Event Driven Programming

After you create your window and initialize any variables, you need to **register callback functions** that handle any possible events that may occur during the program execution. Once you have registered these callback functions, your program should go into an infinite loop that polls for possible events and dispatches them to the corresponding function that was registered earlier.

Types of Events

- Window Events
- Keyboard Events
- Mouse Events
- Idle Function for when no events are in the event queue

Since OpenGL has no built-in event handling capabilities, we use the "glut.h" library which provides window management facilities and input handling routines.

The "Invisible" Event Loop

First, you register events you are interested in. This is usually done in your main() procedure. Once events are registered and your program calls the **glutMainLoop()** function, your program goes into a state where it monitors system activity for events you registered earlier.

In effect, **glutMainLoop** calls an "invisible" event loop which might look something like the following. Note that this is simply for illustrative purposes. This loop may be implemented completely differently in the code. Plus, you never actually see this event loop.

```
for (;;)
  {
    switch (EventType)
      {
        case KeyboardPressEvent:
          CallMyKeyPressHandler();
        break;

        case MouseButtonEvent:
          CallMyMouseButtonHandler();
        break;

        case MouseCursorMovementEvent:
          CallMyMouseMovementHandler();
        break;

        case WindowExposeEvent:
          CallMyUpdateDisplayHandler();
        break;

        case WindowResizeEvent:
          CallMyResizeWindowHandler();
          CallMyUpdateDisplayHandler();
        break;

        default:
          CallMyIdleTimeHandler();
      } /* End switch */
  }
```

Window Events

There are two main window events you will be concerned with for most of your programs.

- Expose Events
- Reshape / Resize Events

1 Expose Events

When your program first runs and your main window is displayed on the screen, an **Expose Event** is generated. Also, when you obscure the window with another window or move your window, an **Expose Event** is generated. To handle this event, you need a procedure that will completely redraw the contents of your window.

void **glutDisplayFunc**(void (* *displayFunction*)(void));

2 Reshape / Resize Events

If you resize your window, your program needs to redraw the scene. You must decide if you want to keep your aspect ratio the same, or let squares become rectangles, for example. If you omit a **Reshape Event** handler, "glut.h" will use a default **Reshape Event** function.

void **glutReshapeFunc**(void (* *reshapeFunction*)(int Width, int Height));

You may want to have a Reshape Event handler to keep a square a square in your scene. For this you would need to get the Width and Height of the newly resized window, calculate the viewing box for this new window, and pass the results to your camera routine (eg. glOrtho).

Keyboard Events

When you press keys on the keyboard, you must have a **Keyboard Event** function registered to handle them. When using the "glut.h" library, you write one function that handles all keypresses.

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

For example, if you wanted the user to type "q" to quit the program, you might do this in your main() procedure:

```
glutKeyboardFunc(KeyPress);
```

Then, somewhere else in your program, your function might look like this:

```
void KeyPress(unsigned char Key, int X, int Y)
{
  if (toupper(Key) == 'Q')
    exit(0);
}
```

Mouse Events

When using the "glut.h" library, you can get **mouse button** and **mouse movement** events.

When you want to get **mouse button** events, such as mouse button presses and releases, you use the following function:

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

where button will come in as GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON. State will be either GLUT_UP or GLUT_DOWN. You would test for these values in your *mouseButtonFunction*.

With the "glut.h" library, you also have the capability to detect mouse movement with the following two callback functions.

void **glutMotionFunc**(void (* *mouseDownMove*)(int x, int y));
void **glutPassiveMotionFunc**(void (* *mouseUpMove*)(int x, int y));

The **glutMotionFunc** is called when the mouse moves in the window and any mouse button is down. The **glutPassiveMotionFunc** is called when the mouse moves in the window and *no* mouse buttons are down. You should use this one with care since you will be generating lots of mouse movement events when no mouse buttons are pressed.

Idle Function

When no other types of events are being generated, you may actually want to do something. For example, perhaps you have a cube in your scene that you want to rotate without input from the user. Since "glut.h" is an event-driven environment, you have to register a callback function when nothing is happening. This is the job of the **Idle Function**.

void **glutIdleFunc**(void (* *idleFunction*)(void));

In your *idleFunction*, you can do anything you want. For example, you may want to increment the degrees of rotation of the camera about an object or increment a timer.

If you have changed anything that would require the window to be redrawn, you must refresh your display. With the "glut.h" library, you issue a **glutPostRedisplay**(); call which will call your *myDisplay* function automatically.

Starting the Event Loop

After you have registered all of your callback functions, you want your program to poll for any events. You do this by sending your program into a **main loop**. You can think of this loop as a infinite `for(;;)` loop that has a *case* statement for each event that you have registered.

void **glutDisplayFunc**(void (*myDisplayFunction)(void));
void **glutMainLoop**(void);

When you use "glut.h", your *myDisplayFunction* needs to have been registered earlier with **glutDisplayFunc**.

Camera View vs. Model View

In OpenGL you have to be concerned not only about your objects, but also how you are looking at the objects. OpenGL provides two different modes for this.

1 Camera View

Usually, you first set where your camera is located, the direction in which it is pointing, and the viewing volume. You do all of this by first putting OpenGL into PROJECTION mode. Then, you translate and rotate the camera to the correct position. Last, you choose one of the many viewing commands to simulate a camera.

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glTranslatef(0.0,0.0,-10.0);
glFrustum(-5,5,-5,5,10,-1);
```

2 Model View

After your camera is in position you want to put OpenGL into MODELVIEW mode. Then, you set up any translations and rotations that you want your objects to undergo. Finally, you draw your objects.

```
glMatrixMode(GL_MODELVIEW);
glTranslatef(3.0,2.0,-5.0);
glRotatef(45.0,0.0,0.0,1.0);
glBegin(GL_POLYGON);
  glVertex3i(1,2,3);
  glVertex3i(2,3,1);
  glVertex3i(0,1,0);
glEnd();
```

Usually, you will position the camera and the objects any time you need to redraw your scene. This means that you usually put all of these commands in your myDisplay function. However, if you never move the camera, you can set the camera position up at the beginning, and then leave OpenGL in MODELVIEW mode. This will allow you to redraw your objects only, but may save a little time since you don't need to recalculate the camera position.