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.
What is GLUT?

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

GLUT_RGBA I want to display colors
GLUT_DOUBLE I want to do double-buffering
GLUT_DEPTH I want to use a depth-buffer while rendering

```c
glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
```

// set the initial window configuration:

```c
glutInitWindowPosition( 0, 0 );
glutInitWindowSize( INIT_WINDOW_SIZE, INIT_WINDOW_SIZE );
```

// open the window and set its title:

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

Constants not beginning with GL_ or GLUT_ are user-defined
Using GLUT to Specify Event-driven Callback Functions

```c
glutSetWindow( MainWindow );
glutDisplayFunc( Display );
glutReshapeFunc( Resize );
glutKeyboardFunc( Keyboard );
glutMouseFunc( MouseButton );
glutMotionFunc( MouseMotion );

glutPassiveMotionFunc( NULL );
glutVisibilityFunc( Visibility );
glutEntryFunc( NULL );
glutSpecialFunc( NULL );
glutSpaceballMotionFunc( NULL );
glutSpaceballRotateFunc( NULL );
glutSpaceballButtonFunc( NULL );
glutButtonBoxFunc( NULL );
glutDialsFunc( NULL );
glutTabletMotionFunc( NULL );
glutTabletButtonFunc( NULL );
glutMenuStateFunc( NULL );
glutTimerFunc( -1, NULL, 0 );
glutIdleFunc( NULL );
```

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)n", c, c);

    switch(c)
    {
    case 'o': case 'O':
        WhichProjection = ORTHO;
        break;
    case 'p': case 'P':
        WhichProjection = PERSP;
        break;
    case 'q': case 'Q':
    case ESCAPE:
        DoMainMenu(QUIT);
        break;
    default:
        fprintf(stderr, "Don't know what to do with keyboard hit: '%c' (0x%0x)n", c, c);
    }

    // force a call to Display( ):
    glutSetWindow(MainWindow);
    glutPostRedisplay();
    glutPostRedisplay();
}
```

Where the mouse was when the key was hit

The key that was hit

Assign new display parameter values depending on what key was hit

Good programming practice

`glutPostRedisplay()` forces your `Display()` function to be called to redraw the scene with the new display parameter values.
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\n", button, state, x, y );

    // 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 );
    }

    // 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
    }
}
```

- Where the mouse was when the button was hit
- GLUT_DOWN or GLUT_UP
- Which button was hit
The **MouseMotion** Callback Function

```c
void MouseMotion(int x, int y)
{
    if (DebugOn != 0)
        fprintf(stderr, "MouseMotion: %d, %d\n", x, y);

    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();
}
```

- If the mouse moved with the left button down, do a rotate
- If the mouse moved with the middle button down, do a scale
- **glutPostRedisplay()** forces your Display() function to be called to redraw the scene with the new display parameter values
The **Animate** Idle Callback Function

The Idle Function gets called when the GLUT event handler has nothing else to do

```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( );
}
```

Setting it up in InitGraphics( )

We'll talk about this later. This is a good way to control your animations!
Pop-up Menus are easy to Create with GLUT

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 `glutAddMenuEntry( )`.

This is how you create hierarchical sub-menus

Finally, tell GLUT which mouse button activates the entire menu hierarchy
The GLUT 3D Objects

glutSolidSphere( radius, slices, stacks );
glutWireSphere( radius, slices, stacks );
glutSolidCube( size );
glutWireCube( size );
glutSolidCone( base, height, slices, stacks );
glutWireCone( base, height, slices, stacks );
glutSolidTorus( innerRadius, outerRadius, nsides, nrings );
glutWireTorus( innerRadius, outerRadius, nsides, nrings );
glutSolidDodecahedron();
glutWireDodecahedron();
glutSolidOctahedron();
glutWireOctahedron();
glutSolidTetrahedron();
glutWireTetrahedron();
glutSolidIcosahedron();
glutWireIcosahedron();
glutSolidTeapot( size );
glutWireTeapot( size );

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