

# COMP3201 – Computer Graphics

## Module 1: Introduction and Graphics Primitives

### 1.2 Basic OpenGL/GLUT Programming

#### 1.2.1 A Basic Program

```
/* helloworld.c          */

#include <GL/glut.h>
#include <stdlib.h>

GLvoid display(GLvoid)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

void init(void)
{
    /* set background color */
    glClearColor(1.0,1.0,0.0,1.0);
}

int main(int argc, char* argv[])
{
    GLint width;
    GLint height;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );

    width = glutGet(GLUT_SCREEN_WIDTH);
    height = glutGet(GLUT_SCREEN_HEIGHT);

    glutInitWindowPosition(width/6, height/6);
    glutInitWindowSize(width/4, height/4);
    glutCreateWindow(argv[0]);

    init();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```

#### 1.2.2 Notes on a Basic Program

- Functions **init** and **display** are programmed by the user to handle any initialisation requirements and display any objects, respectively.
- The function **main** calls some **GLUT** functions to initialise the **glut** library, initialise the requested buffers, open a window, and register appropriate callback functions before entering the event-handling loop.

- **glutInit** initialises the **GLUT** library.
- **glutInitDisplayMode** sets the initial display mode to the bitwise OR of the bit masks of its arguments (**GLUT\_SINGLE**, **GLUT\_RGB** and **GLUT\_DEPTH** in this example). That is, it initialises the requested buffers.
- **glutDisplayFunc(display)** registers a callback function for updating the current window; **display** is a user-defined function.
- **glutMainLoop()** is the **GLUT** event processing loop; it invokes callbacks, and is an infinite loop. It should be the last function in **main()**. Until a special key has been programmed to exit the program, use Ctrl-C to kill the process.

### 1.2.3 Screen Size

To determine the size of the screen, use **glutGet**:

```
screenwidth = glutGet(GLUT_SCREEN_WIDTH);
screenheight = glutGet(GLUT_SCREEN_HEIGHT);
```

### 1.2.4 Window Size and Position

For a window to be half the screen width and height, and for this window to be positioned towards the upper left-hand corner of the screen, use the following **glut** functions:

```
glutInitWindowPosition(screenwidth/4, screenheight/4);
glutInitWindowSize(screenwidth/2, screenheight/2);
glutCreateWindow(argv[0]);
```

The last statement creates the window and displays the program name (as given by **argv[0]**) at the top of the window.

Note that screen width and height are given in pixels, and that the window position coordinates are relative to the upper-left corner of the screen.

### 1.2.5 Background Colour

Having created a window, initialise the background colour of the window by placing the following **gl** command in the **init** routine:

```
glClearColor(1.0, 1.0, 0.0, 1.0);
```

the first three arguments correspond to the Red/Green/Blue components (the values range from 0 to 1 and represent what proportion of the colour component is to be used), while the fourth argument is the alpha-blending/transparency value; alpha-blending also ranges from 0 to 1. The colour used here is yellow. Remember that, because OpenGL is a state machine, this colour is used until reset by another call to **glClearColor**. The default colour is black, represented by (0.0, 0.0, 0.0, 0.0).

Having specified the background colour to be used, the colour is then applied to the window by the following **gl** function calls which should be put in the **display** routine:

```
glClear(GL_COLOR_BUFFER_BIT);
```

**glFlush();**

The **glFlush** function forces all pending operations to begin, and it sends all data to the server. It is a good idea to call **glFlush()** at the end of every frame (**glFlush** is an asynchronous function call).

Another **gl** function is **glFinish()**, which forces all pending operations to complete (**glFinish** is a synchronous function call). This could be used, for example, for synchronising tasks, but it impacts on performance and so should only be used when necessary.

### **1.2.6 Processing Keyboard Input**

The keyboard and mouse are two of the physical input devices that will be used during this course, and both are handled by the **GLUT** framework.

Keyboard input is handled by two separate callbacks. The callback for the ASCII generating keys (e.g. alphanumeric keys) is set by **glutKeyboardFunc()** and the callback for non-ASCII generating keys (e.g. function keys, cursor keys) is set with **glutSpecialFunc()**.

The function prototypes for these callbacks are:

```
GLvoid keyboard(GLubyte, GLint, GLint);  
GLvoid specialkeys(GLint, GLint, GLint);
```

The callbacks are registered with the **GLUT** framework by

```
glutKeyboardFunc(keyboard);  
glutSpecialFunc(specialkeys);
```

Finally, the actual processing to be carried out according to the key pressed is defined in the routines **keyboard** or **specialkeys**. For example:

```
GLvoid keyboard(GLubyte key, GLint x, GLint y)  
{  
    switch (key) {  
        case ' ':          /* SPACE key */  
            /* some processing here */  
            glutPostRedisplay();  
            break;  
        case KEY_ESC:   /* exit when escape key used */  
            exit(0);  
    }  
}
```

The **GLUT** function **glutPostRedisplay()** sends a request to the **GLUT** framework asking for the display callback to be called. Note that you may like to specify **#define KEY\_ESC 27** (27 is the ASCII value for the escape key), as **GLUT** hasn't defined a macro for this.

### **1.2.7 Processing Mouse Input**

The function **glutMouseFunc** is used to register a callback to handle mouse input. The callback is called on the **down** and **up** motion of the mouse button. The callback function is passed information about whether the button is **up** or **down** and also the position of the mouse when the button was pressed.

There is also **glutMotionFunc** to register a callback for the mouse being moved while one of its buttons is pressed. **glutPassiveMotionFunc** registers a callback for the mouse being moved while no button is pressed.

### **1.2.8 Checking for Errors**

OpenGL sets a flag when an error occurs, and the function **glGetError** can be used to check for errors. OpenGL only records the first error to occur. All subsequent errors are ignored until the error buffer is cleared by a call to **glGetError**.

For example, you can use the routine

```
GLvoid checkError( const char* const label )
{
    GLenum error;
    error = glGetError();
    while ( GL_NO_ERROR != error )
    {
        fprintf( stderr, "%s: %s\n", label,
gluErrorString(error) );
        error = glGetError();
    }
}
```

and then scan for errors within your **display** routine by placing the command **checkError("display");** directly above **glFlush();**.