The GL Utility Toolkit (GLUT)

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

The GL Utility Toolkit (GLUT) serves two major purposes:
1. It interfaces with your operating system and window system
2. It provides various application utilities, such as drawing 3D shapes for you

You can find GLUT (actually freeGLUT) at:
http://freeglut.sourceforge.net/

You don’t actually have to go out here. We will give you some libraries that are ready-to-use.

Using GLUT to Setup the Window

All the GLUT_XXX constants are defined in glut.h

Constants not beginning with GL_ or GLUT_ are user-defined

glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
// set the initial window configuration:

// set the initial window configuration:

glutInitWindowPosition( 0, 0 );
glutInitWindowSize( INIT_WINDOW_SIZE, INIT_WINDOW_SIZE );
// open the window and set its title:

// open the window and set its title:

MainWindow = glutCreateWindow( WINDOWTITLE );
glutSetWindowTitle( WINDOWTITLE );

Using GLUT to Specify Event-driven Callback Functions

For example, the Keyboard( ) function gets called whenever a keyboard key is hit

A NULL callback function means that this event will be ignored
The **Keyboard Callback Function**

```c
void Keyboard( unsigned char c, int x, int y )
{
    if( DebugOn != 0 )
        fprintf( stderr, "Keyboard: '%c' (0x%0x)
    switch( c )
    { case 'o':  case 'O':
        WhichProjection = ORTHO; break;
    case 'p':  case 'P':
        WhichProjection = PERSP; break;
    case 'q':  case 'Q':
    case ESCAPE:
        DoMainMenu( QUIT );     // will not ever return
                                // keep the compiler happy
        break;
    default:
        fprintf( stderr, "Don't know what to do with keyboard hit: '%c' (0x%0x)\n    // force a call to Display( ):
    glutSetWindow( MainWindow );
    glutPostRedisplay( )
}
```

The **MouseButton Callback Function**

```c
void MouseButton( int button, int state, int x, int y )
{
    int b = 0;                      // LEFT, MIDDLE, or RIGHT
    if( DebugOn != 0 )
        fprintf( stderr, "MouseButton: %d, %d, %d, %d
    // get the proper button bit mask:
    switch( button )
    { case GLUT_LEFT_BUTTON:
        b = LEFT;               break;
    case GLUT_MIDDLE_BUTTON:
        b = MIDDLE;             break;
    case GLUT_RIGHT_BUTTON:
        b = RIGHT;              break;
    default:
        b = 0;fprintf( stderr, "Unknown mouse button: %d\n    // button down sets the bit, up clears the bit:
    if( state == GLUT_DOWN )
        { Xmouse = x; Ymouse = y;
          ActiveButton |= b;              // set the proper bit
        }
    else
        { ActiveButton &= ~b;             // clear the proper bit
        }
```

The **MouseMotion Callback Function**

```c
void MouseMotion( int x, int y )
{
    if( DebugOn != 0 )
        fprintf( stderr, "MouseMotion: %d, %d\n    int dx = x - Xmouse;            // change in mouse coords
    int dy = y - Ymouse;            // change in mouse coords
    if( ( ActiveButton & LEFT ) != 0 )
        { Xrot += ( ANGFACT*dy );
          Yrot += ( ANGFACT*dx );
    }
    if( ( ActiveButton & MIDDLE ) != 0 )
        { Scale += SCLFACT * (float) ( dx - dy );
          // keep object from turning inside-out or disappearing:
          if( Scale < MINSCALE )
              Scale = MINSCALE;
    Xmouse = x;                     // new current position
    Ymouse = y;
    glutSetWindow( MainWindow );
    glutPostRedisplay( )
```

The **Animate Idle Callback Function**

```c
void Animate( )
{
    // put animation stuff in here -- change some global variables
    // for Display( ) to find:
    int ms = glutGet( GLUT_ELAPSED_TIME ); // milliseconds
    ms %=  MS_IN_THE_ANIMATION_CYCLE;
    Time = (float)ms / (float)MS_IN_THE_ANIMATION_CYCLE;        // \[ 0., 1. )
    // force GLUT to do a call to Display( ) next time it is convenient:
    glutSetWindow( MainWindow );
    glutPostRedisplay( )
```
This is the color menu’s callback function. When the user selects from this pop-up menu, its callback function gets executed. Its argument is the integer ID of the menu item that was selected. You specify that integer ID in \texttt{glutAddMenuEntry()}. This is how you create hierarchical sub-menus. Finally, tell GLUT which mouse button activates the entire menu hierarchy.

The GLUT 3D Objects

<table>
<thead>
<tr>
<th>Solid Type</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>glutSolidSphere</code></td>
<td>radius, slices, stacks</td>
</tr>
<tr>
<td><code>glutWireSphere</code></td>
<td></td>
</tr>
<tr>
<td><code>glutSolidCube</code></td>
<td>size</td>
</tr>
<tr>
<td><code>glutWireCube</code></td>
<td></td>
</tr>
<tr>
<td><code>glutSolidCone</code></td>
<td>base, height, slices, stacks</td>
</tr>
<tr>
<td><code>glutWireCone</code></td>
<td>base, height, slices, stacks</td>
</tr>
<tr>
<td><code>glutSolidTorus</code></td>
<td>innerRadius, outerRadius, nsides, nrings</td>
</tr>
<tr>
<td><code>glutWireTorus</code></td>
<td>innerRadius, outerRadius, nsides, nrings</td>
</tr>
<tr>
<td><code>glutSolidDodecahedron</code></td>
<td></td>
</tr>
<tr>
<td><code>glutWireDodecahedron</code></td>
<td></td>
</tr>
<tr>
<td><code>glutSolidOctahedron</code></td>
<td></td>
</tr>
<tr>
<td><code>glutWireOctahedron</code></td>
<td></td>
</tr>
<tr>
<td><code>glutSolidTetrahedron</code></td>
<td></td>
</tr>
<tr>
<td><code>glutWireTetrahedron</code></td>
<td></td>
</tr>
<tr>
<td><code>glutSolidIcosahedron</code></td>
<td></td>
</tr>
<tr>
<td><code>glutWireIcosahedron</code></td>
<td></td>
</tr>
<tr>
<td><code>glutSolidTeapot</code></td>
<td>size</td>
</tr>
<tr>
<td><code>glutWireTeapot</code></td>
<td>size</td>
</tr>
</tbody>
</table>

Without lighting, the GLUT solids don’t look very cool. I’d recommend you stick with the wireframe versions of the GLUT 3D objects for now. We will get to lighting soon.

In case you have a hard time remembering which direction "slices" are, think of this:

slicesstacks

Without lighting

With lighting