OpenGL 4 Tutorial 1: Basic Rendering Framework

Anton Gerdelan

March 29, 2012

freeGLUT uses function pointers (call-backs) as kind of event-driven system. The original GLUT gave you no way to cleanly interrupt the programme, but freeGLUT adds a few well-needed support functions to make life a bit easier. If you are an object-oriented programmer you will find it rather difficult to turn this framework into a class, so I suggest not bothering with that!

This tutorial follows on from the previous one. We will add code to the end of that programme.

1 Window Set-Up

Okay, after initialising freeGLUT (see Hello World), and before glutCreateWindow you probably want to tell it how big to make the window, in pixels:

Look up the following function’s API documentation and give it some sensible parameters. This function needs to go before the glutCreateWindow function.

```c
glutInitWindowSize();
```

And also tell freeGLUT what kind of display to use. We want to use a depth-buffer for sorting in 3D. We also want to use double-buffering to clean up our display. I also explicitly specify a colour buffer here, although this should be the default:

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

2 Forcing OpenGL Version

Optionally, we can force openGL to use a specific version. In the Hello World article we captured integers for the latest version supported by the system. We could use that here if we wanted to:

```c
glutInitContextVersion(); // look up this function and give it sensible parameters
glutInitContextFlags(GLUT_FORWARD_COMPATIBLE); // do not allow deprecated
glutInitContextProfile(GLUT_CORE_PROFILE); // force current version of spec
```

The forward-compatible context means that deprecated functions (listed in blue on the OpenGL “quick reference card”) are not allowed. The core profile also disables all functions from older versions from being used. This is a good way to make sure that we are only using the latest version of OpenGL, and no out-moded (and potentially slower) functions. It also gives our programme longer shelf life. We might want to temporarily enable some older functions for debugging purposes (it can be easier to draw lines with the old fixed-function approach), in which case we can change these flags.
3 Call-backs and Interrupts

freeGLUT allows us to set an interrupt when the user clicks on the ‘x’ to close our window (otherwise we are in trouble):

```c
glutSetOption(GLUT_ACTION_ON_WINDOW_CLOSE, GLUT_ACTION_GLUTMAINLOOP_RETURNS);
```

Here it is telling glut to stop its callbacks from calling back.

At this point we can set up our framework using freeGLUT’s callbacks:

```c
glutDisplayFunc(renderCallback); // register rendering callback
glutReshapeFunc(resizeCallback); // register callback for reshape
glutKeyboardFunc(keyboardCallback);
```

These are the basics that we want. It is worth having a look at the freeGLUT/GLUT documentation to see the rest.

- The first is simple; when freeGLUT thinks that it’s time to render another frame it will call a function that we make, called `renderCallback()`.
- Once when the programme is first started, and any time the window is resized by the user `resizeCallback()` is called.
- Whenever the user presses a key on the keyboard, a function called `keyboardCallback()` will be called.

You can give these functions any name that you like, but the arguments must be as specified in documentation.

3.1 glutDisplayFunc

The display function needs no parameters, and has a void return type. Add a declaration of the function above `main()`. Then write the callback function:

```c
void renderCallback() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // clear buffer using colour
    /*
     * add everything that we want to draw here
     ...
    */
    glutSwapBuffers(); // swap drawing back-buffer to displayed front buffer
    glutPostRedisplay(); // flag for redraw
}
```

First we wipe the buffer clean for drawing. If we have the

```c
GL_COLOR_BUFFER_BIT
```

set then it will use a background colour that we will give inside our main function. We also tell it to clear out the depth buffer (which we need to do if drawing in 3D). Inside here we will loop through everything that we need to draw, then tell freeGLUT that the buffer is finished and it can flip it onto the view. We also tell freeGLUT that it should call this function again as soon as possible. We will come back to this function later.
3.2 glutReshapeFunc

The resize callback has two mandatory parameters. When freeGLUT calls this function once when the programme starts, and then any time the user resizes the window. Freeglut will supply us with the new size of the window. Add a declaration for this function. Then we can define it:

```c
void resizeCallback(int width, int height) {
    glViewport(0, 0, width, height); // change viewport size in gl
    /* if using perspective projection then we will want to
    change the projection matrix here as well */
}
```

We use the actually width and height values here to set the viewport size in OpenGL.

3.3 glutKeyboardFunc

The keyboard callback allows us to handle key presses. It is quite nice, compared to other input handlers, because it just gives us the character as a char. If the shift key is held it gives us an upper-case version. This limits us to standard keys though - there is another callback for handling function keys.

```c
void keyboardCallback(unsigned char key, int x, int y) {
    printf("the %c key was pressed\n", key);
}
```

4 OpenGL Programme Structure

You can see that using a call-back function like this to modify something is going to get messy. The easiest option is to use a few global variables. In my programmes I tend to make 2 or 3 global "manager" structs that can be accessed from any callback:

- a "ShaderProgrammeManager" holds all the shaders used
- a "RenderableManager" holds all the meshes/geometry/vertex buffers in the scene
- a "Camera" holds the main scene camera

This can tidy things up a bit. Turning this into a C++ OO class can get a bit ugly - you would need to make static methods (which are declared outside the class declaration) and static attributes.

5 Go!

We probably want to set a background colour, and enable some generally useful OpenGL behaviour. OpenGL colours are usually red, blue, green, alpha and the values of each component are between 0 and 1.

```c
glClearColor() // look up this function and give it some parameters
```

We want to enable depth-testing by default.

```c
glEnable(); // look up this function and find out how
```
Look up the `glEnable()` reference for a full list of options. Normally we would associate `glEnable` options with each different mesh or rendered geometry, so we would call them in the render callback, but the options above should apply to most things that we want to draw.

We also want to define a depth testing function to “less than”

```
glDepthFunc(); // look up this function and find out how
```

And enable culling of back-faces, so that we do not waste time drawing faces that are on the ”back” side with respect to the view direction. We have to tell OpenGL which convention we are using as the front and which side is the back. Triangles are built by joining three vertices together. If we give our vertices to OpenGL in counter-clockwise order (which we will) then we want that side to be the front:

```
glEnable(GL_CULL_FACE); // enable culling of faces
glCullFace(GL_BACK); // cull the back sides
glFrontFace(GL_CCW); // counter-clockwise is the front
```

We now have some functions to handle rendering and keyboard input, and have defined some default rendering options, but it needs to be kicked-off. The traditional GLUT-style main loop handles its own loop, which we already set to end when the window is closed:

```
 glutMainLoop();
```

You can also tell freeGLUT to stop from elsewhere e.g. is ‘q’ is pressed:

```
 glutLeaveMainLoop();
```

That’s it! We should have a basic framework that renders a window and view-port (the space GL uses for drawing inside the window) that continually renders.

6 Problem?

- compilation complains that “error: glutInitContextVersion was not declared in this scope” you are including the old GL/glut.h, not the new GL/freeglut.h. This may be a problem on Mac.
- window appears but contents are transparent, or a strange colour are you using glClear inside a working render callback?
- programme keeps running, even after closing window did you set the appropriate glut function to quit glutMain on window close? is your glutMain function inside an infinite while loop? If so, consider using a different type of interrupt instead of just finishing the glutMain function

7 Possible Extensions

7.1 Frame-Rate Ticker

If we wanted to we could also create a ”timer” callback that is called every 1/4 of a second or so (not too often) and use that to display the ”frames per second” (rendering frequency) in the title bar of the window. I have
found freeGLUT time functions to be too inaccurate for real-time calculations like movement, but it gives us a reasonable yard-stick for processing speed.

If we first add a global variable:

```c
int frameCount;
```

Then increment it inside the rendering callback:

```c
frameCount++;
```

We can now add a timer to our callbacks:

```c
glutTimerFunc(0, timerCallback, 0); // register a timer callback
```

And define it:

```c
void timerCallback(int value) {
    if (value != 0) {
        // update frame counter in window title
        char tmp[128];
        sprintf(tmp, "opengl @ %iFPS", frameCount * 4);
        glutSetWindowTitle(tmp);
    }
    frameCount = 0;
    glutTimerFunc(250, timerCallback, 1); // start timer again (every 1/4 second)
}
```

### 7.2 Settings File

As we get more and more settings and variables for initialisation it might be a nice idea to load these from a text file, so that we don’t have to recompile to change the window size, for example.