supported by the OpenGL implementation, a fatal error will be generated with an explanatory message. To avoid this, `glutGet(GLUT_DISPLAY_MODE_POSSIBLE)` should be called to determine if the *initial display mode* is supported by the OpenGL implementation.
- The Backspace, Delete, and Escape keys generate ASCII characters, so detect these key presses with the `glutKeyboardFunc` callback, not with the `glutSpecialFunc` callback.
- Keep in mind that when a window is damaged, you should assume *all* of the ancillary buffers are damaged and redraw them all.
- Keep in mind that after a `glutSwapBuffers`, you should assume the state of the back buffer becomes undefined.
- If not using `glutSwapBuffers` for double buffered animation, remember to use `glFlush` to make sure rendering requests are dispatched to the frame buffer. While many OpenGL implementations will automatically flush pending commands, this is specifically not mandated.
- Remember that it is illegal to create or destroy menus or change, add, or remove menu items while a menu (and any cascaded sub-menus) are in use (that is, "popped up"). Use the menu status callback to know when to avoid menu manipulation.
- It is more efficient to use `glutHideOverlay` and `glutShowOverlay` to control the display state of a window's overlay instead of removing and re-establishing an overlay every time an overlay is needed.
- Few workstations have support for multiple simultaneously installed overlay colormaps. For this reason, if an overlay is cleared or otherwise not be used, it is best to hide it using `glutHideOverlay` to avoid other windows with active overlays from being displayed with the wrong colormap. If your application uses multiple overlays, use `glutCopyColormap` to promote colormap sharing.
- If you are encountering GLUT warnings or fatal errors in your programs, try setting a debugger breakpoint in `__glutWarning` or `__glutFatalError` (though these names are potentially implementation dependent) to determine where within your program the error occurred.
- GLUT has no special routine for exiting the program. GLUT programs should use ANSI C's `exit` routine. If a program needs to perform special operations before quitting the program, use the ANSI C `onexit` routine to register exit callbacks. GLUT will exit the program unilaterally when fatal errors occur or when the window system requests the program to terminate. For this reason, avoid calling any GLUT routines within an exit callback.
- Definitely, definitely, use the `-gldebug` option to look for OpenGL errors when OpenGL rendering does not appear to be operating properly. OpenGL errors are only reported if you explicitly look for them!

13 FORTRAN Binding

All GLUT functionality is available through the GLUT FORTRAN API. The GLUT FORTRAN binding is intended to be used in conjunction with the OpenGL and GLU FORTRAN APIs.

A FORTRAN routine using GLUT routines should include the GLUT FORTRAN header file. While this is potentially system dependent, on Unix systems this is normally done by including after the `SUBROUTINE`, `FUNCTION`, or `PROGRAM` line:

```
#include "GL/fglut.h"
```

Though the FORTRAN 77 specification differentiates identifiers by their first six characters only, the GLUT FORTRAN binding (and the OpenGL and GLU FORTRAN bindings) assume identifiers are not limited to 6 characters.

The FORTRAN GLUT binding library archive is typically named `libfglut.a` on Unix systems. FORTRAN GLUT programs need to link with the system's OpenGL and GLUT libraries and the respective Fortran binding 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 for a GLUT FORTRAN program would look like:

```
f77 -o foo foo.c -lfglut -lglut -lfGLU -lGLU -lfGL -lGL \
  -lXmu -lXi -lXext -lX11 -lm
```

13.1 Names for the FORTRAN GLUT Binding

Allowing for FORTRAN's case-insensitivity, the GLUT FORTRAN binding constant and routine names are the same as the C binding's names.

The OpenGL Architectural Review Board (ARB) official OpenGL FORTRAN API prefixes every routine and constant with the letter F. The justification was to avoid name space collisions with the C names in anachronistic compilers. Nearly all modern FORTRAN compilers avoid these name space clashes via other means (underbar suffixing of FORTRAN routines is used by most Unix FORTRAN compilers).

The GLUT FORTRAN API does *not* use such prefixing conventions because of the documentation and coding confusion introduced by such prefixes. The confusion is heightened by FORTRAN's default implicit variable initialization so programmers may realize the lack of a constant prefix as a result of a run-time error. The confusion introduced to support the prefixes was not deemed worthwhile simply to support anachronistic compliers.

The OpenGL ARB's official FORTRAN binding for the GLU and OpenGL APIs that chose to prefix the letter `F` before every OpenGL and GLU constant and routine name.

13.2 Font Naming Caveat

Because GLUT fonts are compiled directly into GLUT programs as data, and programs should only have the fonts compiled into them that they use, GLUT font names like `GLUT_BITMAP_TIMES_ROMAN_24` are really symbols so the linker should only pull in used fonts.

Unfortunately, because some supposedly modern FORTRAN compilers link declared but unused data `EXTERNAL`s, "GL/fglut.h" does not explicitly declare `EXTERNAL` the GLUT font symbols. Declaring the GLUT font symbols as `EXTERNAL` risks forcing every GLUT FORTRAN program to contain the data for every GLUT font. GLUT Fortran programmers should explicitly declare `EXTERNAL` the GLUT fonts they use. Example:

```
        SUBROUTINE PRINTA
#include "GL/fglut.h"
        EXTERNAL GLUT_BITMAP_TIMES_ROMAN_24
        CALL glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 65)
        END
```

13.3 NULL Callback

FORTRAN does not support passing `NULL` as a callback parameter the way ANSI C does. For this reason, `GLUTNULL` is used in place of `NULL` in GLUT FORTRAN programs to indicate a `NULL` callback.

14 Implementation Issues

While this specification is primarily intended to describe the GLUT API and not its implementation, the section describes implementation issues that are likely to help both GLUT implementors properly implement GLUT and provide GLUT programmers with information to better utilize GLUT.

14.1 Name Space Conventions

The GLUT implementation should have a well-defined name space for both exported symbols and visible, but not purposefully exported symbols. All exported functions are prefixed by `glut`. All exported macro definitions are prefixed by `GLUT_`. No data symbols are exported. All internal symbols that might be user-visible but not intended to be exported should be prefixed by `__glut`. Users of the GLUT API should *not* use any `__glut` prefixed symbols.

14.2 Modular Implementation

It is often the case that windowing libraries tend to result in large, bulky programs because a large measure of "dynamically dead" code is linked into the programs because it can not be determined at link time that the program will never require (that is, execute) the code. A consideration (not a primary one though) in GLUT's API design is make the API modular enough that programs using a limited subset of GLUT's API can minimize the portion of the GLUT library implementation required. This does assume the implementation of GLUT is structured to take advantage of the API's modularity.

A good implementation can be structured so significant chunks of code for color index colormap management, non-standard device support (Spaceball, dial & button box, and tablet), overlay management, pop-up menus, miscellaneous window management routines (pop, push, show, hide, full-screen, iconify), geometric shape rendering, and font rendering only need to be pulled into GLUT programs when the interface to this functionality is explicitly used by the GLUT program.

14.3 Error Checking and Reporting

How errors and warnings about improper GLUT usage are reported to GLUT programs is implementation dependent. The recommended behavior in the case of an error is to output a message and exit. In the case of a warning, the recommended behavior is to output a message and continue. All improper uses of the GLUT interface do not need to be caught or reported. What conditions are caught or reported should be based on how expensive the condition is to check for. For example, an implementation may not check every `glutSetWindow` call to determine if the window identifier is valid.

The run-time overhead of error checking for a very common operation may outweight the benefit of clean error reporting. This trade-off is left for the implementor to make. The implementor should also consider the difficulty of diagnosing the improper usage without a message being output. For example, if a GLUT program attempts to create a menu while a menu is in use (improper usage!), this warrants a message because this improper usage may often be benign, allowing the bug to easily go unnoticed.

14.4 Avoid Unspecified GLUT Usage Restrictions

GLUT implementations should be careful to not limit the conditions under which GLUT routines may be called. GLUT implementations are expected to be resilient when GLUT programs call GLUT routines with defined behavior at "unexpected" times. For example, a program should be permitted to destroy the *current window* from within a display callback (assuming the user does not then call GLUT routines requiring a *current window*). This means after dispatching callbacks, a GLUT implementation should be "defensive" about how the program might have used manipulated GLUT state during the callback.

A GLUT State

This appendix specifies precisely what programmer visible state GLUT maintains. There are three categories of programmer visible state that GLUT maintains: global, window, and menu. The window and menu state categories are maintained for each created window or menu. Additional overlay-related window state is maintained when an overlay is established for a window for the lifetime of the overlay.

The tables below name each element of state, define its type, specify what GLUT API entry points set or change the state (if possible), specify what GLUT API entry point or `glutGet`, `glutDeviceGet`, or `glutLayerGet` state constant is used to get the state (if possible), and how the state is initially set. For details of how any API entry point operates on the specified state, see the routine's official description. Footnotes for each category of state indicate additional caveats to the element of state.

A.1 Types of State

These types are used to specify GLUT's programmer visible state:

Bitmask A group of boolean bits.

Boolean True or false.

Callback A handle to a user-supplied routine invoked when the given callback is triggered (or `NULL` which is the default callback).

ColorCell Red, green, and blue color component triple, an array of which makes a colormap.

Cursor A GLUT cursor name.

Integer An integer value.

Layer Either normal plane or overlay.

MenuItem Either a menu entry or a submenu trigger. Both subtypes contain of a *String* name. A menu entry has an *Integer* value. A submenu cascade has an *Integer* menu name naming its associated submenu.

MenuState Either in use or not in use.

Stacking An ordering for top-level windows and sub-windows having the same parent. Higher windows obscure lower windows.

State One of shown, hidden, or iconified.

String A string of ASCII characters.

Timer A triple of a timer *Callback*, an *Integer* callback parameter, and a time in milliseconds (that expires in real time).

A.2 Global State

There are two types of global state: program controlled state which can be modified directly or indirectly by the program, and fixed system dependent state.

A.2.1 Program Controlled State

Name	Type	Set/Change	Get	Initial
currentWindow	Integer	glutSetWindow (1)	glutGetWindow	0
currentMenu	Integer	glutSetMenu (2)	glutGetMenu	0
initWindowX	Integer	glutInitWindowPosition	GLUT_INIT_WINDOW_X	-1
initWindowY	Integer	glutInitWindowPosition	GLUT_INIT_WINDOW_Y	-1
initWindowWidth	Integer	glutInitWindowSize	GLUT_INIT_WINDOW_WIDTH	300
initWindowHeight	Integer	glutInitWindowSize	GLUT_INIT_WINDOW_HEIGHT	300
initDisplayMode	Bitmask	glutInitDisplayMode	GLUT_INIT_DISPLAY_MODE	GLUT_RGB, GLUT_SINGLE, GLUT_DEPTH
idleCallback	Callback	glutIdleFunc	-	NULL
menuState	MenuState	-	(3)	NotInUse
menuStateCallback	Callback	glutMenuEntryFunc	-	NULL
timerList	list of Timer	glutTimerFunc	-	none

(1) The *currentWindow* is also changed implicitly by every window or menu callback (to the window triggering the callback) and the creation of a window (to the window being created).

(2) The *currentMenu* is also changed implicitly by every menu callback (to the menu triggering the callback) and the creation of a menu (to the menu being created).

(3) The menu state callback is triggered when the menuState changes.

A.2.2 Fixed System Dependent State

Name	Type	Get
screenWidth	Integer	GLUT_SCREEN_WIDTH
screenHeight	Integer	GLUT_SCREEN_HEIGHT
screenWidthMM	Integer	GLUT_SCREEN_WIDTH_MM
screenHeightMM	Integer	GLUT_SCREEN_HEIGHT_MM
hasKeyboard	Boolean	GLUT_HAS_KEYBOARD
hasMouse	Boolean	GLUT_HAS_MOUSE
hasSpaceball	Boolean	GLUT_HAS_SPACEBALL
hasDialAndButtonBox	Boolean	GLUT_HAS_DIAL_AND_BUTTON_BOX
hasTablet	Boolean	GLUT_HAS_TABLET
numMouseButtons	Integer	GLUT_NUM_MOUSE_BUTTONS
numSpaceballButtons	Integer	GLUT_NUM_SPACEBALL_BUTTONS
numButtonBoxButtons	Integer	GLUT_NUM_BUTTON_BOX_BUTTONS
numDials	Integer	GLUT_NUM_DIALS
numTabletButtons	Integer	GLUT_NUM_TABLET_BUTTONS

A.3 Window State

For the purposes of listing the window state elements, window state is classified into three types: base state, frame buffer capability state, and layer state. The tags *top-level*, *sub-win*, and *cindex* indicate the table entry applies only to top-level windows, subwindows, or color index windows respectively.

A.3.1 Basic State

Name	Type	Set/Change	Get	Initial
number	Integer	-	glutGetWindow	*top-level:* glutCreateWindow (1) *sub-win:* glutCreateSubWindow (1)
x	Integer	glutPositionWindow	GLUT_WINDOW_X	*top-level:* initWindowX (2) *sub-win:* glutCreateSubWindow
y	Integer	glutPositionWindow	GLUT_WINDOW_Y	*top-level:* initWindowY (3) *sub-win:* glutCreateSubWindow
width	Integer	glutReshapeWindow	GLUT_WINDOW_WIDTH	*top-level:* initWindowWidth (4) *sub-win:* glutCreateSubWindow
height	Integer	glutReshapeWindow	GLUT_WINDOW_HEIGHT	*top-level:* initWindowHeight (5) *sub-win:* glutCreateSubWindow
top-level: fullScreen	Boolean	glutFullScreen glutPositionWindow glutReshapeWindow (6)		False
cursor	Cursor	glutSetCursor	GLUT_WINDOW_CURSOR	GLUT_CURSOR_INHERIT
stacking	Stacking	glutPopWindow glutPushWindow	-	top
displayState	State (7)	glutShowWindow (8) glutHideWindow glutIconifyWindow	-	shown
visibility	Visibility	(9)	(10)	undefined
redisplay	Boolean	glutPostRedisplay (11)	-	False
top-level: windowTitle	String	glutWindowTitle	-	glutCreateWindow
top-level: iconTitle	String	glutIconTitle	-	glutCreateWindow
displayCallback	Callback	glutDisplayFunc	-	NULL (12)
reshapeCallback	Callback	glutReshapeFunc	-	NULL (13)
keyboardCallback	Callback	glutKeyboardFunc	-	NULL
mouseCallback	Callback	glutMouseFunc	-	NULL
motionCallback	Callback	glutMotionFunc	-	NULL
passiveMotionCallback	Callback	glutPassiveMotionFunc	-	NULL
specialCallback	Callback	glutSpecialFunc	-	NULL
spaceballMotionCallback	Callback	glutSpaceballMotionFunc	-	NULL
spaceballRotateCallback	Callback	glutSpaceballRotateFunc	-	NULL
spaceballButtonCallback	Callback	glutSpaceballButtonFunc	-	NULL
buttonBoxCallback	Callback	glutButtonBoxFunc	-	NULL
dialsCallback	Callback	glutDialsFunc	-	NULL
tabletMotionCallback	Callback	glutTabletMotionFunc	-	NULL
tabletButtonCallback	Callback	glutTabletButtonFunc	-	NULL
visibilityCallback	Callback	glutVisibilityFunc	-	NULL
entryCallback	Callback	glutEntryFunc	-	NULL
cindex: colormap	array of ColorCell	glutSetColor	glutGetColor	undefined
windowParent	Integer	-	GLUT_WINDOW_PARENT	*top-level:* 0 *sub-win:* (14)
numChildren	Integer	glutCreateSubWindow glutDestroyWindow glutCopyColormap	GLUT_NUM_CHILDREN	0
leftMenu	Integer	glutAttachMenu glutDetachMenu	-	0
middleMenu	Integer	glutAttachMenu glutDetachMenu	-	0
rightMenu	Integer	glutAttachMenu glutDetachMenu	-	0

(1) Assigned dynamically from unassigned window numbers greater than zero.

(2) If *initWindowX* is greater or equal to zero *and initWindowY* is greater or equal to zero then *initWindowX*, else window location left to window system to decide.

(3) If *initWindowY* is greater or equal to zero *and initWindowX* is greater or equal to zero then *initWindowY*, else window location left to window system to decide.

(4) If *initWindowWidth* is greater than zero *and initWindowHeight* is greater than zero the *initWindowWidth*, else window size left to window system to decide.

(5) If *initWindowHeight* is greater than zero *and initWindowWidth* is greater than zero then *initWindowHeight*, else window size left to window system to decide.

(6) `glutFullScreen` sets to true; `glutPositionWindow` and `glutReshapeWindow` set to false.

(7) Subwindows can not be iconified.

(8) Window system events can also change the displayState.

(9) Visibility of a window can change for window system dependent reason, for example, a new window may occlude the window. `glutPopWindow` and `glutPushWindow` can affect window visibility as a side effect.

(10) The visibility callback set by `glutVisibilityFunc` allows the visibility state to be tracked.

(11) The redisplay state can be explicitly enabled by `glutRedisplayFunc` or implicitly in response to normal plane redisplay events from the window system.

(12) A window's *displayCallback must* be registered before the first display callback would be triggered (or the program is terminated).

(13) Instead of being a no-op as most `NULL` callbacks are, a `NULL` *reshapeCallback* sets the OpenGL viewport to render into the complete window, i.e., `glViewport(0,0,width, height)`.

(14) Determined by *currentWindow* at `glutCreateSubWindow` time.

A.3.2 Frame Buffer Capability State

Name	Type	Get
Total number of bits in color buffer	Integer	GLUT_WINDOW_BUFFER_SIZE
Number of bits in stencil buffer	Integer	GLUT_WINDOW_STENCIL_SIZE
Number of bits in depth buffer	Integer	GLUT_WINDOW_DEPTH_SIZE
Number of bits of red stored in color buffer	Integer	GLUT_WINDOW_RED_SIZE
Number of bits of green stored in color buffer	Integer	GLUT_WINDOW_GREEN_SIZE
Number of bits of blue stored in color buffer	Integer	GLUT_WINDOW_BLUE_SIZE
Number of bits of alpha stored in color buffer	Integer	GLUT_WINDOW_ALPHA_SIZE
Number of bits of red stored in accumulation buffer	Integer	GLUT_WINDOW_ACCUM_RED_SIZE
Number of bits of green stored in accumulation buffer	Integer	GLUT_WINDOW_ACCUM_GREEN_SIZE
Number of bits of blue stored in accumulation buffer	Integer	GLUT_WINDOW_ACCUM_BLUE_SIZE
Number of bits of alpha stored in accumulation buffer	Integer	GLUT_WINDOW_ACCUM_ALPHA_SIZE
Color index colormap size	Integer	GLUT_WINDOW_COLORMAP_SIZE
If double buffered	Boolean	GLUT_WINDOW_DOUBLEBUFFER
If RGBA color model	Boolean	GLUT_WINDOW_RGBA
If stereo	Boolean	GLUT_WINDOW_STEREO
Number of samples for multisampling	Integer	GLUT_WINDOW_MULTISAMPLE

A window's (normal plane) frame buffer capability state is derived from the global initDisplayMode state at the window's creation. A window's frame buffer capabilities can not be changed.

A.3.3 Layer State

Name	Type	Set/Change	Get	Initial
hasOverlay	Boolean	glutEstablishOverlay glutRemoveOverlay	GLUT_HAS_OVERLAY	False
overlayPossible	Boolean	(1)	GLUT_OVERLAY_POSSIBLE	False
layerInUse	Layer	glutUseLayer (2)	GLUT_LAYER_IN_USE	normal plane
cindex: transparentIndex	Integer	-	GLUT_TRANSPARENT_INDEX	(3)
overlayRedisplay	Boolean	glutPostOverlayRedisplay (4)	-	False
overlayDisplayCallback	Callback	glutOverlayDisplayFunc	-	NULL
overlayDisplayState	State	glutShowOverlay glutHideOverlay	-	shown
normalDamaged	Boolean	(5)	GLUT_NORMAL_DAMAGED	False
overlayDamaged	Boolean	(6)	GLUT_NORMAL_DAMAGED	False

(1) Whether an overlay is possible is based on the *initDisplayMode* state *and* the frame buffer capability state of the window.

(2) The *layerInUse* is implicitly set to overlay after `glutEstablishOverlay`; likewise, `glutRemoveOverlay` resets the state to normal plane.

(3) The *transparentIndex* is set when a color index overlay is established. It cannot be set; it may change if the overlay is re-established. When no overlay is in use or if the overlay is not color index, the *transparentIndex* is -1.

(4) The *overlayRedisplay* state can be explicitly enabled by `glutPostOverlayRedisplay` or implicitly in response to overlay redisplay events from the window system.

(5) Set when the window system reports a region of the window's normal plane is undefined (for example, damaged by another window moving or being initially shown). The specifics of when damage occurs are left to the window system to determine. The window's *redisplay* state is always set true when damage occurs. *normalDamaged* is cleared whenever the window's display callback returns.

(6) Set when the window system reports a region of the window's overlay plane is undefined (for example, damaged by another window moving or being initially shown). The specifics of when damage occurs are left to the window system to determine. The damage may occur independent from damage to the window's normal plane. The window's *redisplay* state is always set true when damage occurs. *normalDamaged* is cleared whenever the window's display callback returns.

When an overlay is established, *overlay* frame buffer capability state is maintained as described in Section A.3.2. The *layerInUse* determines whether `glutGet` returns normal plane or overlay state when an overlay is established.

A.4 Menu State

Name	Type	Set/Change	Get	Initial
number	Integer	-	glutSetMenu	*top-level:* glutCreateMenu (1)
select	Callback	-	-	glutCreateMenu
items	list of MenuItem	-	-	-
numItems	Integer	-	GLUT_MENU_NUM_ITEMS	0

(1) Assigned dynamically from unassigned window numbers greater than zero.

B glut.h ANSI C Header File

```
#ifndef __glut_h__
#define __glut_h__

/* Copyright (c) Mark J. Kilgard, 1994, 1995, 1996. */

/* This program is freely distributable without licensing fees  and is
   provided without guarantee or warrantee expressed or  implied. This
   program is -not- in the public domain. */

#include <GL/gl.h>
#include <GL/glu.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * GLUT API revision history:
 *
 * GLUT_API_VERSION is updated to reflect incompatible GLUT
 * API changes (interface changes, semantic changes, deletions,
 * or additions).
 *
 * GLUT_API_VERSION=1  First public release of GLUT.  11/29/94
 *
 * GLUT_API_VERSION=2  Added support for OpenGL/GLX multisampling,
 * extension.  Supports new input devices like tablet, dial and button
 * box, and Spaceball.  Easy to query OpenGL extensions.
 *
 * GLUT_API_VERSION=3  glutMenuStatus added.
 *
 */
#ifndef GLUT_API_VERSION  /* allow this to be overriden */
#define GLUT_API_VERSION                3
#endif

/*
 * GLUT implementation revision history:
 *
 * GLUT_XLIB_IMPLEMENTATION is updated to reflect both GLUT
 * API revisions and implementation revisions (ie, bug fixes).
 *
 * GLUT_XLIB_IMPLEMENTATION=1  mjk's first public release of
 * GLUT Xlib-based implementation.  11/29/94
 *
 * GLUT_XLIB_IMPLEMENTATION=2  mjk's second public release of
 * GLUT Xlib-based implementation providing GLUT version 2
 * interfaces.
 *
 * GLUT_XLIB_IMPLEMENTATION=3  mjk's GLUT 2.2 images. 4/17/95
 *
 * GLUT_XLIB_IMPLEMENTATION=4  mjk's GLUT 2.3 images. 6/??/95
 *
 * GLUT_XLIB_IMPLEMENTATION=5  mjk's GLUT 3.0 images. 10/??/95
 *
 */
#ifndef GLUT_XLIB_IMPLEMENTATION  /* allow this to be overriden */
#define GLUT_XLIB_IMPLEMENTATION        5
#endif

/* display mode bit masks */
#define GLUT_RGB                        0
#define GLUT_RGBA                       GLUT_RGB
#define GLUT_INDEX                      1
#define GLUT_SINGLE                     0
#define GLUT_DOUBLE                     2
```

```
#define GLUT_ACCUM                      4
#define GLUT_ALPHA                      8
#define GLUT_DEPTH                      16
#define GLUT_STENCIL                    32
#if (GLUT_API_VERSION >= 2)
#define GLUT_MULTISAMPLE                128
#define GLUT_STEREO                     256
#endif
#if (GLUT_API_VERSION >= 3)
#define GLUT_LUMINANCE                  512
#endif

/* mouse buttons */
#define GLUT_LEFT_BUTTON                0
#define GLUT_MIDDLE_BUTTON              1
#define GLUT_RIGHT_BUTTON               2

/* mouse button callback state */
#define GLUT_DOWN                       0
#define GLUT_UP                         1

#if (GLUT_API_VERSION >= 2)
/* function keys */
#define GLUT_KEY_F1                     1
#define GLUT_KEY_F2                     2
#define GLUT_KEY_F3                     3
#define GLUT_KEY_F4                     4
#define GLUT_KEY_F5                     5
#define GLUT_KEY_F6                     6
#define GLUT_KEY_F7                     7
#define GLUT_KEY_F8                     8
#define GLUT_KEY_F9                     9
#define GLUT_KEY_F10                    10
#define GLUT_KEY_F11                    11
#define GLUT_KEY_F12                    12
/* directional keys */
#define GLUT_KEY_LEFT                   100
#define GLUT_KEY_UP                     101
#define GLUT_KEY_RIGHT                  102
#define GLUT_KEY_DOWN                   103
#define GLUT_KEY_PAGE_UP                104
#define GLUT_KEY_PAGE_DOWN              105
#define GLUT_KEY_HOME                   106
#define GLUT_KEY_END                    107
#define GLUT_KEY_INSERT                 108
#endif

/* entry/exit callback state */
#define GLUT_LEFT                       0
#define GLUT_ENTERED                    1

/* menu usage callback state */
#define GLUT_MENU_NOT_IN_USE            0
#define GLUT_MENU_IN_USE                1

/* visibility callback state */
#define GLUT_NOT_VISIBLE                0
#define GLUT_VISIBLE                    1

/* color index component selection values */
#define GLUT_RED                        0
#define GLUT_GREEN                      1
#define GLUT_BLUE                       2

/* layers for use */
#define GLUT_NORMAL                     0
#define GLUT_OVERLAY                    1

```

```
/* stroke font opaque addresses (use constants instead in source code) */
extern void *glutStrokeRoman;
extern void *glutStrokeMonoRoman;

/* stroke font constants (use these in GLUT program) */
#define GLUT_STROKE_ROMAN               (&glutStrokeRoman)
#define GLUT_STROKE_MONO_ROMAN          (&glutStrokeMonoRoman)

/* bitmap font opaque addresses (use constants instead in source code) */
extern void *glutBitmap9By15;
extern void *glutBitmap8By13;
extern void *glutBitmapTimesRoman10;
extern void *glutBitmapTimesRoman24;
extern void *glutBitmapHelvetica10;
extern void *glutBitmapHelvetica12;
extern void *glutBitmapHelvetica18;

/* bitmap font constants (use these in GLUT program) */
#define GLUT_BITMAP_9_BY_15             (&glutBitmap9By15)
#define GLUT_BITMAP_8_BY_13             (&glutBitmap8By13)
#define GLUT_BITMAP_TIMES_ROMAN_10      (&glutBitmapTimesRoman10)
#define GLUT_BITMAP_TIMES_ROMAN_24      (&glutBitmapTimesRoman24)
#if (GLUT_API_VERSION >= 3)
#define GLUT_BITMAP_HELVETICA_10        (&glutBitmapHelvetica10)
#define GLUT_BITMAP_HELVETICA_12        (&glutBitmapHelvetica12)
#define GLUT_BITMAP_HELVETICA_18        (&glutBitmapHelvetica18)
#endif

/* glutGet parameters */
#define GLUT_WINDOW_X                   100
#define GLUT_WINDOW_Y                   101
#define GLUT_WINDOW_WIDTH               102
#define GLUT_WINDOW_HEIGHT              103
#define GLUT_WINDOW_BUFFER_SIZE         104
#define GLUT_WINDOW_STENCIL_SIZE        105
#define GLUT_WINDOW_DEPTH_SIZE          106
#define GLUT_WINDOW_RED_SIZE            107
#define GLUT_WINDOW_GREEN_SIZE          108
#define GLUT_WINDOW_BLUE_SIZE           109
#define GLUT_WINDOW_ALPHA_SIZE          110
#define GLUT_WINDOW_ACCUM_RED_SIZE      111
#define GLUT_WINDOW_ACCUM_GREEN_SIZE    112
#define GLUT_WINDOW_ACCUM_BLUE_SIZE     113
#define GLUT_WINDOW_ACCUM_ALPHA_SIZE    114
#define GLUT_WINDOW_DOUBLEBUFFER        115
#define GLUT_WINDOW_RGBA                116
#define GLUT_WINDOW_PARENT              117
#define GLUT_WINDOW_NUM_CHILDREN        118
#define GLUT_WINDOW_COLORMAP_SIZE       119
#if (GLUT_API_VERSION >= 2)
#define GLUT_WINDOW_NUM_SAMPLES         120
#define GLUT_WINDOW_STEREO              121
#endif
#if (GLUT_API_VERSION >= 3)
#define GLUT_WINDOW_CURSOR              122
#endif
#define GLUT_SCREEN_WIDTH               200
#define GLUT_SCREEN_HEIGHT              201
#define GLUT_SCREEN_WIDTH_MM            202
#define GLUT_SCREEN_HEIGHT_MM           203
#define GLUT_MENU_NUM_ITEMS             300
#define GLUT_DISPLAY_MODE_POSSIBLE      400
#define GLUT_INIT_WINDOW_X              500
#define GLUT_INIT_WINDOW_Y              501
#define GLUT_INIT_WINDOW_WIDTH          502
#define GLUT_INIT_WINDOW_HEIGHT         503
#define GLUT_INIT_DISPLAY_MODE          504
#if (GLUT_API_VERSION >= 2)
```

```
#define GLUT_ELAPSED_TIME              700
#endif

#if (GLUT_API_VERSION >= 2)
/* glutDeviceGet parameters */
#define GLUT_HAS_KEYBOARD              600
#define GLUT_HAS_MOUSE                 601
#define GLUT_HAS_SPACEBALL             602
#define GLUT_HAS_DIAL_AND_BUTTON_BOX   603
#define GLUT_HAS_TABLET                604
#define GLUT_NUM_MOUSE_BUTTONS         605
#define GLUT_NUM_SPACEBALL_BUTTONS     606
#define GLUT_NUM_BUTTON_BOX_BUTTONS    607
#define GLUT_NUM_DIALS                 608
#define GLUT_NUM_TABLET_BUTTONS        609
#endif

#if (GLUT_API_VERSION >= 3)
/* glutLayerGet parameters */
#define GLUT_OVERLAY_POSSIBLE          800
#define GLUT_LAYER_IN_USE              801
#define GLUT_HAS_OVERLAY               802
#define GLUT_TRANSPARENT_INDEX         803
#define GLUT_NORMAL_DAMAGED            804
#define GLUT_OVERLAY_DAMAGED           805

/* glutUseLayer parameters */
#define GLUT_NORMAL                    0
#define GLUT_OVERLAY                   1

/* glutGetModifiers return mask */
#define GLUT_ACTIVE_SHIFT              1
#define GLUT_ACTIVE_CTRL               2
#define GLUT_ACTIVE_ALT                4

/* glutSetCursor parameters */
/* Basic arrows */
#define GLUT_CURSOR_RIGHT_ARROW        0
#define GLUT_CURSOR_LEFT_ARROW         1
/* Symbolic cursor shapees */
#define GLUT_CURSOR_INFO               2
#define GLUT_CURSOR_DESTROY            3
#define GLUT_CURSOR_HELP               4
#define GLUT_CURSOR_CYCLE              5
#define GLUT_CURSOR_SPRAY              6
#define GLUT_CURSOR_WAIT               7
#define GLUT_CURSOR_TEXT               8
#define GLUT_CURSOR_CROSSHAIR          9
/* Directional cursors */
#define GLUT_CURSOR_UP_DOWN            10
#define GLUT_CURSOR_LEFT_RIGHT         11
/* Sizing cursors */
#define GLUT_CURSOR_TOP_SIDE           12
#define GLUT_CURSOR_BOTTOM_SIDE        13
#define GLUT_CURSOR_LEFT_SIDE          14
#define GLUT_CURSOR_RIGHT_SIDE         15
#define GLUT_CURSOR_TOP_LEFT_CORNER    16
#define GLUT_CURSOR_TOP_RIGHT_CORNER   17
#define GLUT_CURSOR_BOTTOM_RIGHT_CORNER 18
#define GLUT_CURSOR_BOTTOM_LEFT_CORNER  19
/* Inherit from parent window */
#define GLUT_CURSOR_INHERIT            100
/* Blank cursor */
#define GLUT_CURSOR_NONE               101
/* Fullscreen crosshair (if available) */
#define GLUT_CURSOR_FULL_CROSSHAIR     102
#endif
```

```
/* GLUT initialization sub-API */
extern void glutInit(int *argcp, char **argv);
extern void glutInitDisplayMode(unsigned int mode);
extern void glutInitWindowPosition(int x, int y);
extern void glutInitWindowSize(int width, int height);
extern void glutMainLoop(void);

/* GLUT window sub-api */
extern int glutCreateWindow(char *title);
extern int glutCreateSubWindow(int win, int x, int y, int width, int height);
extern void glutDestroyWindow(int win);
extern void glutPostRedisplay(void);
extern void glutSwapBuffers(void);
extern int glutGetWindow(void);
extern void glutSetWindow(int win);
extern void glutSetWindowTitle(char *title);
extern void glutSetIconTitle(char *title);
extern void glutPositionWindow(int x, int y);
extern void glutReshapeWindow(int width, int height);
extern void glutPopWindow(void);
extern void glutPushWindow(void);
extern void glutIconifyWindow(void);
extern void glutShowWindow(void);
extern void glutHideWindow(void);
#if (GLUT_API_VERSION >= 3)
extern void glutFullScreen(void);
extern void glutSetCursor(int cursor);

/* GLUT overlay sub-API */
extern void glutEstablishOverlay(void);
extern void glutRemoveOverlay(void);
extern void glutUseLayer(GLenum layer);
extern void glutPostOverlayRedisplay(void);
extern void glutShowOverlay(void);
extern void glutHideOverlay(void);
#endif

/* GLUT menu sub-API */
extern int glutCreateMenu(void (*)(int));
extern void glutDestroyMenu(int menu);
extern int glutGetMenu(void);
extern void glutSetMenu(int menu);
extern void glutAddMenuEntry(char *label, int value);
extern void glutAddSubMenu(char *label, int submenu);
extern void glutChangeToMenuEntry(int item, char *label, int value);
extern void glutChangeToSubMenu(int item, char *label, int submenu);
extern void glutRemoveMenuItem(int item);
extern void glutAttachMenu(int button);
extern void glutDetachMenu(int button);

/* GLUT callback sub-api */
extern void glutDisplayFunc(void (*)(void));
extern void glutReshapeFunc(void (*)(int width, int height));
extern void glutKeyboardFunc(void (*)(unsigned char key, int x, int y));
extern void glutMouseFunc(void (*)(int button, int state, int x, int y));
extern void glutMotionFunc(void (*)(int x, int y));
extern void glutPassiveMotionFunc(void (*)(int x, int y));
extern void glutEntryFunc(void (*)(int state));
extern void glutVisibilityFunc(void (*)(int state));
extern void glutIdleFunc(void (*)(void));
extern void glutTimerFunc(unsigned int millis, void (*)(int value), int value);
extern void glutMenuStateFunc(void (*)(int state));
#if (GLUT_API_VERSION >= 2)
extern void glutSpecialFunc(void (*)(int key, int x, int y));
extern void glutSpaceballMotionFunc(void (*)(int x, int y, int z));
extern void glutSpaceballRotateFunc(void (*)(int x, int y, int z));
extern void glutSpaceballButtonFunc(void (*)(int button, int state));
extern void glutButtonBoxFunc(void (*)(int button, int state));
```

```
extern void glutDialsFunc(void (*)(int dial, int value));
extern void glutTabletMotionFunc(void (*)(int x, int y));
extern void glutTabletButtonFunc(void (*)(int button, int state, int x, int y));
#if (GLUT_API_VERSION >= 3)
extern void glutMenuStatusFunc(void (*)(int status, int x, int y));
extern void glutOverlayDisplayFunc(void (*)(void));
#endif
#endif

/* GLUT color index sub-api */
extern void glutSetColor(int, GLfloat red, GLfloat green, GLfloat blue);
extern GLfloat glutGetColor(int ndx, int component);
extern void glutCopyColormap(int win);

/* GLUT state retrieval sub-api */
extern int glutGet(GLenum type);
extern int glutDeviceGet(GLenum type);
#if (GLUT_API_VERSION >= 2)
/* GLUT extension support sub-API */
extern int glutExtensionSupported(char *name);
#endif
#if (GLUT_API_VERSION >= 3)
extern int glutGetModifiers(void);
extern int glutLayerGet(GLenum type);
#endif

/* GLUT font sub-API */
extern void glutBitmapCharacter(void *font, int character);
extern int glutBitmapWidth(void *font, int character);
extern void glutStrokeCharacter(void *font, int character);
extern int glutStrokeWidth(void *font, int character);

/* GLUT pre-built models sub-API */
extern void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
extern void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
extern void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
extern void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
extern void glutWireCube(GLdouble size);
extern void glutSolidCube(GLdouble size);
extern void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
extern void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
extern void glutWireDodecahedron(void);
extern void glutSolidDodecahedron(void);
extern void glutWireTeapot(GLdouble size);
extern void glutSolidTeapot(GLdouble size);
extern void glutWireOctahedron(void);
extern void glutSolidOctahedron(void);
extern void glutWireTetrahedron(void);
extern void glutSolidTetrahedron(void);
extern void glutWireIcosahedron(void);
extern void glutSolidIcosahedron(void);

#ifdef __cplusplus
}

#endif
#endif                  /* __glut_h__ */
```

C fglut.h FORTRAN Header File

```
C  Copyright (c) Mark J. Kilgard, 1994.

C  This program is freely distributable without licensing fees
C  and is provided without guarantee or warrantee expressed or
C  implied.  This program is -not- in the public domain.

C  GLUT Fortran header file

C  display mode bit masks
        integer*4 GLUT_RGB
        parameter ( GLUT_RGB = 0 )
        integer*4 GLUT_RGBA
        parameter ( GLUT_RGBA = 0 )
        integer*4 GLUT_INDEX
        parameter ( GLUT_INDEX = 1 )
        integer*4 GLUT_SINGLE
        parameter ( GLUT_SINGLE = 0 )
        integer*4 GLUT_DOUBLE
        parameter ( GLUT_DOUBLE = 2 )
        integer*4 GLUT_ACCUM
        parameter ( GLUT_ACCUM = 4 )
        integer*4 GLUT_ALPHA
        parameter ( GLUT_ALPHA = 8 )
        integer*4 GLUT_DEPTH
        parameter ( GLUT_DEPTH = 16 )
        integer*4 GLUT_STENCIL
        parameter ( GLUT_STENCIL = 32 )
        integer*4 GLUT_MULTISAMPLE
        parameter ( GLUT_MULTISAMPLE = 128 )
        integer*4 GLUT_STEREO
        parameter ( GLUT_STEREO = 256 )

C  mouse buttons
        integer*4 GLUT_LEFT_BUTTON
        parameter ( GLUT_LEFT_BUTTON = 0 )
        integer*4 GLUT_MIDDLE_BUTTON
        parameter ( GLUT_MIDDLE_BUTTON = 1 )
        integer*4 GLUT_RIGHT_BUTTON
        parameter ( GLUT_RIGHT_BUTTON = 2 )

C  mouse button callback state
        integer*4 GLUT_DOWN
        parameter ( GLUT_DOWN = 0 )
        integer*4 GLUT_UP
        parameter ( GLUT_UP = 1 )

C  special key callback values
        integer*4 GLUT_KEY_F1
        parameter ( GLUT_KEY_F1 = 1 )
        integer*4 GLUT_KEY_F2
        parameter ( GLUT_KEY_F2 = 2 )
        integer*4 GLUT_KEY_F3
        parameter ( GLUT_KEY_F3 = 3 )
        integer*4 GLUT_KEY_F4
        parameter ( GLUT_KEY_F4 = 4 )
        integer*4 GLUT_KEY_F5
        parameter ( GLUT_KEY_F5 = 5 )
        integer*4 GLUT_KEY_F6
        parameter ( GLUT_KEY_F6 = 6 )
        integer*4 GLUT_KEY_F7
        parameter ( GLUT_KEY_F7 = 7 )
        integer*4 GLUT_KEY_F8
        parameter ( GLUT_KEY_F8 = 8 )
        integer*4 GLUT_KEY_F9
        parameter ( GLUT_KEY_F9 = 9 )
        integer*4 GLUT_KEY_F10
```

```
      parameter ( GLUT_KEY_F10 = 10 )
      integer*4 GLUT_KEY_F11
      parameter ( GLUT_KEY_F11 = 11 )
      integer*4 GLUT_KEY_F12
      parameter ( GLUT_KEY_F12 = 12 )
      integer*4 GLUT_KEY_LEFT
      parameter ( GLUT_KEY_LEFT = 100 )
      integer*4 GLUT_KEY_UP
      parameter ( GLUT_KEY_UP = 101 )
      integer*4 GLUT_KEY_RIGHT
      parameter ( GLUT_KEY_RIGHT = 102 )
      integer*4 GLUT_KEY_DOWN
      parameter ( GLUT_KEY_DOWN = 103 )
      integer*4 GLUT_KEY_PAGE_UP
      parameter ( GLUT_KEY_PAGE_UP = 104 )
      integer*4 GLUT_KEY_PAGE_DOWN
      parameter ( GLUT_KEY_PAGE_DOWN = 105 )
      integer*4 GLUT_KEY_HOME
      parameter ( GLUT_KEY_HOME = 106 )
      integer*4 GLUT_KEY_END
      parameter ( GLUT_KEY_END = 107 )
      integer*4 GLUT_KEY_INSERT
      parameter ( GLUT_KEY_INSERT = 108 )

C  entry/exit callback state
      integer*4 GLUT_LEFT
      parameter ( GLUT_LEFT = 0 )
      integer*4 GLUT_ENTERED
      parameter ( GLUT_ENTERED = 1 )

C  menu usage callback state
      integer*4 GLUT_MENU_NOT_IN_USE
      parameter ( GLUT_MENU_NOT_IN_USE = 0 )
      integer*4 GLUT_MENU_IN_USE
      parameter ( GLUT_MENU_IN_USE = 1 )

C  visibility callback state
      integer*4 GLUT_NOT_VISIBLE
      parameter ( GLUT_NOT_VISIBLE = 0 )
      integer*4 GLUT_VISIBLE
      parameter ( GLUT_VISIBLE = 1 )

C  color index component selection values
      integer*4 GLUT_RED
      parameter ( GLUT_RED = 0 )
      integer*4 GLUT_GREEN
      parameter ( GLUT_GREEN = 1 )
      integer*4 GLUT_BLUE
      parameter ( GLUT_BLUE = 2 )

C  XXX Unfortunately, SGI's Fortran compiler links with
C  EXTERNAL data even if it is not used.  This defeats
C  the purpose of GLUT naming fonts via opaque symbols.
C  This means GLUT Fortran programmers should explicitly
C  declared EXTERNAL GLUT fonts in subroutines where
C  the fonts are used.

C  stroke font opaque names
C     external GLUT_STROKE_ROMAN
C     external GLUT_STROKE_MONO_ROMAN

C  bitmap font opaque names
C     external GLUT_BITMAP_9_BY_15
C     external GLUT_BITMAP_8_BY_13
C     external GLUT_BITMAP_TIMES_ROMAN_10
C     external GLUT_BITMAP_TIMES_ROMAN_24
C     external GLUT_BITMAP_HELVETICA_10
C     external GLUT_BITMAP_HELVETICA_12
```

```
C       external GLUT_BITMAP_HELVETICA_18

C  glutGet parameters
        integer*4 GLUT_WINDOW_X
        parameter ( GLUT_WINDOW_X = 100 )
        integer*4 GLUT_WINDOW_Y
        parameter ( GLUT_WINDOW_Y = 101 )
        integer*4 GLUT_WINDOW_WIDTH
        parameter ( GLUT_WINDOW_WIDTH = 102 )
        integer*4 GLUT_WINDOW_HEIGHT
        parameter ( GLUT_WINDOW_HEIGHT = 103 )
        integer*4 GLUT_WINDOW_BUFFER_SIZE
        parameter ( GLUT_WINDOW_BUFFER_SIZE = 104 )
        integer*4 GLUT_WINDOW_STENCIL_SIZE
        parameter ( GLUT_WINDOW_STENCIL_SIZE = 105 )
        integer*4 GLUT_WINDOW_DEPTH_SIZE
        parameter ( GLUT_WINDOW_DEPTH_SIZE = 106 )
        integer*4 GLUT_WINDOW_RED_SIZE
        parameter ( GLUT_WINDOW_RED_SIZE = 107 )
        integer*4 GLUT_WINDOW_GREEN_SIZE
        parameter ( GLUT_WINDOW_GREEN_SIZE = 108 )
        integer*4 GLUT_WINDOW_BLUE_SIZE
        parameter ( GLUT_WINDOW_BLUE_SIZE = 109 )
        integer*4 GLUT_WINDOW_ALPHA_SIZE
        parameter ( GLUT_WINDOW_ALPHA_SIZE = 110 )
        integer*4 GLUT_WINDOW_ACCUM_RED_SIZE
        parameter ( GLUT_WINDOW_ACCUM_RED_SIZE = 111 )
        integer*4 GLUT_WINDOW_ACCUM_GREEN_SIZE
        parameter ( GLUT_WINDOW_ACCUM_GREEN_SIZE = 112 )
        integer*4 GLUT_WINDOW_ACCUM_BLUE_SIZE
        parameter ( GLUT_WINDOW_ACCUM_BLUE_SIZE = 113 )
        integer*4 GLUT_WINDOW_ACCUM_ALPHA_SIZE
        parameter ( GLUT_WINDOW_ACCUM_ALPHA_SIZE = 114 )
        integer*4 GLUT_WINDOW_DOUBLEBUFFER
        parameter ( GLUT_WINDOW_DOUBLEBUFFER = 115 )
        integer*4 GLUT_WINDOW_RGBA
        parameter ( GLUT_WINDOW_RGBA = 116 )
        integer*4 GLUT_WINDOW_PARENT
        parameter ( GLUT_WINDOW_PARENT = 117 )
        integer*4 GLUT_WINDOW_NUM_CHILDREN
        parameter ( GLUT_WINDOW_NUM_CHILDREN = 118 )
        integer*4 GLUT_WINDOW_COLORMAP_SIZE
        parameter ( GLUT_WINDOW_COLORMAP_SIZE = 119 )
        integer*4 GLUT_WINDOW_NUM_SAMPLES
        parameter ( GLUT_WINDOW_NUM_SAMPLES = 120 )
        integer*4 GLUT_WINDOW_STEREO
        parameter ( GLUT_WINDOW_STEREO = 121 )
        integer*4 GLUT_WINDOW_CURSOR
        parameter ( GLUT_WINDOW_CURSOR = 122 )
        integer*4 GLUT_SCREEN_WIDTH
        parameter ( GLUT_SCREEN_WIDTH = 200 )
        integer*4 GLUT_SCREEN_HEIGHT
        parameter ( GLUT_SCREEN_HEIGHT = 201 )
        integer*4 GLUT_SCREEN_WIDTH_MM
        parameter ( GLUT_SCREEN_WIDTH_MM = 202 )
        integer*4 GLUT_SCREEN_HEIGHT_MM
        parameter ( GLUT_SCREEN_HEIGHT_MM = 203 )
        integer*4 GLUT_MENU_NUM_ITEMS
        parameter ( GLUT_MENU_NUM_ITEMS = 300 )
        integer*4 GLUT_DISPLAY_MODE_POSSIBLE
        parameter ( GLUT_DISPLAY_MODE_POSSIBLE = 400 )
        integer*4 GLUT_INIT_WINDOW_X
        parameter ( GLUT_INIT_WINDOW_X = 500 )
        integer*4 GLUT_INIT_WINDOW_Y
        parameter ( GLUT_INIT_WINDOW_Y = 501 )
        integer*4 GLUT_INIT_WINDOW_WIDTH
        parameter ( GLUT_INIT_WINDOW_WIDTH = 502 )
        integer*4 GLUT_INIT_WINDOW_HEIGHT
```

```
      parameter ( GLUT_INIT_WINDOW_HEIGHT = 503 )
      integer*4 GLUT_INIT_DISPLAY_MODE
      parameter ( GLUT_INIT_DISPLAY_MODE = 504 )
      integer*4 GLUT_ELAPSED_TIME
      parameter ( GLUT_ELAPSED_TIME = 700 )

C  glutDeviceGet parameters
      integer*4 GLUT_HAS_KEYBOARD
      parameter ( GLUT_HAS_KEYBOARD = 600 )
      integer*4 GLUT_HAS_MOUSE
      parameter ( GLUT_HAS_MOUSE = 601 )
      integer*4 GLUT_HAS_SPACEBALL
      parameter ( GLUT_HAS_SPACEBALL = 602 )
      integer*4 GLUT_HAS_DIAL_AND_BUTTON_BOX
      parameter ( GLUT_HAS_DIAL_AND_BUTTON_BOX = 603 )
      integer*4 GLUT_HAS_TABLET
      parameter ( GLUT_HAS_TABLET = 604 )
      integer*4 GLUT_NUM_MOUSE_BUTTONS
      parameter ( GLUT_NUM_MOUSE_BUTTONS = 605 )
      integer*4 GLUT_NUM_SPACEBALL_BUTTONS
      parameter ( GLUT_NUM_SPACEBALL_BUTTONS = 606 )
      integer*4 GLUT_NUM_BUTTON_BOX_BUTTONS
      parameter ( GLUT_NUM_BUTTON_BOX_BUTTONS = 607 )
      integer*4 GLUT_NUM_DIALS
      parameter ( GLUT_NUM_DIALS = 608 )
      integer*4 GLUT_NUM_TABLET_BUTTONS
      parameter ( GLUT_NUM_TABLET_BUTTONS = 609 )

C  glutLayerGet parameters
      integer*4 GLUT_OVERLAY_POSSIBLE
      parameter ( GLUT_OVERLAY_POSSIBLE = 800 )
      integer*4 GLUT_LAYER_IN_USE
      parameter ( GLUT_LAYER_IN_USE = 801 )
      integer*4 GLUT_HAS_OVERLAY
      parameter ( GLUT_HAS_OVERLAY = 802 )
      integer*4 GLUT_TRANSPARENT_INDEX
      parameter ( GLUT_TRANSPARENT_INDEX = 803 )
      integer*4 GLUT_NORMAL_DAMAGED
      parameter ( GLUT_NORMAL_DAMAGED = 804 )
      integer*4 GLUT_OVERLAY_DAMAGED
      parameter ( GLUT_OVERLAY_DAMAGED = 805 )

C  glutUseLayer parameters
      integer*4 GLUT_NORMAL
      parameter ( GLUT_NORMAL = 0 )
      integer*4 GLUT_OVERLAY
      parameter ( GLUT_OVERLAY = 1 )

C  glutGetModifiers return mask
      integer*4 GLUT_ACTIVE_SHIFT
      parameter ( GLUT_ACTIVE_SHIFT = 1 )
      integer*4 GLUT_ACTIVE_CTRL
      parameter ( GLUT_ACTIVE_CTRL = 2 )
      integer*4 GLUT_ACTIVE_ALT
      parameter ( GLUT_ACTIVE_ALT = 4 )

C  glutSetCursor parameters
      integer*4 GLUT_CURSOR_RIGHT_ARROW
      parameter ( GLUT_CURSOR_RIGHT_ARROW = 0 )
      integer*4 GLUT_CURSOR_LEFT_ARROW
      parameter ( GLUT_CURSOR_LEFT_ARROW = 1 )
      integer*4 GLUT_CURSOR_INFO
      parameter ( GLUT_CURSOR_INFO = 2 )
      integer*4 GLUT_CURSOR_DESTROY
      parameter ( GLUT_CURSOR_DESTROY = 3 )
      integer*4 GLUT_CURSOR_HELP
      parameter ( GLUT_CURSOR_HELP = 4 )
      integer*4 GLUT_CURSOR_CYCLE
```

```
      parameter ( GLUT_CURSOR_CYCLE = 5 )
      integer*4 GLUT_CURSOR_SPRAY
      parameter ( GLUT_CURSOR_SPRAY = 6 )
      integer*4 GLUT_CURSOR_WAIT
      parameter ( GLUT_CURSOR_WAIT = 7 )
      integer*4 GLUT_CURSOR_TEXT
      parameter ( GLUT_CURSOR_TEXT = 8 )
      integer*4 GLUT_CURSOR_CROSSHAIR
      parameter ( GLUT_CURSOR_CROSSHAIR = 9 )
      integer*4 GLUT_CURSOR_UP_DOWN
      parameter ( GLUT_CURSOR_UP_DOWN = 10 )
      integer*4 GLUT_CURSOR_LEFT_RIGHT
      parameter ( GLUT_CURSOR_LEFT_RIGHT = 11 )
      integer*4 GLUT_CURSOR_TOP_SIDE
      parameter ( GLUT_CURSOR_TOP_SIDE = 12 )
      integer*4 GLUT_CURSOR_BOTTOM_SIDE
      parameter ( GLUT_CURSOR_BOTTOM_SIDE = 13 )
      integer*4 GLUT_CURSOR_LEFT_SIDE
      parameter ( GLUT_CURSOR_LEFT_SIDE = 14 )
      integer*4 GLUT_CURSOR_RIGHT_SIDE
      parameter ( GLUT_CURSOR_RIGHT_SIDE = 15 )
      integer*4 GLUT_CURSOR_TOP_LEFT_CORNER
      parameter ( GLUT_CURSOR_TOP_LEFT_CORNER = 16 )
      integer*4 GLUT_CURSOR_TOP_RIGHT_CORNER
      parameter ( GLUT_CURSOR_TOP_RIGHT_CORNER = 17 )
      integer*4 GLUT_CURSOR_BOTTOM_RIGHT_CORNER
      parameter ( GLUT_CURSOR_BOTTOM_RIGHT_CORNER = 18 )
      integer*4 GLUT_CURSOR_BOTTOM_LEFT_CORNER
      parameter ( GLUT_CURSOR_BOTTOM_LEFT_CORNER = 19 )
      integer*4 GLUT_CURSOR_INHERIT
      parameter ( GLUT_CURSOR_INHERIT = 100 )
      integer*4 GLUT_CURSOR_NONE
      parameter ( GLUT_CURSOR_NONE = 101 )
      integer*4 GLUT_CURSOR_FULL_CROSSHAIR
      parameter ( GLUT_CURSOR_FULL_CROSSHAIR = 102 )

C  GLUT functions
      integer*4 glutcreatewindow
      integer*4 glutgetwindow
      integer*4 glutcreatemenu
      integer*4 glutgetmenu
      real glutgetcolor
      integer*4 glutget
      integer*4 glutdeviceget
      integer*4 glutextensionsupported

C  GLUT NULL name
      external glutnull
```

References

[1] Kurt Akeley, "RealityEngine Graphics," *Proceedings of SIGGRAPH '93*, July 1993.

[2] F.C. Crow, "The Origins of the Teapot," *IEEE Computer Graphics and Applications*, January 1987.

[3] Phil Karlton, *OpenGL Graphics with the X Window System*, Ver. 1.0, Silicon Graphics, April 30, 1993.

[4] Mark J. Kilgard, "Going Beyond the MIT Sample Server: The Silicon Graphics X11 Server," *The X Journal*, SIGS Publications, January 1993.

[5] Mark Kilgard, "Programming X Overlay Windows," *The X Journal*, SIGS Publications, July 1993.

[6] Mark Kilgard, "OpenGL and X, Part 2: Using OpenGL with Xlib," *The X Journal*, SIGS Publications, Jan/Feb 1994.

[7] Mark Kilgard, "OpenGL and X, Part 3: Integrating OpenGL with Motif," *The X Journal*, SIGS Publications, Jul/Aug 1994.

[8] Mark Kilgard, "An OpenGL Toolkit," *The X Journal*, SIGS Publications, Nov/Dec 1994.

[9] Jackie Neider, Tom Davis, Mason Woo, *OpenGL Programming Guide: The official guide to learning OpenGL, Release 1*, Addison Wesley, 1993.

[10] OpenGL Architecture Review Board, *OpenGL Reference Manual: The official reference document for OpenGL, Release 1*, Addison Wesley, 1992.

[11] Mark Patrick, George Sachs, *X11 Input Extension Library Specification*, X Consortium Standard, X11R6, April 18, 1994.

[12] Mark Patrick, George Sachs, *X11 Input Extension Protocol Specification*, X Consortium Standard, X11R6, April 17, 1994.

[13] Robert Scheifler, James Gettys, *X Window System: The complete Reference to Xlib, X Protocol, ICCCM, XLFD*, third edition, Digital Press, 1992.

[14] Mark Segal, Kurt Akeley, *The OpenGL™ Graphics System: A Specification*, Version 1.0, Silicon Graphics, June 30, 1992.

[15] Silicon Graphics, *Graphics Library Programming Guide*, Document Number 007-1210-040, 1991.

[16] Silicon Graphics, *Graphics Library Window and Font Library Guide*, Document Number 007-1329-010, 1991.

Index