Appendix D

Basics of GLUT: The OpenGL Utility Toolkit

This appendix describes a subset of Mark Kilgard's OpenGL Utility Toolkit (GLUT), which is fully documented in his book, *OpenGL Programming for the X Window System* (Reading, MA: Addison-Wesley Developers Press, 1996). GLUT has become a popular library for OpenGL programmers, because it standardizes and simplifies window and event management. GLUT has been ported atop a variety of OpenGL implementations, including both the X Window System and Microsoft Windows NT.

This appendix has the following major sections:

- "Initializing and Creating a Window"
- "Handling Window and Input Events"
- "Loading the Color Map"
- "Initializing and Drawing Three-Dimensional Objects"
- "Managing a Background Process"
- "Running the Program"

(See "How to Obtain the Sample Code" in the Preface for information about how to obtain the source code for GLUT.)

With GLUT, your application structures its event handling to use callback functions. (This method is similar to using the Xt Toolkit, also known as the X Intrinsics, with a widget set.) For example, first you open a window and register callback routines for specific events. Then, you create a main loop without an exit. In that loop, if an event occurs, its registered callback functions are executed. Upon completion of the callback functions, flow of control is returned to the main loop.

Initializing and Creating a Window

Before you can open a window, you must specify its characteristics: Should it be single-buffered or double-buffered? Should it store colors as RGBA values or as color indices? Where should it appear on your display? To specify the answers to these questions, call glutInit(), glutInitDisplayMode(), glutInitWindowSize(), and glutInitWindowPosition() before you call glutCreateWindow() to open the window.

---

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

---

glutInit() should be called before any other GLUT routine, because it initializes the GLUT library. glutInit() will also process command line options, but the specific options are window system dependent. For the X Window System, -iconic, -geometry, and -display are examples of command line options, processed by glutInit(). (The parameters to the glutInit() should be the same as those to main().)

---

void glutInitDisplayMode(unsigned int *mode*);

Specifies a display mode (such as RGBA or color-index, or single- or double-buffered) for windows created when glutCreateWindow() is called. You can also specify that the window have an associated depth, stencil, and/or accumulation buffer. The *mask* argument is a bitwise ORed combination of GLUT_RGBA or GLUT_INDEX, GLUT_SINGLE or GLUT_DOUBLE, and any of the buffer-enabling flags: GLUT_DEPTH, GLUT_STENCIL, or GLUT_ACCUM. For example, for a double-buffered, RGBA-mode window with a depth and stencil buffer, use GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH | GLUT_STENCIL. The default value is GLUT_RGBA | GLUT_SINGLE (an RGBA, single-buffered window).

---

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

Requests windows created by glutCreateWindow() to have an initial size and position. The arguments (*x*, *y*) indicate the location of a corner of the window, relative

to the entire display. The *width* and *height* indicate the window's size (in pixels). The initial window size and position are hints and may be overridden by other requests.

int glutCreateWindow(char ***name*);

Opens a window with previously set characteristics (display mode, width, height, and so on). The string *name* may appear in the title bar if your window system does that sort of thing. The window is not initially displayed until glutMainLoop() is entered, so do not render into the window until then.

The value returned is a unique integer identifier for the window. This identifier can be used for controlling and rendering to multiple windows (each with an OpenGL rendering context) from the same application.

Handling Window and Input Events

After the window is created, but before you enter the main loop, you should register callback functions using the following routines.

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

Specifies the function that's called whenever the contents of the window need to be redrawn. The contents of the window may need to be redrawn when the window is initially opened, when the window is popped and window damage is exposed, and when glutPostRedisplay() is explicitly called.

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

Specifies the function that's called whenever the window is resized or moved. The argument *func* is a pointer to a function that expects two arguments, the new width and height of the window. Typically, *func* calls glViewport(), so that the display is clipped to the new size, and it redefines the projection matrix so that the aspect ratio of the projected image matches the viewport, avoiding aspect ratio distortion. If glutReshapeFunc() isn't called or is deregistered by passing NULL, a default reshape function is called, which calls glViewport(*0*, *0*, *width*, *height*).

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

Specifies the function, *func*, that's called when a key that generates an ASCII character is pressed. The *key* callback parameter is the generated ASCII value. The *x* and *y* callback parameters indicate the location of the mouse (in window-relative coordinates) when the key was pressed.

void glutMouseFunc(void (*_func_)(int _button_, int _state_, int _x_, int _y_));

Specifies the function, *func*, that's called when a mouse button is pressed or released. The *button* callback parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON. The *state* callback parameter is either GLUT_UP or GLUT_DOWN, depending upon whether the mouse has been released or pressed. The *x* and *y* callback parameters indicate the location (in window-relative coordinates) of the mouse when the event occurred.

void glutMotionFunc(void (*_func_)(int _x_, int _y_));

Specifies the function, *func*, that's called when the mouse pointer moves within the window while one or more mouse buttons is pressed. The *x* and *y* callback parameters indicate the location (in window-relative coordinates) of the mouse when the event occurred.

void glutPostRedisplay(void);

Marks the current window as needing to be redrawn. At the next opportunity, the callback function registered by glutDisplayFunc() will be called.

Loading the Color Map

If you're using color-index mode, you might be surprised to discover there's no OpenGL routine to load a color into a color lookup table. This is because the process of loading a color map depends entirely on the window system. GLUT provides a generalized routine to load a single color index with an RGB value, glutSetColor().

void glutSetColor(GLint *index*, GLfloat *red*, GLfloat *green*, GLfloat *blue*);

Loads the index in the color map, *index*, with the given *red*, *green*, and *blue* values. These values are normalized to lie in the range [0.0,1.0].

Initializing and Drawing Three-Dimensional Objects

Many sample programs in this guide use three-dimensional models to illustrate various rendering properties. The following drawing routines are included in GLUT to avoid having to reproduce the code to draw these models in each program. The routines render all their graphics in immediate mode. Each three-dimensional model comes in two flavors: wireframe without surface normals, and solid with shading and surface normals.

Use the solid version when you're applying lighting. Only the teapot generates texture coordinates.

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

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

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

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

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

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

```
void glutWireDodecahedron(GLdouble radius);
void glutSolidDodecahedron(GLdouble radius);
```

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

void glutSolidCone(GLdouble *radius*, GLdouble *height*, GLint *slices*,
GLint *stacks*);

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

Managing a Background Process

You can specify a function that's to be executed if no other events are pending—for example, when the event loop would otherwise be idle—with glutIdleFunc(). This is particularly useful for continuous animation or other background processing.

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

Specifies the function, *func*, to be executed if no other events are pending. If NULL (zero) is passed in, execution of *func* is disabled.

Running the Program

After all the setup is completed, GLUT programs enter an event processing loop, glutMainLoop().

```
void glutMainLoop(void);
```

Enters the GLUT processing loop, never to return. Registered callback functions will be called when the corresponding events instigate them.