Appendix E

The OpenGL Programming Guide Auxiliary Library

This appendix describes the auxiliary library that was written using OpenGL for this guide. It has the following major sections:

- "**Initializing and Exiting 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**" for information about how to obtain the source code for the auxiliary library.

With the auxiliary library, 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 Exiting 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 *auxInitDisplayMode()* and *auxInitPosition()* before you call *auxInitWindow()* to open the window.

void *auxInitWindow*(GLbyte **titleString*);

Opens a window with the characteristics specified by *auxInitDisplayMode()* and *auxInitPosition()*. The string *titleString* appears in the title bar, if your window system does that sort of thing. The Escape key is bound to an exiting function that kills the window, exits the program, and generally cleans up. Also, the default color for the background is set to black for an RGBA window and to color index 0 for a color−index window.

void *auxInitDisplayMode*(GLbitfield *mask*);

Tells *auxInitWindow()* whether to create an RGBA or color−index window, or a single− or double−buffered window. 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 AUX_RGBA or AUX_INDEX, AUX_SINGLE or AUX_DOUBLE, and any of the buffer−enabling flags: AUX_DEPTH, AUX_STENCIL, or AUX_ACCUM. For example, for a double−buffered, RGBA−mode window with a depth and stencil buffer, use AUX_DOUBLE | AUX_RGBA | AUX_DEPTH | AUX_STENCIL. The default value is AUX_INDEX | AUX_SINGLE, or a color−index, single−buffered window.

void *auxInitPosition*(GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*);

Tells *auxInitWindow()* where to position a window on the screen. The arguments (*x*, *y*) indicate the location of the lower left corner of the window, and *width* and *height* indicate the window's size (in pixels). The default values are (0, 0) for (*x*, *y*) and (100, 100) for *(width*, *height*).

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 three routines.

void *auxReshapeFunc*(void (**function*)(GLsizei, GLsizei));

Specifies the function that's called whenever the window is resized, moved, or exposed. The argument *function* is a pointer to a function that expects two arguments, the new width and height of the window. Typically, *function* 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 you don't call *auxReshapeFunc()*, a default reshape function is called, which assumes a two−dimensional orthographic projection. With this auxiliary library, the window is automatically redrawn after every reshaping event.

void *auxKeyFunc*(GLint *key*, void (**function*)(void)) ;

Specifies the function, *function*, that's called when the keyboard key indicated by *key* is pressed. Use one of the defined auxiliary library constants for *key*: AUX_A through AUX_Z, AUX_a through AUX_z, AUX_0 through AUX_9, AUX_LEFT, AUX_RIGHT, AUX_UP, AUX_DOWN (the arrow keys), AUX_ESCAPE, AUX_SPACE, or AUX_RETURN. With this auxiliary library, the window is automatically redrawn after every processed key event, although in a real application, you might wait for several events to be completed before drawing.

void *auxMouseFunc*(GLint *button*, GLint *mode*,
void (**function*)(AUX_EVENTREC *));

Specifies the function, *function*, that's called when the mouse button indicated by *button* enters the mode defined by *mode*. The *button* argument can be AUX_LEFTBUTTON, AUX_MIDDLEBUTTON, or AUX_RIGHTBUTTON (assuming a right−handed setup). The *mode* argument indicates whether the button is clicked, AUX_MOUSEDOWN, or released, AUX_MOUSEUP. The *function* argument must take one argument, which is a pointer to a structure of type AUX_EVENTREC. The *auxMouseFunc()* routine allocates memory for the structure. For example, to determine the pointer coordinates at the time of the event, you might define *function* like this:

```
void function(AUX_EVENTREC *event)
{ GLint x, y;
    x = event->data[AUX_MOUSEX];
    y = event->data[AUX_MOUSEY];
     ...
}
```

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. The auxiliary library provides a generalized routine to load a single color index with an RGB value, *auxSetOneColor()*. You need to implement this routine for your particular system.

void *auxSetOneColor*(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 the auxiliary library to avoid having to reproduce the code to draw these models in each program. 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. The argument for these routines allows you to scale the object that's drawn.

void *auxWireSphere*(GLdouble *radius*);
void *auxSolidSphere*(GLdouble *radius*);

void *auxWireCube*(GLdouble *size*);
void *auxSolidCube*(GLdouble *size*);

void *auxWireBox*(GLdouble *width*, GLdouble *height*, GLdouble *depth*);
void *auxSolidBox*(GLdouble *width*, GLdouble *height*, GLdouble *depth*);

void *auxWireTorus*(GLdouble *innerRadius*, GLdouble *outerRadius*);
void *auxSolidTorus*(GLdouble *innerRadius*, GLdouble *outerRadius*);

void *auxWireCylinder*(GLdouble *radius*, GLdouble *height*);
void *auxSolidCylinder*(GLdouble *radius*, GLdouble *height*);

void *auxWireIcosahedron*(GLdouble *radius*);
void *auxSolidIcosahedron*(GLdouble *radius*);

void *auxWireOctahedron*(GLdouble *radius*);
void *auxSolidOctahedron*(GLdouble *radius*);

void *auxWireTetrahedron*(GLdouble *radius*);
void *auxSolidTetrahedron*(GLdouble *radius*);

void *auxWireDodecahedron*(GLdouble *radius*);
void *auxSolidDodecahedron*(GLdouble *radius*);

void *auxWireCone*(GLdouble *radius*, GLdouble *height*);
void *auxSolidCone*(GLdouble *radius*, GLdouble *height*);

void *auxWireTeapot*(GLdouble *size*);
void *auxSolidTeapot*(GLdouble *size*);

Draws the specified wireframe or solid object. These routines are self−initializing; that is, the first time a rendering request is made, a display list is created for the object. Every subsequent time the routine is called, the same display list is executed. All these models are drawn centered at the origin. When drawn with unit scale factors, these models fit into a box with all coordinates from −1 to 1. Use the arguments for these routines to scale the objects.

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 *auxIdleFunc()*. This routine takes a pointer to the function as its only argument. Pass in zero to disable the execution of the function.

void *auxIdleFunc*(void **func*);

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

Running the Program

The examples in the book typically draw the scene each time the window is created, moved, or reshaped, or if some input event occurs. Use *auxMainLoop()* to specify the routine that draws the scene.

void *auxMainLoop*(void(**displayFunc*)(void));

Specifies the function, *displayFunc*, that's called when the window needs to be updated. *displayFunc* should redraw the objects in your scene.