Tutorial 3

Ok, so far we have learned how to draw coloured 3D objects. In this tutorial we will learn how to rotate objects and use keyboard controls.

Keyboard Control

To get key presses in our application GLUT provides us with two function, one for normal keys i.e. ASCII keys and one for special keys i.e. non-ASCII based keys for example function keys, arrow keys, etc. These functions are described below:

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

- func - The pointer to function that will process the normal keyboard events. Passing NULL will make GLUT to ignore all normal key events.

The argument to glutKeyboardFunc must have three arguments and of same type, otherwise compiler will throw error. The first argument 'key' is the ASCII value of key pressed and other two argument viz, 'x' and 'y' provides the location of mouse pointer. We have used this function for to capture Escape key to close our application. Example function code:

```
void processNormalKeys(unsigned char key, int x, int y) {
        if (key == 27)
                exit(0);
}
```

For capturing special key events GLUT provides us with function glutSpecialFunc. The signature of this function is as follows:

void glutSpecialFunc(void (*func) (int key, int x, int y));
Parameters:

- func - The pointer to function that will process special keyboard events.

We are going to write a function that rotates the cube using arrow keys and page up and page down keys. Left and right arrow keys rotates the cube with respect to y-axis, Up and Down arrow keys rotates the cube with respect to x-axis and Page Up and Page Down keys rotates the cube with respect to z-axis. Offset is the variable which controls the increment/decrement value for x, y and z angles.

```
void processSpecialKeys(int key, int x, int y) {
  switch(key) {
    case GLUT_KEY_LEFT :
    y_angle-=offset;
    break;
    case GLUT_KEY_RIGHT :
    y_angle+=offset;
    break;
    case GLUT_KEY_UP :
    x_angle-=offset;
    break;
    case GLUT_KEY_DOWN :
    x_angle+=offset;
    break;
    case GLUT_KEY_PAGE_UP :
    z_angle-=offset;
    break;
    case GLUT_KEY_PAGE_DOWN :
    z_angle+=offset;
    break;
  }

  //Redraw
  glutPostRedisplay();
}
```

To make the effects visible we need to redraw the scene on screen. GLUT provides a function glutPostRedisplay() to force redraw the screen. These functions are then registered with GLUT using

```
  glutKeyboardFunc(processNormalKeys);
  glutSpecialFunc(processSpecialKeys);
```

Rotation

Now that we have acquired all the angles for rotating the cube, lets start with rotation. GLUT provides the following function for rotation

void glRotatef(float angle, float x, float y, float z);
Parameters :

- angle - angle of rotation in degrees
- x - x co-ordinate of the axis about which the rotation occurs.
- y - y co-ordinate of the axis about which the rotation occurs.
- z - z co-ordinate of the axis about which the rotation occurs.

Note that only the direction of axis matters, magnitude dosent matters. So for rotation about x, y or z axis we use the function as follows

```
//rotation about x-axis by x_angle
glRotatef(x_angle, 1.0, 0.0, 0.0);

//rotation about y-axis by y_angle
glRotatef(y_angle, 0.0, 1.0, 0.0);

//rotation about z-axis by z_angle
glRotatef(z_angle, 0.0, 0.0, 1.0);
```

Also note that the sequence on these rotation matters, first the object is rotated on x-axis then on y-axis and then on z-axis. Also look at the problem of *gimbal lock* with such sequence of rotations.

Now that we have learned to how to rotate things, lets look take an insight the working of rotate function.

$$\begin{pmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The above matrix represents the rotation matrix formed by glRotatef() function, where c = cos(angle), s = sin(angle) and ||(x,y,z)||= 1 (if not, the GL will normalize this vector). This matrix is then multiplied with the currently loaded GL matrix. Thus anything rendered till now will get rotated. If we want to rotate the specific object only, then we need something more. Thus GLUT provides us with two very useful functions, viz. glPushMatrix() and glPopMatrix().

void glPushMatrix(void)

This function pushes the current GL matrix on stack and loads an new identity matrix. Thus any operations performed after calling this function will not affect the previous matrix and thus does not affects anything other than objects rendered after calling glPushMatrix().

void glPopMatrix(void)

This function pops the matrix last loaded on stack and multiplies it with the current matrix. Thus any transformations performed on top matrix will also affect the current matrix.

```
glPushMatrix();
glTranslate(-originalX, -originalY, -originalZ);//translate to origin
glRotate(angle, 0, 1, 0);
glTranslate(originalX, originalY, originalZ);   //translate back
...
glPopMatrix();
```

A general approach to do transformations on individual object is to push the current matrix, do transformations and pop the matrix again.

References

1. http://www.lighthouse3d.com/tutorials/glut-tutorial/keyboard/
2. http://www.lighthouse3d.com/tutorials/glut-tutorial/animation/
3. https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html