The OpenGL Utility Toolkit (GLUT) Programming Interface

API Version 2

Mark J. Kilgard
Silicon Graphics, Inc.

August 2, 1995

OpenGL is a trademark of Silicon Grahics, Inc. X Window System is a trademark of X Consortium, Inc. Spaceball is a registered trademark of Spatial Systems Inc.

The author has taken care in preparation of this documentation but makes no expressed or implied warranty of any kind and assumes no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising from the use of information or programs contained herein.

Copyright ©1994 Mark J. Kilgard. All rights reserved.

All rights reserved. No part of this documentation may be reproduced, in any form or by any means, without permission in writing from the author.

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Design Philosophy . . . 2
1.3 API Version 2 . . . 3
1.4 Conventions . . . 3
1.5 Terminology . . . 3

2 Initialization **4**
2.1 glutInit . . . 4
2.2 glutInitWindowPosition, glutInitWindowSize . . . 5
2.3 glutInitDisplayMode . . . 6

3 Beginning Event Processing **6**
3.1 glutMainLoop . . . 6

4 Window Management **7**
4.1 glutCreateWindow . . . 7
4.2 glutCreateSubWindow . . . 7
4.3 glutSetWindow, glutGetWindow . . . 8
4.4 glutDestroyWindow . . . 8
4.5 glutPostRedisplay . . . 8
4.6 glutSwapBuffers . . . 9
4.7 glutPositionWindow . . . 9
4.8 glutReshapeWindow . . . 9
4.9 glutPopWindow, glutPushWindow . . . 10
4.10 glutShowWindow, glutHideWindow, glutIconifyWindow . . . 10
4.11 glutSetWindowTitle, glutSetIconTitle . . . 11

5 Menu Management **11**
5.1 glutCreateMenu . . . 11
5.2 glutSetMenu, glutGetMenu . . . 11
5.3 glutDestroyMenu . . . 12
5.4 glutAddMenuEntry . . . 12
5.5 glutAddSubMenu . . . 12
5.6 glutChangeToMenuEntry . . . 12
5.7 glutChangeToSubMenu . . . 13
5.8 glutRemoveMenuItem . . . 13
5.9 glutAttachMenu, glutDetachMenu . . . 13

6 Callback Registration **14**
6.1 glutDisplayFunc . . . 14
6.2 glutReshapeFunc . . . 15
6.3 glutKeyboardFunc . . . 15
6.4 glutMouseFunc . . . 15
6.5 glutMotionFunc, glutPassiveMotionFunc . . . 16
6.6 glutVisibilityFunc . . . 16
6.7 glutEntryFunc . . . 17
6.8 glutSpecialFunc . . . 17
6.9 glutSpaceballMotionFunc . . . 18
6.10 glutSpaceballRotateFunc . . . 18
6.11 glutSpaceballButtonFunc . . . 19
6.12 glutButtonBoxFunc . . . 19
6.13 glutDialsFunc . . . 19

6.14 glutTabletMotionFunc . . . 20
6.15 glutTabletButtonFunc . . . 20
6.16 glutMenuStateFunc . . . 21
6.17 glutIdleFunc . . . 21
6.18 glutTimerFunc . . . 21

7 Color Index Colormap Management **22**
7.1 glutSetColor . . . 22
7.2 glutGetColor . . . 22
7.3 glutCopyColormap . . . 23

8 State Retrieval **23**
8.1 glutGet . . . 23
8.2 glutDeviceGet . . . 25
8.3 glutExtensionSupported . . . 25

9 Font Rendering **26**
9.1 glutBitmapCharacter . . . 26
9.2 glutStrokeCharacter . . . 26

10 Geometric Shape Rendering **27**
10.1 glutSolidSphere, glutWireSphere . . . 27
10.2 glutSolidCube, glutWireCube . . . 27
10.3 glutSolidCone, glutWireCone . . . 28
10.4 glutSolidTorus, glutWireTorus . . . 28
10.5 glutSolidDodecahedron, glutWireDodecahedron . . . 28
10.6 glutSolidOctahedron, glutWireOctahedron . . . 29
10.7 glutSolidTetrahedron, glutWireTetrahedron . . . 29
10.8 glutSolidIcosahedron, glutWireIcosahedron . . . 29
10.9 glutSolidTeapot, glutWireTeapot . . . 29

11 Usage Advice **30**

12 FORTRAN Binding **30**
12.1 Names for the FORTRAN GLUT Binding . . . 31
12.2 Font Naming Caveat . . . 31
12.3 NULL Callback . . . 31

13 Implementation Issues **31**
13.1 Name Space Conventions . . . 31

A GLUT State **32**
A.1 Types of State . . . 32
A.2 Global State . . . 32
A.3 Window State . . . 32
A.4 Menu State . . . 34

B glut.h ANSI C Header File **34**

C fglut.h FORTRAN Header File **38**

References **43**

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.

An ANSI C implementation of GLUT for the X Window System [10] has been implemented.

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 [5] published in *The X Journal*.

The remainder of this section describes GLUT's design philosophy and usage model. The subsequent sections specify the GLUT routines, grouped by functionality. The final sections discuss usage hints, 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 [11, 7] 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 [1] describes the standard by which OpenGL interacts with the X Window System.

The predecessor to OpenGL is IRIS GL [12, 13]. 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 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 [3] or Motif [4] 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* [6]. 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.

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 character pointers to 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 is 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 permits window system constraints 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/or *menu* to the appropriate window and/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 shapes. 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 continuously calls GLUT callbacks as necessary.

Window Management. These routines create and control GLUT windows.

Menu Management. These routines create and control GLUT 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 shapes 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: 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. Version 1 programs should recompiled with GLUT API version 2 libraries should need no source code changes.

1.4 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>
```

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
```

1.5 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 implemented for. 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.

Dials and button box A sophisticated input device consisting of a pad of buttons and an array of rotating dials.

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 registered, is called.

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.

Multisampling A technique for hardware antialiasing generally available only on expensive 3D graphics hardware. 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.

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 3 axes of rotation and 3 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 twisting or directional force on the ball, translations and rotations are generated.

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 to select more menu choices.

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

Tablet A sophisticated, precised 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; and iconified window is not visible but could be made visible in response to some user action like clipping the window's corresponding icon.

Window system The broad notion of the window system 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. Though the GLUT program can still override this default using window size command line options processed by `glutInit`.

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.

The options parsed 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 behavior. 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` set the *initial display mode*.

Usage

```
void glutInitDisplayMode(GLenum mode);
```

`mode` Display mode, normally the bitwise ORing 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.

Description

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

3 Beginning Event Processing

After a GLUT program has done initial setup like 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 integer identifier for the window. The range of allocated identifiers starts at one. This window identifier can be used when calling `glutSetWindow`.

X Notes

The proper X Inter-Client Communication Conventions Manual (ICCCM) toplevel properties are established. The `WM_COMMAND` property is only established for the first window created.

4.2 glutCreateSubWindow

`glutCreateSubWindow` creates a subwindow of the *current window*.

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 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` will return zero.

4.4 glutDestroyWindow

`glutDestroyWindow` destroys the specified window.

Usage

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

`win` Identifier of GLUT window to destroy.

Description

`glutDestroyWindow` destroy the window specified by `win` and the window's associated OpenGL context and logical colormap (if the window is color index). 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 *current window* as needing to be redisplayed. The next time through `glutMainLoop`, the window's display callback will be called to redisplay the window. Multiple calls to `glutPostRedisplay` before the next display callback opportunity generates only a single redisplay. `glutPostRedisplay` maybe be called within a window's display callback to re-mark that window for redisplay.

Logically, window damage notification for a window is treated as a `glutPostRedisplay` on the damaged window.

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 *current window*. Specifically, `glutSwapBuffers` promotes the contents of the back buffer 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 window 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 `glutPositionWindow` requests are not processed immediately. The request is executed after returning to the main event loop. This allows multiple `glutPositionWindow` and `glutReshapeWindow` 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.

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 `glutReshapeWindow` requests are not processed immediately. The request is executed after returning to the main event loop. This allows multiple `glutReshapeWindow` and `glutPositionWindow` 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.

4.9 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 sub-windows. 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.

4.10 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). `glutHideWindow` will hide the *current window*. `glutIconifyWindow` will iconify a top-level window but GLUT prohibits iconification of a sub-window. 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.

4.11 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 only be called 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. The policy by which the window and icon name are displayed is window system dependent.

5 Menu Management

5.1 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 menu identifier range is separate from the window identifier range so menu and window identifiers are not distinct. Implicitly, the *current menu* is set to the newly created menu.

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

5.2 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 previously *current menu* was destroyed, `glutGetMenu` returns zero.

5.3 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.

5.4 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.

5.5 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.

5.6 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.

5.7 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.

5.8 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.

5.9 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.

6 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.

X Notes

The X GLUT implementation uses the X Input extension [8, 9] 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 the X GLUT implementation 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. Refer to the X implementation itself for details.

6.1 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 a 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. The display callback is called with no parameters. The entire window should be redisplayed in response to the callback.

The redisplay state of a window is set. Multiple posted redisplays for a window are coalesced by GLUT to minimize the number of display callbacks called.

Initially when a window is created, no display callback exists for the window. If a redisplay is posted for the window, it will simply be discarded. It is the responsibility of the programmer to install a display callback for the window before the window is shown. Passing `NULL` to `glutDisplayFunc` will deregister the existing display callback.

6.2 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 the first display callback for a window is triggered *if* a reshape has not previously been triggered for the window. 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)`.

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 called for sub-windows when their size is changed using `glutReshapeWindow`.

6.3 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.

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

6.4 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 only be possible to generate 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.

Passing `NULL` to `glutMouseFunc` disables the generation of mouse callbacks.

6.5 glutMotionFunc, glutPassiveMotionFunc

`glutMotionFunc` and `glutPassiveMotionFunc` set the motion and passive motion callback 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.

6.6 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, ie. until the window's visibility changes, further rendering to the window is discarded.

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

If the visibility callback for a window is disabled and later reenabled, 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 reenable the callback, you are guaranteed the next visibility change will be reported.

6.7 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 Notes

An X implementation of GLUT should generate accurate enter/leave callbacks.

6.8 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.

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: the delete key is generated as an ASCII character.

6.9 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 is ineffectual and 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.

6.10 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.

6.11 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 deteremined 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.

6.12 glutButtonBoxFunc

`glutButtonBoxFunc` sets the dials & 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 dials & button box button callback for the *current window*. The dials & button box button callback for a window is called when the window has dials & button box input focus (normally, when the mouse is in the window) and the user generates dials & button box button presses. The `button` parameter will be the button number (starting at one). The number of available dials & button box buttons can be deteremined 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 dials & button box button callback when a dials & button box device is not available is ineffectual and not an error. In this case, no dials & button box button callbacks will be generated.

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

6.13 glutDialsFunc

`glutDialsFunc` sets the dials & 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 dials & button box dials callback for the *current window*. The dials & button box dials callback for a window is called when the window has dials & button box input focus (normally, when the mouse is in the window) and the user generates dials & button box dial changes. The `dial` parameter will be the dial number (starting at one). The number of available dials & button box dials can be deteremined 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 dials & button box dials callback when a dials & button box device is not available is ineffectual and not an error. In this case, no dials & button box dials callbacks will be generated.

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

6.14 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.

6.15 glutTabletButtonFunc

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

Usage

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

`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 deteremined 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.

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.

6.16 glutMenuStateFunc

`glutMenuStateFunc` sets the global menu state callback.

Usage

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

Description

`glutMenuStateFunc` sets the global menu state callback so a GLUT program can determine when a menu is in use or not. When a menu state 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 `value` parameter when pop-up menus are no longer in use. The `func` parameter names the callback function. Other callbacks continue to operate (except user input callbacks) when pop-up menus are in use so the menu state 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 state callbacks. There is a single menu state callback for GLUT.

When the menu state 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 `glutMenuStateFunc` disables the generation of the menu state callback.

6.17 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. Progams 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.

6.18 glutTimerFunc

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

Usage

```
void glutTimerFunc(unsigned long 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.

7 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 hand loading of colormaps.

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.

7.1 glutSetColor

`glutSetColor` sets the color of a colormap entry.

Usage

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

`ndx` 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 `ndx` color index colormap entry of the *current window*'s logical colormap with the color specified by `red`, `green`, and `blue`. The *current window* should be a color index window. `ndx` should be zero or greater and less than the total number of colormap entries for the window. If the *current window*'s colormap was copied by reference, a `glutSetColor` call will force the duplication of the colormap.

7.2 glutGetColor

`glutGetColor` retrieves a red, green, or blue component for a given color index colormap entry for the *current window*'s logical colormap.

Usage

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

`ndx` 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. `ndx` 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 less than zero or greater or equal to the value returned by `glutGet(GLUT_WINDOW_COLORMAP_SIZE)`, that is if the color index is outside the valid range of color indices.

7.3 glutCopyColormap

`glutCopyColormap` copies the logical colormap 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*. 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.

8 State Retrieval

GLUT maintains a considerable amount of programmer visible state. Some (but not all) of this state may be directly retrieved. Two GLUT state retreival routines return this state. A third GLUT routine helps to easily determine what OpenGL extensions are supported by the underlying OpenGL implementation.

8.1 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_STEREO` One if the *current window* is stereo, zero otherwise.

`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. 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.

8.2 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 dials & button box device. If no dials & button box device is supported, zero is returned.

`GLUT_NUM_DIALS` Number of dials supported by the dials & 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.

8.3 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`.

9 Font Rendering

GLUT supports two type of font rendering: stroke fonts, meaning each glyph is generated as geometry composed of lines; 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.

9.1 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
```

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.

9.2 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 Geometric Shape Rendering

GLUT includes a number of routines for generating easily recognizable geometric shapes. These routines reflect functionality available in the `aux` toolkit 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).

10.1 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.

10.2 glutSolidCube, glutWireCube

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

Usage

```
void glutSolidCube(GLdouble size);
void glutWireTorus(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`.

10.3 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.

10.4 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.

10.5 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 $\sqrt{3}$.

10.6 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.

10.7 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 $\sqrt{3}$.

10.8 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.

10.9 glutSolidTeapot, glutWireTeapot

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

Usage

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

`size` Relative size of the teapot.

11 Usage Advice

Do not change state that will affect the way a window will be drawn in a window's display callback. Your display callback should be idempotent.

If you need to redisplay a window, instead of rendering in whatever callback you happen to be in, call `glutPostRedisplay`. 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.

Avoid using color index mode if possible.

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 in idle and timer callback you explicitly set the window or menu appropriately using `glutSetWindow` and `glutSetMenu`.

If you register a single function as a callback routine for multiple windows, you can call `glutGetWindow` within the callback to determine what window is generating 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.

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 key presses for these keys using the `glutKeyboardFunc` callback, not with the `glutSpecialFunc` 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!

12 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"
```

The GLUT Fortran binding (and the OpenGL and GLU Fortran bindings) assume identifiers are not limited to 6 characters.

12.1 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 ARB's 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.

12.2 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 EXTERNALs, "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
```

12.3 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.

13 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.

13.1 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.

A GLUT State

This appendix specifies precisely what programmer visible state GLUT maintains. There are three categories of programmer visible state GLUT maintains: global, window, and menu. The window and menu state categories are maintained for each created window or menu.

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` 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. Notes for each category of state indicate additional caveats to the element of state.

A.1 Types of State

The programmer visible state maintained by GLUT is of varying types:

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.

Integer An integer value.

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.

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

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
menuStateCallback	Callback	glutMenuEntryFunc	-	NULL
timerList	list of Timer	glutTimerFunc	-	none
screenWidth	Integer	-	GLUT_SCREEN_WIDTH	system dependent
screenHeight	Integer	-	GLUT_SCREEN_HEIGHT	system dependent
screenWidthMM	Integer	-	GLUT_SCREEN_WIDTH_MM	system dependent
screenHeightMM	Integer	-	GLUT_SCREEN_HEIGHT_MM	system dependent

(1) The currentWindow is also changed implicitly by every window 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).

A.3 Window State

For the purposes of listing the window state elements, window state is classified into two types: generic state and frame buffer capability 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 Generic 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
stacking	Stacking	glutPopWindow glutPushWindow	-	top
displayState	State (6)	glutShowWindow (7) glutHideWindow glutIconifyWindow	-	shown
visibility	Visibility	(8)	(9)	undefined
redisplay	Boolean	glutPostRedisplay (10)	-	False
toplevel: windowTitle	String	glutWindowTitle	-	glutCreateWindow
toplevel: iconTitle	String	glutIconTitle	-	glutCreateWindow
displayCallback	Callback	glutDisplayFunc	-	NULL
reshapeCallback	Callback	glutReshapeFunc	-	NULL (11)
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: (12)
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) Subwindows can not be iconified.

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

(8) 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.

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

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

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

(12) 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 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.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. */

/* 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.
 *
 */
#ifndef GLUT_API_VERSION                /* allow this to be overriden */
#define GLUT_API_VERSION                2
#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
 *
 */
#ifndef GLUT_XLIB_IMPLEMENTATION        /* allow this to be overriden */
```

```
48  #define GLUT_XLIB_IMPLEMENTATION        4
49  #endif

50  /* display mode bit masks */
51  #define GLUT_RGB                        0
52  #define GLUT_RGBA                       GLUT_RGB
53  #define GLUT_INDEX                      1
54  #define GLUT_SINGLE                     0
55  #define GLUT_DOUBLE                     2
56  #define GLUT_ACCUM                      4
57  #define GLUT_ALPHA                      8
58  #define GLUT_DEPTH                      16
59  #define GLUT_STENCIL                    32
60  #if (GLUT_API_VERSION >= 2)
61  #define GLUT_MULTISAMPLE                128
62  #define GLUT_STEREO                     256
63  #endif

64  /* mouse buttons */
65  #define GLUT_LEFT_BUTTON                0
66  #define GLUT_MIDDLE_BUTTON              1
67  #define GLUT_RIGHT_BUTTON               2

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

71  #if (GLUT_API_VERSION >= 2)
72  /* function keys */
73  #define GLUT_KEY_F1                     1
74  #define GLUT_KEY_F2                     2
75  #define GLUT_KEY_F3                     3
76  #define GLUT_KEY_F4                     4
77  #define GLUT_KEY_F5                     5
78  #define GLUT_KEY_F6                     6
79  #define GLUT_KEY_F7                     7
80  #define GLUT_KEY_F8                     8
81  #define GLUT_KEY_F9                     9
82  #define GLUT_KEY_F10                    10
83  #define GLUT_KEY_F11                    11
84  #define GLUT_KEY_F12                    12
85  /* directional keys */
86  #define GLUT_KEY_LEFT                   100
87  #define GLUT_KEY_UP                     101
88  #define GLUT_KEY_RIGHT                  102
89  #define GLUT_KEY_DOWN                   103
90  #define GLUT_KEY_PAGE_UP                104
91  #define GLUT_KEY_PAGE_DOWN              105
92  #define GLUT_KEY_HOME                   106
93  #define GLUT_KEY_END                    107
94  #define GLUT_KEY_INSERT                 108
95  #endif

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

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

102 /* visibility callback state */
103 #define GLUT_NOT_VISIBLE                0
104 #define GLUT_VISIBLE                    1

105 /* color index component selection values */
106 #define GLUT_RED                        0
107 #define GLUT_GREEN                      1
```

```
108  #define GLUT_BLUE                       2

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

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

115  /* bitmap font opaque addresses (use constants instead in source code) */
116  extern void *glutBitmap9By15;
117  extern void *glutBitmap8By13;
118  extern void *glutBitmapTimesRoman10;
119  extern void *glutBitmapTimesRoman24;

120  /* bitmap font constants (use these in GLUT program) */
121  #define GLUT_BITMAP_9_BY_15             (&glutBitmap9By15)
122  #define GLUT_BITMAP_8_BY_13             (&glutBitmap8By13)
123  #define GLUT_BITMAP_TIMES_ROMAN_10      (&glutBitmapTimesRoman10)
124  #define GLUT_BITMAP_TIMES_ROMAN_24      (&glutBitmapTimesRoman24)

125  /* glutGet parameters */
126  #define GLUT_WINDOW_X                   100
127  #define GLUT_WINDOW_Y                   101
128  #define GLUT_WINDOW_WIDTH               102
129  #define GLUT_WINDOW_HEIGHT              103
130  #define GLUT_WINDOW_BUFFER_SIZE         104
131  #define GLUT_WINDOW_STENCIL_SIZE        105
132  #define GLUT_WINDOW_DEPTH_SIZE          106
133  #define GLUT_WINDOW_RED_SIZE            107
134  #define GLUT_WINDOW_GREEN_SIZE          108
135  #define GLUT_WINDOW_BLUE_SIZE           109
136  #define GLUT_WINDOW_ALPHA_SIZE          110
137  #define GLUT_WINDOW_ACCUM_RED_SIZE      111
138  #define GLUT_WINDOW_ACCUM_GREEN_SIZE    112
139  #define GLUT_WINDOW_ACCUM_BLUE_SIZE     113
140  #define GLUT_WINDOW_ACCUM_ALPHA_SIZE    114
141  #define GLUT_WINDOW_DOUBLEBUFFER        115
142  #define GLUT_WINDOW_RGBA                116
143  #define GLUT_WINDOW_PARENT              117
144  #define GLUT_WINDOW_NUM_CHILDREN        118
145  #define GLUT_WINDOW_COLORMAP_SIZE       119
146  #if (GLUT_API_VERSION >= 2)
147  #define GLUT_WINDOW_NUM_SAMPLES         120
148  #define GLUT_WINDOW_STEREO              121
149  #endif
150  #define GLUT_SCREEN_WIDTH               200
151  #define GLUT_SCREEN_HEIGHT              201
152  #define GLUT_SCREEN_WIDTH_MM            202
153  #define GLUT_SCREEN_HEIGHT_MM           203
154  #define GLUT_MENU_NUM_ITEMS             300
155  #define GLUT_DISPLAY_MODE_POSSIBLE      400
156  #define GLUT_INIT_WINDOW_X              500
157  #define GLUT_INIT_WINDOW_Y              501
158  #define GLUT_INIT_WINDOW_WIDTH          502
159  #define GLUT_INIT_WINDOW_HEIGHT         503
160  #define GLUT_INIT_DISPLAY_MODE          504
161  #if (GLUT_API_VERSION >= 2)
162  #define GLUT_ELAPSED_TIME               700
163  #endif

164  #if (GLUT_API_VERSION >= 2)
165  /* glutDeviceGet parameters */
166  #define GLUT_HAS_KEYBOARD               600
167  #define GLUT_HAS_MOUSE                  601
168  #define GLUT_HAS_SPACEBALL              602
169  #define GLUT_HAS_DIAL_AND_BUTTON_BOX    603
```

```
170  #define GLUT_HAS_TABLET                 604
171  #define GLUT_NUM_MOUSE_BUTTONS          605
172  #define GLUT_NUM_SPACEBALL_BUTTONS      606
173  #define GLUT_NUM_BUTTON_BOX_BUTTONS     607
174  #define GLUT_NUM_DIALS                  608
175  #define GLUT_NUM_TABLET_BUTTONS         609
176  #endif

177  /* GLUT initialization sub-API */
178  extern void glutInit (int *, char **);
179  extern void glutInitDisplayMode (unsigned long);
180  extern void glutInitWindowPosition (int, int);
181  extern void glutInitWindowSize (int, int);
182  extern void glutMainLoop (void);

183  /* GLUT window sub-api */
184  extern int glutCreateWindow (char *);
185  extern int glutCreateSubWindow (int, int, int, int, int);
186  extern void glutDestroyWindow (int);
187  extern void glutPostRedisplay (void);
188  extern void glutSwapBuffers (void);
189  extern int glutGetWindow (void);
190  extern void glutSetWindow (int);
191  extern void glutSetWindowTitle (char *);
192  extern void glutSetIconTitle (char *);
193  extern void glutPositionWindow (int, int);
194  extern void glutReshapeWindow (int, int);
195  extern void glutPopWindow (void);
196  extern void glutPushWindow (void);
197  extern void glutIconifyWindow (void);
198  extern void glutShowWindow (void);
199  extern void glutHideWindow (void);

200  /* GLUT menu sub-API */
201  extern int glutCreateMenu (void (*)(int));
202  extern void glutDestroyMenu (int);
203  extern int glutGetMenu (void);
204  extern void glutSetMenu (int);
205  extern void glutAddMenuEntry (char *, int);
206  extern void glutAddSubMenu (char *, int);
207  extern void glutChangeToMenuEntry (int, char *, int);
208  extern void glutChangeToSubMenu (int, char *, int);
209  extern void glutRemoveMenuItem (int);
210  extern void glutAttachMenu (int);
211  extern void glutDetachMenu (int);

212  /* GLUT callback sub-api */
213  extern void glutDisplayFunc (void (*)(void));
214  extern void glutReshapeFunc (void (*)(int, int));
215  extern void glutKeyboardFunc (void (*)(unsigned char, int, int));
216  extern void glutMouseFunc (void (*)(int, int, int, int));
217  extern void glutMotionFunc (void (*)(int, int));
218  extern void glutPassiveMotionFunc (void (*)(int, int));
219  extern void glutEntryFunc (void (*)(int));
220  extern void glutVisibilityFunc (void (*)(int));
221  extern void glutIdleFunc (void (*)(void));
222  extern void glutTimerFunc (unsigned long, void (*)(int), int);
223  extern void glutMenuStateFunc (void (*)(int));
224  #if (GLUT_API_VERSION >= 2)
225  extern void glutSpecialFunc (void (*)(int, int, int));
226  extern void glutSpaceballMotionFunc (void (*)(int, int, int));
227  extern void glutSpaceballRotateFunc (void (*)(int, int, int));
228  extern void glutSpaceballButtonFunc (void (*)(int, int));
229  extern void glutButtonBoxFunc (void (*)(int, int));
230  extern void glutDialsFunc (void (*)(int, int));
231  extern void glutTabletMotionFunc (void (*)(int, int));
232  extern void glutTabletButtonFunc (void (*)(int, int, int, int));
233  #endif
```

```
234 /* GLUT color index sub-api */
235 extern void glutSetColor (int, GLfloat, GLfloat, GLfloat);
236 extern GLfloat glutGetColor (int, int);
237 extern void glutCopyColormap (int);

238 /* GLUT state retrieval sub-api */
239 extern int glutGet (GLenum);
240 extern int glutDeviceGet (GLenum);

241 /* GLUT font sub-API */
242 extern void glutStrokeCharacter (void *, int);
243 extern void glutBitmapCharacter (void *, int);

244 /* GLUT pre-built models sub-API */
245 extern void glutWireSphere (GLdouble, GLint, GLint);
246 extern void glutSolidSphere (GLdouble, GLint, GLint);
247 extern void glutWireCone (GLdouble, GLdouble, GLint, GLint);
248 extern void glutSolidCone (GLdouble, GLdouble, GLint, GLint);
249 extern void glutWireCube (GLdouble);
250 extern void glutSolidCube (GLdouble);
251 extern void glutWireTorus (GLdouble, GLdouble, GLint, GLint);
252 extern void glutSolidTorus (GLdouble, GLdouble, GLint, GLint);
253 extern void glutWireDodecahedron (void);
254 extern void glutSolidDodecahedron (void);
255 extern void glutWireTeapot (GLdouble);
256 extern void glutSolidTeapot (GLdouble);
257 extern void glutWireOctahedron (void);
258 extern void glutSolidOctahedron (void);
259 extern void glutWireTetrahedron (void);
260 extern void glutSolidTetrahedron (void);
261 extern void glutWireIcosahedron (void);
262 extern void glutSolidIcosahedron (void);

263 #if (GLUT_API_VERSION >= 2)
264 /* GLUT extension support sub-API */
265 extern int glutExtensionSupported (char *);
266 #endif

267 #ifdef __cplusplus
268 }
269 #endif

270 #endif /* __glut_h__ */
```

C fglut.h FORTRAN Header File

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

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

 5 C  GLUT Fortran header file

 6 C  display mode bit masks
 7       integer*4 GLUT_RGB
 8       parameter ( GLUT_RGB = 0 )
 9       integer*4 GLUT_RGBA
10       parameter ( GLUT_RGBA = 0 )
11       integer*4 GLUT_INDEX
12       parameter ( GLUT_INDEX = 1 )
13       integer*4 GLUT_SINGLE
14       parameter ( GLUT_SINGLE = 0 )
15       integer*4 GLUT_DOUBLE
16       parameter ( GLUT_DOUBLE = 2 )
```

```
17          integer*4 GLUT_ACCUM
18          parameter ( GLUT_ACCUM = 4 )
19          integer*4 GLUT_ALPHA
20          parameter ( GLUT_ALPHA = 8 )
21          integer*4 GLUT_DEPTH
22          parameter ( GLUT_DEPTH = 16 )
23          integer*4 GLUT_STENCIL
24          parameter ( GLUT_STENCIL = 32 )
25          integer*4 GLUT_MULTISAMPLE
26          parameter ( GLUT_MULTISAMPLE = 128 )
27          integer*4 GLUT_STEREO
28          parameter ( GLUT_STEREO = 256 )

29  C  mouse buttons
30          integer*4 GLUT_LEFT_BUTTON
31          parameter ( GLUT_LEFT_BUTTON = 0 )
32          integer*4 GLUT_MIDDLE_BUTTON
33          parameter ( GLUT_MIDDLE_BUTTON = 1 )
34          integer*4 GLUT_RIGHT_BUTTON
35          parameter ( GLUT_RIGHT_BUTTON = 2 )

36  C  mouse button callback state
37          integer*4 GLUT_DOWN
38          parameter ( GLUT_DOWN = 0 )
39          integer*4 GLUT_UP
40          parameter ( GLUT_UP = 1 )

41  C  special key callback values
42          integer*4 GLUT_KEY_F1
43          parameter ( GLUT_KEY_F1 = 1 )
44          integer*4 GLUT_KEY_F2
45          parameter ( GLUT_KEY_F2 = 2 )
46          integer*4 GLUT_KEY_F3
47          parameter ( GLUT_KEY_F3 = 3 )
48          integer*4 GLUT_KEY_F4
49          parameter ( GLUT_KEY_F4 = 4 )
50          integer*4 GLUT_KEY_F5
51          parameter ( GLUT_KEY_F5 = 5 )
52          integer*4 GLUT_KEY_F6
53          parameter ( GLUT_KEY_F6 = 6 )
54          integer*4 GLUT_KEY_F7
55          parameter ( GLUT_KEY_F7 = 7 )
56          integer*4 GLUT_KEY_F8
57          parameter ( GLUT_KEY_F8 = 8 )
58          integer*4 GLUT_KEY_F9
59          parameter ( GLUT_KEY_F9 = 9 )
60          integer*4 GLUT_KEY_F10
61          parameter ( GLUT_KEY_F10 = 10 )
62          integer*4 GLUT_KEY_F11
63          parameter ( GLUT_KEY_F11 = 11 )
64          integer*4 GLUT_KEY_F12
65          parameter ( GLUT_KEY_F12 = 12 )
66          integer*4 GLUT_KEY_LEFT
67          parameter ( GLUT_KEY_LEFT = 100 )
68          integer*4 GLUT_KEY_UP
69          parameter ( GLUT_KEY_UP = 101 )
70          integer*4 GLUT_KEY_RIGHT
71          parameter ( GLUT_KEY_RIGHT = 102 )
72          integer*4 GLUT_KEY_DOWN
73          parameter ( GLUT_KEY_DOWN = 103 )
74          integer*4 GLUT_KEY_PAGE_UP
75          parameter ( GLUT_KEY_PAGE_UP = 104 )
76          integer*4 GLUT_KEY_PAGE_DOWN
77          parameter ( GLUT_KEY_PAGE_DOWN = 105 )
78          integer*4 GLUT_KEY_HOME
79          parameter ( GLUT_KEY_HOME = 106 )
80          integer*4 GLUT_KEY_END
81          parameter ( GLUT_KEY_END = 107 )
```

```
 82          integer*4 GLUT_KEY_INSERT
 83          parameter ( GLUT_KEY_INSERT = 108 )

 84  C  entry/exit callback state
 85          integer*4 GLUT_LEFT
 86          parameter ( GLUT_LEFT = 0 )
 87          integer*4 GLUT_ENTERED
 88          parameter ( GLUT_ENTERED = 1 )

 89  C  menu usage callback state
 90          integer*4 GLUT_MENU_NOT_IN_USE
 91          parameter ( GLUT_MENU_NOT_IN_USE = 0 )
 92          integer*4 GLUT_MENU_IN_USE
 93          parameter ( GLUT_MENU_IN_USE = 1 )

 94  C  visibility callback state
 95          integer*4 GLUT_NOT_VISIBLE
 96          parameter ( GLUT_NOT_VISIBLE = 0 )
 97          integer*4 GLUT_VISIBLE
 98          parameter ( GLUT_VISIBLE = 1 )

 99  C  color index component selection values
100          integer*4 GLUT_RED
101          parameter ( GLUT_RED = 0 )
102          integer*4 GLUT_GREEN
103          parameter ( GLUT_GREEN = 1 )
104          integer*4 GLUT_BLUE
105          parameter ( GLUT_BLUE = 2 )

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

112  C  stroke font opaque names
113  C       external GLUT_STROKE_ROMAN
114  C       external GLUT_STROKE_MONO_ROMAN

115  C  bitmap font opaque names
116  C       external GLUT_BITMAP_9_BY_15
117  C       external GLUT_BITMAP_8_BY_13
118  C       external GLUT_BITMAP_TIMES_ROMAN_10
119  C       external GLUT_BITMAP_TIMES_ROMAN_24

120  C  glutGet parameters
121          integer*4 GLUT_WINDOW_X
122          parameter ( GLUT_WINDOW_X = 100 )
123          integer*4 GLUT_WINDOW_Y
124          parameter ( GLUT_WINDOW_Y = 101 )
125          integer*4 GLUT_WINDOW_WIDTH
126          parameter ( GLUT_WINDOW_WIDTH = 102 )
127          integer*4 GLUT_WINDOW_HEIGHT
128          parameter ( GLUT_WINDOW_HEIGHT = 103 )
129          integer*4 GLUT_WINDOW_BUFFER_SIZE
130          parameter ( GLUT_WINDOW_BUFFER_SIZE = 104 )
131          integer*4 GLUT_WINDOW_STENCIL_SIZE
132          parameter ( GLUT_WINDOW_STENCIL_SIZE = 105 )
133          integer*4 GLUT_WINDOW_DEPTH_SIZE
134          parameter ( GLUT_WINDOW_DEPTH_SIZE = 106 )
135          integer*4 GLUT_WINDOW_RED_SIZE
136          parameter ( GLUT_WINDOW_RED_SIZE = 107 )
137          integer*4 GLUT_WINDOW_GREEN_SIZE
138          parameter ( GLUT_WINDOW_GREEN_SIZE = 108 )
139          integer*4 GLUT_WINDOW_BLUE_SIZE
140          parameter ( GLUT_WINDOW_BLUE_SIZE = 109 )
141          integer*4 GLUT_WINDOW_ALPHA_SIZE
```

```
142           parameter ( GLUT_WINDOW_ALPHA_SIZE = 110 )
143           integer*4 GLUT_WINDOW_ACCUM_RED_SIZE
144           parameter ( GLUT_WINDOW_ACCUM_RED_SIZE = 111 )
145           integer*4 GLUT_WINDOW_ACCUM_GREEN_SIZE
146           parameter ( GLUT_WINDOW_ACCUM_GREEN_SIZE = 112 )
147           integer*4 GLUT_WINDOW_ACCUM_BLUE_SIZE
148           parameter ( GLUT_WINDOW_ACCUM_BLUE_SIZE = 113 )
149           integer*4 GLUT_WINDOW_ACCUM_ALPHA_SIZE
150           parameter ( GLUT_WINDOW_ACCUM_ALPHA_SIZE = 114 )
151           integer*4 GLUT_WINDOW_DOUBLEBUFFER
152           parameter ( GLUT_WINDOW_DOUBLEBUFFER = 115 )
153           integer*4 GLUT_WINDOW_RGBA
154           parameter ( GLUT_WINDOW_RGBA = 116 )
155           integer*4 GLUT_WINDOW_PARENT
156           parameter ( GLUT_WINDOW_PARENT = 117 )
157           integer*4 GLUT_WINDOW_NUM_CHILDREN
158           parameter ( GLUT_WINDOW_NUM_CHILDREN = 118 )
159           integer*4 GLUT_WINDOW_COLORMAP_SIZE
160           parameter ( GLUT_WINDOW_COLORMAP_SIZE = 119 )
161           integer*4 GLUT_WINDOW_NUM_SAMPLES
162           parameter ( GLUT_WINDOW_NUM_SAMPLES = 120 )
163           integer*4 GLUT_WINDOW_STEREO
164           parameter ( GLUT_WINDOW_STEREO = 121 )
165           integer*4 GLUT_SCREEN_WIDTH
166           parameter ( GLUT_SCREEN_WIDTH = 200 )
167           integer*4 GLUT_SCREEN_HEIGHT
168           parameter ( GLUT_SCREEN_HEIGHT = 201 )
169           integer*4 GLUT_SCREEN_WIDTH_MM
170           parameter ( GLUT_SCREEN_WIDTH_MM = 202 )
171           integer*4 GLUT_SCREEN_HEIGHT_MM
172           parameter ( GLUT_SCREEN_HEIGHT_MM = 203 )
173           integer*4 GLUT_MENU_NUM_ITEMS
174           parameter ( GLUT_MENU_NUM_ITEMS = 300 )
175           integer*4 GLUT_DISPLAY_MODE_POSSIBLE
176           parameter ( GLUT_DISPLAY_MODE_POSSIBLE = 400 )
177           integer*4 GLUT_INIT_WINDOW_X
178           parameter ( GLUT_INIT_WINDOW_X = 500 )
179           integer*4 GLUT_INIT_WINDOW_Y
180           parameter ( GLUT_INIT_WINDOW_Y = 501 )
181           integer*4 GLUT_INIT_WINDOW_WIDTH
182           parameter ( GLUT_INIT_WINDOW_WIDTH = 502 )
183           integer*4 GLUT_INIT_WINDOW_HEIGHT
184           parameter ( GLUT_INIT_WINDOW_HEIGHT = 503 )
185           integer*4 GLUT_INIT_DISPLAY_MODE
186           parameter ( GLUT_INIT_DISPLAY_MODE = 504 )
187           integer*4 GLUT_ELAPSED_TIME
188           parameter ( GLUT_ELAPSED_TIME = 700 )

189   C  glutDeviceGet parameters
190           integer*4 GLUT_HAS_KEYBOARD
191           parameter ( GLUT_HAS_KEYBOARD = 600 )
192           integer*4 GLUT_HAS_MOUSE
193           parameter ( GLUT_HAS_MOUSE = 601 )
194           integer*4 GLUT_HAS_SPACEBALL
195           parameter ( GLUT_HAS_SPACEBALL = 602 )
196           integer*4 GLUT_HAS_DIAL_AND_BUTTON_BOX
197           parameter ( GLUT_HAS_DIAL_AND_BUTTON_BOX = 603 )
198           integer*4 GLUT_HAS_TABLET
199           parameter ( GLUT_HAS_TABLET = 604 )
200           integer*4 GLUT_NUM_MOUSE_BUTTONS
201           parameter ( GLUT_NUM_MOUSE_BUTTONS = 605 )
202           integer*4 GLUT_NUM_SPACEBALL_BUTTONS
203           parameter ( GLUT_NUM_SPACEBALL_BUTTONS = 606 )
204           integer*4 GLUT_NUM_BUTTON_BOX_BUTTONS
205           parameter ( GLUT_NUM_BUTTON_BOX_BUTTONS = 607 )
206           integer*4 GLUT_NUM_DIALS
207           parameter ( GLUT_NUM_DIALS = 608 )
208           integer*4 GLUT_NUM_TABLET_BUTTONS
```

```
209           parameter ( GLUT_NUM_TABLET_BUTTONS = 609 )

210   C  GLUT functions
211           integer*4 glutcreatewindow
212           integer*4 glutgetwindow
213           integer*4 glutcreatemenu
214           integer*4 glutgetmenu
215           real glutgetcolor
216           integer*4 glutget
217           integer*4 glutdeviceget
218           integer*4 glutextensionsupported

219   C  GLUT NULL name
220           external glutnull
```

References

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

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

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

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

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

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

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

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

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

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

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

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

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