GLUT Tutorial

Keyboard

GLUT allows us to build applications that detect keyboard input using either the "normal" keys, or the special keys like F1 and Up. In this section we'll see how to detect which key was pressed, what further information we get from GLUT, and how to deal with that.

As you have probably noticed by now, whenever you want to take control of the processing of an event you have to tell GLUT in advance which function is going to perform such task. Up until now we used GLUT to tell the windows system which functions we wanted to do the rendering when the window needed to be repainted, which function to call when the system was idle, and which function to call when the window was resized.

Similarly we must do the same thing for keyboard events. We must notify the windows system, using GLUT, which function(s) will perform the required processing when a key is pressed. This procedure of notifying that when an event occurs we want to execute a particular function is also called "register a callback function".

GLUT provides two functions to register callbacks for keyboard events that occur when you press a key. The first one, *glutKeyboardFunc*, is used to tell the windows system which function we want to process the "normal" key events. By "normal" keys, we mean letters, numbers, anything that has an ASCII code. The syntax for this function is as follows:

---

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

Parameters:

func - The name of the function that will process the "normal" keyboard events. Passing NULL as an argument causes GLUT to ignore "normal" keys.

The function used as an argument to *glutKeyboardFunc* needs to have three arguments. The first indicates the ASCII code of the key pressed, the remaining two arguments provide the mouse position when the key is pressed. The mouse position is relative to the top left corner of the client area of the window.

A possible implementation for this function is to provide a way out of the application when the user presses the ESCAPE key. Note that when the *glutMainLoop* function was presented we mentioned that it was an infinite loop, i.e. it never returns. The only way out of this loop is to call the system *exit* function. So that's exactly what our function will do, when the user presses escape it calls the system *exit* function causing the application to terminate (remember to include stdlib.h in the source code). Next we present the function code:

```
void processNormalKeys(unsigned char key, int x, int y) {

        if (key == 27)
                exit(0);
}
```

Note that we are using exactly the same signature as the one specified in the syntax of *glutKeyboardFunc*. If you don't do this you'll get an error when compiling this in VC, and we don't want that, do we?

OK, ready to move on? Lets tackle the special keys now. GLUT provides the function *glutSpecialFunc* so that you can register your function for special key events processing. The syntax for this function is as follows:

---

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

Parameters:

func - The name of the function that will process the special keyboard events. Passing NULL as an argument causes GLUT to ignore the special keys.

We're going to write a function that changes the color of our triangle when some of the special keys are pressed. This function will paint the triangle using red if F1 is pressed, green if F2 is pressed, and blue if F3 is pressed.

```
void processSpecialKeys(int key, int x, int y) {

        switch(key) {
                case GLUT_KEY_F1 :
                                red = 1.0;
                                green = 0.0;
                                blue = 0.0; break;
                case GLUT_KEY_F2 :
                                red = 0.0;
                                green = 1.0;
                                blue = 0.0; break;
                case GLUT_KEY_F3 :
                                red = 0.0;
                                green = 0.0;
                                blue = 1.0; break;
        }
}
```

The GLUT_KEY_* are predefined constants in *glut.h*. The full set of constants is presented next:

```
GLUT_KEY_F1             F1 function key
GLUT_KEY_F2             F2 function key
GLUT_KEY_F3             F3 function key
GLUT_KEY_F4             F4 function key
GLUT_KEY_F5             F5 function key
GLUT_KEY_F6             F6 function key
GLUT_KEY_F7             F7 function key
GLUT_KEY_F8             F8 function key
GLUT_KEY_F9             F9 function key
GLUT_KEY_F10            F10 function key
GLUT_KEY_F11            F11 function key
GLUT_KEY_F12            F12 function key
GLUT_KEY_LEFT           Left function key
GLUT_KEY_RIGHT          Up function key
GLUT_KEY_UP             Right function key
GLUT_KEY_DOWN           Down function key
GLUT_KEY_PAGE_UP        Page Up function key
GLUT_KEY_PAGE_DOWN      Page Down function key
GLUT_KEY_HOME           Home function key
```

```
GLUT_KEY_END           End function key
GLUT_KEY_INSERT        Insert function key
```

In order for the code defined above on *processSpecialKeys* to compile we must add the declaration of the *red*, *green*, and *blue* variables to the beginning of our code. Furthermore, for the code to have the desired effect we must change the function responsible for the rendering, *renderScene*.

```
...
// all variables initialized to 1.0, meaning
// the triangle will initially be white
float red=1.0, blue=1.0, green=1.0;



void renderScene(void) {
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotatef(angle,0.0,1.0,0.0);

        // this is where we set the actual color
        // glColor specifies the color of all further drawings
        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();
}
```

OK, now we're ready to tell GLUT that the functions we just defined are the ones that will process keyboard events. In other words it is time to call GLUT's *glutKeyboardFunc* and *glutSpecialFunc*. The call to these functions can be made anywhere, meaning that we may change the processing function for keyboard event processing at any time. However this is not an usual feature, so we'll place it on the main function. Next we present the new main function, with keyboard processing is presented (note that this function is based on the previous sections of this tutorial):

```
void main(int argc, char **argv) {
        glutInit(&argc, argv);

        // This is where we say that we want a double buffer
        glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);

        glutInitWindowPosition(100,100);
        glutInitWindowSize(320,320);
        glutCreateWindow("3D Tech - GLUT Tutorial");
        glutDisplayFunc(renderScene);
        glutIdleFunc(renderScene);
        glutReshapeFunc(changeSize);
```

```
        // here are the new entries
        glutKeyboardFunc(processNormalKeys);
        glutSpecialFunc(processSpecialKeys);

        // enable depth testing
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

The Visual Basic project can be downloaded here (glut3.zip).

CTRL, ALT and SHIFT

Sometimes we may want to know if one modifier key, i.e. CTRL, ALT or SHIFT is being pressed. GLUT provides a function that detects if any modifier is being pressed. This function should only be called inside the functions that process keyboard or mouse input events. The syntax for this function is:

```
int glutGetModifiers(void);
```

The return value for this function is either one of three predefined constants (in *glut.h*), or any bitwise OR combination of them. The constants are:

- GLUT_ACTIVE_SHIFT - Set if either you press the SHIFT key, or Caps Lock is on. Note that if they are both on then the constant is not set.
- GLUT_ACTIVE_CTRL - Set if you press the CTRL key.
- GLUT_ACTIVE_ALT - Set if you press the ALT key. Beware that the windows system may intercept some modifiers, no callback is generated in these cases. So lets extent our *processNormalKeys* a little bit to see how to handle these modifier keys. Suppose that you want the variable *red* to be 0.0 when the user presses r, and 1.0 when the user presses ALT + r. The following piece of code will do the trick:

```
void processNormalKeys(unsigned char key, int x, int y) {

        if (key == 27)
                exit(0);
        else if (key=='r') {
                int mod = glutGetModifiers();
                if (mod == GLUT_ACTIVE_ALT)
                        red = 0.0;
                else
                        red = 1.0;
        }
}
```

Notice that if we wanted to do something if 'R' is pressed, then the following code will not work. This is because the key is 'R' and not 'r', therefore you should not use the SHIFT modifier to turn lowercase into uppercase or to achieve any symbol that has a defined ASCII code. You can use SHIFT with F1, though.

```
void processNormalKeys(unsigned char key, int x, int y) {

        if (key == 27)
                exit(0);
```

```
        // this is incorrect if we're looking for an 'R'
        else if (key=='r') {
                int mod = glutGetModifiers();
                if (mod == GLUT_ACTIVE_SHIFT)
                        ...
        }
}
```

One last thing, how do you detect CTRL+ALT+F1? In this case we must detect two modifiers at the same time. In order to achieve this we do a bitwise OR with the desired constants. The following piece of code only changes the color to red if combination CTRL+ALT+F1 is pressed.

```
void processSpecialKeys(int key, int x, int y) {

        int mod;
        switch(key) {
                case GLUT_KEY_F1 :
                   mod = glutGetModifiers();
                   if (mod == (GLUT_ACTIVE_CTRL|GLUT_ACTIVE_ALT)) {
                        red = 1.0; green = 0.0; blue = 0.0;
                   }
                   break;
                case GLUT_KEY_F2 :
                   red = 0.0;
                   green = 1.0;
                   blue = 0.0; break;
                case GLUT_KEY_F3 :
                   red = 0.0;
                   green = 0.0;
                   blue = 1.0; break;
        }
}
```

One final note about registering callbacks for keyboard and mouse events: when you register a callback for these events there maybe a performance penalty because the system is looking for these events. If you don't want to handle these events avoid registering callback functions to handle them.