GLUT Tutorial

Mouse

In the previous section we saw how to add interactivity to an OpenGL application using GLUT's Keyboard functionality. Now its time to explore the mouse. GLUTs mouse interface provides a lot of options for adding mouse interactivy, namely detecting clicks and mouse motion.

Detecting Mouse Clicks
As in the keyboard version, GLUT provides a way for you to register the function that will be responsable for processing events generated by mouse clicks. The name of this function is *glutMouseFunc*,and it is commonly called in the initialization phase of the application. The syntax is as follows:

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

Parameters:
        func - The name of the function that will handle mouse click events
```

As we can see from the signature of *glutMouseFunc*, the function that will handle the mouse click events must have four parameters. The first relates to which button was pressed, or released. This argument can have one of three values:

- GLUT_LEFT_BUTTON
- GLUT_MIDDLE_BUTTON
- GLUT_RIGHT_BUTTON The second argument relates to the state of the button when the callback was generated, i.e. pressed or released. The possible values are:
- GLUT_DOWN
- GLUT_UP When a callback is generated with the state GLUT_DOWN, the application can assume that a GLUT_UP will come afterwards even if the mouse moves outside the window. However if the application calls *glutMouseFunc* again with NULL as argument then GLUT will stop sending mouse state changes.

The remaining two parameters provide the (*x*,*y*) coordinates of the mouse relatively to the upper left corner of the client area of the window.

Detecting Motion
GLUT provides mouse motion detection capabilities to an application. There are two types of motion that GLUT handles: active and passive motion. Active motion occurs when the mouse is moved and a button is pressed. Passive motion is when the mouse is moving but no buttons are pressed. If an application is tracking motion, an event will be generated per frame during the period that the mouse is moving.
As usual you must register with GLUT the function that will be responsable for handling the motion events. GLUT allows us to specify two different functions: one for tracking passive motion, and another to track active motion.
The signatures for the GLUT functions are as follows:

```
void glutMotionFunc(void (*func) (int x,int y));
void glutPassiveMotionFunc(void (*func) (int x, int y));

Parameters:
        func - the function that will be responsible for the respective type of motion.
```

The parameters for the motion processing function are the (*x*,*y*) coordinates of the mouse relatively to the upper left corner of the window's client area.

Detecting when the mouse enters or leaves the window
GLUT is also able to detect when the mouse leaves or enters the window region. A callback function can be registered to handle these two events. The GLUT function to register this callback is *glutEntryFunc* and the syntax is as follows:

void glutEntryFunc(void (*func)(int state));

Parameters:
func - the function that will handle these events.

The parameter of the function that will handle these events tells us if the mouse has entered of left the window region. GLUT defines two constants that can be used in the application:

- GLUT_LEFT
- GLUT_ENTERED

Note: This doesn't work exactly as it says in Microsoft Windows, this is because in Microsoft's OS the focus is changed with a mouse click. Although you can change this is your own system using some tools from Microsoft, others are likely to have the standard setting so its probably better if you don't use this feature in Microsoft Windows to detect when the mouse enters/leaves the window.
Putting it all together
The first thing we should do is to register with GLUT which function will be responsible for handling mouse events. Therefore we are going to rewrite our main function to include all the necessary callback's registrations. We are going to add all the functionality described above to our application to present a simple example from which you can learn the missing pieces, or just pick up the things that where not that clear in this tutorial.

```
void main(int argc, char **argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowPosition(100,100);
	glutInitWindowSize(320,320);
	glutCreateWindow("SnowMen");
	glutDisplayFunc(renderScene);
	glutIdleFunc(renderScene);
	glutReshapeFunc(changeSize);


	//adding here the mouse processing callbacks
	glutMouseFunc(processMouse);
	glutMotionFunc(processMouseActiveMotion);
	glutPassiveMotionFunc(processMousePassiveMotion);
	glutEntryFunc(processMouseEntry);

	glutMainLoop();
}
```

OK, now lets have some fun! We're going to define the callback functions we registered to do some weird stuff. When a mouse button is pressed, and the ALT key is also pressed, we're going to change the triangle's color. So the left button makes the triangle red, the middle turns the triangle's color to green, whereas the right button makes a blue triangle. Next we present a function to do just this (note that the name of the function must be *processMouse* as this is the name registered in the main function:

```
void processMouse(int button, int state, int x, int y) {


        specialKey = glutGetModifiers();
        // if both a mouse button, and the ALT key, are pressed  then
        if ((state == GLUT_DOWN) &&
                        (specialKey == GLUT_ACTIVE_ALT)) {

                // set the color to pure red for the left button
                if (button == GLUT_LEFT_BUTTON) {
                        red = 1.0; green = 0.0; blue = 0.0;
                }
                // set the color to pure green for the middle button
                else if (button == GLUT_MIDDLE_BUTTON) {
                        red = 0.0; green = 1.0; blue = 0.0;
                }
                // set the color to pure blue for the right button
                else {
                        red = 0.0; green = 0.0; blue = 1.0;
                }
        }
}
```

Now lets have a more subtle color picking method. When a button is pressed, but no ALT key, we're going to set blue to 0.0, and let the red and green components be dependent upon the mouse position on the window's client area. The function bellow does just this:

```
void processMouseActiveMotion(int x, int y) {

        // the ALT key was used in the previous function
        if (specialKey != GLUT_ACTIVE_ALT) {
                // setting red to be relative to the mouse
                // position inside the window
                if (x < 0)
                        red = 0.0;
                else if (x > width)
                        red = 1.0;
                else
                        red = ((float) x)/height;
                // setting green to be relative to the mouse
                // position inside the window
                if (y < 0)
                        green = 0.0;
```

```
		else if (y > width)
			green = 1.0;
		else
			green = ((float) y)/height;
		// removing the blue component.
		blue = 0.0;
	}
}
```

It's time to add some action to passive motion. When the SHIFT key is pressed, the mouse will have a rotation in the X axis relative to the mouse position in the x window coordinate. We had to change slightly the *renderScene* function to do this, so here goes the new *renderScene*:

```
float angleX = 0.0;
...
void renderScene(void) {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glPushMatrix();
	glRotatef(angle,0.0,1.0,0.0);

	// This is the line we added for the
	// rotation on the X axis;
	glRotatef(angleX,1.0,0.0,0.0);

	glColor3f(red,green,blue);

	glBegin(GL_TRIANGLES);
		glVertex3f(-0.5,-0.5,0.0);
		glVertex3f(0.5,0.0,0.0);
		glVertex3f(0.0,0.5,0.0);
	glEnd();
	glPopMatrix();
	angle++;
	glutSwapBuffers();
}
```

Now for the function that will process passive motion events. This function will change the value of *angleX* relatively to the mouse x coordinate value.

```
void processMousePassiveMotion(int x, int y) {

	// User must press the SHIFT key to change the
	// rotation in the X axis
	if (specialKey != GLUT_ACTIVE_SHIFT) {

		// setting the angle to be relative to the mouse
		// position inside the window
```

```
                if (x < 0)
                        angleX = 0.0;
                else if (x > width)
                        angleX = 180.0;
                else
                        angleX = 180.0 * ((float) x)/height;
        }
}
```

Finally, when the mouse leaves the window we will stop the animation. In order to do this we have to change the *renderScene* function again. It is a small change but here goes the renderScene function again.

```
// initially define the increase of the angle by 1.0;
float deltaAngle = 1.0;
...
void renderScene(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotatef(angle,0.0,1.0,0.0);
        glRotatef(angleX,1.0,0.0,0.0);
        glColor3f(red,green,blue);

        glBegin(GL_TRIANGLES);
                glVertex3f(-0.5,-0.5,0.0);
                glVertex3f(0.5,0.0,0.0);
                glVertex3f(0.0,0.5,0.0);
        glEnd();
        glPopMatrix();
        // this is the new line
        // previously it was: angle++;
        angle+=deltaAngle;
        glutSwapBuffers();
}
```

The *processMouseEntry* is the last function. As mentioned before, this doesn't work well in Microsoft Windows, compile this in Linux and it should work.

```
void processMouseEntry(int state) {
        if (state == GLUT_LEFT)
                deltaAngle = 0.0;
        else
                deltaAngle = 1.0;
}
```