Tutorial : CS 475 Example 2b

Rohan Prinja

August 4, 2014

1 About

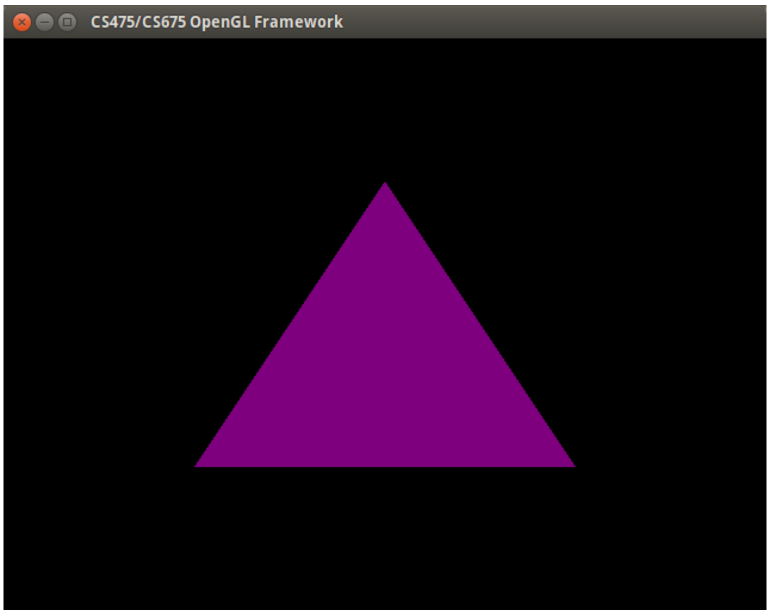
This is a tutorial for the deprecated OpenGL version of the second code example for CS 475. The code can be downloaded from here. When you untar the downloaded `tgz` file, you will find two cpp files `01_triangle.cpp` and `01_triangle_dep.cpp` and a file named `Makefile`. You will also find two shader files, `simple_fs.glsl` and `simple_vs.glsl`.

2 Running the code

Assuming you have everything set up (all relevant libraries etc.) you can compile the code by running `make`. This will produce two executable files `01_triangle` and `01_triangle_dep`. To run them, do `01_triangle` and `01_triangle_dep` respectively.

Both programs do the same thing. When run, they each open a window and display a purple triangle on a black background. The difference between the two is that they use different versions of OpenGL.

You've already read the tutorial for the modern OpenGL version. Good news: this tutorial is much simpler because the deprecated style of doing OpenGL was much simpler!



3 Understanding the code

Let's dive into the code. For this version, there is only one file we need to look at – `01_triangle_dep.cpp`. It uses deprecated OpenGL, namely, OpenGL 2 and below. It is very unlikely that you will be unable to run it.

First off, we include the utility files (although the shader utility file is not really needed for deprecated OpenGL).

```
#include "gl_framework.hpp"
#include "shader_util.hpp"
```

As before, we declare the coordinates of the vertices of the triangle with use an array of 9 `floats`.

```
float points[] = {
    0.0f,  0.5f,  0.0f,
    0.5f, -0.5f,  0.0f,
    -0.5f, -0.5f,  0.0f
  };
```

Good news, no shaders to worry about! Next, we have our rendering function, `renderGL()`.

```
void renderGL(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glColor4f(0.5, 0.0, 0.5, 1.0);
  glBegin(GL_TRIANGLES);
  glVertex3f(0.0f,  0.5f,  0.0f);
  glVertex3f(0.5f, -0.5f,  0.0f);
  glVertex3f(-0.5f, -0.5f,  0.0f);
  glEnd();
}
```

The first line we have already seen in the tutorial for the modern OpenGL version. The rest is unfamiliar to us. Let's try to figure it out.

First, we have a call to a function called `glColor4f()`. From the name `glColor` we can infer that this is a core OpenGL function. The `color` part indicates that we are doing something with color. In fact, we are setting a color. The `4f` indicates that we are passing in a color with 4 components, and each component is a `float` (hence, the `f`). From the previous tutorial, we know that OpenGL deals with colors in the form of floating point values in the range `[0, 1]`. Also, we know that OpenGL uses the RGBA format for specifying 4-element colors. So clearly this color `(0.5, 0.0, 0.5, 1.0)` must be equivalent to purple, since it has the RGB value (128, 0, 128) (since 128 = 0.5 * 255) with an alpha of 255 (no transparency).

It's important to remember that OpenGL is a state machine. The above command was a global command that set the color for **all** future OpenGL drawing commands! If we want a different color, we again have to call the `glColor4f()` function with arguments corresponding to the color we desire.

By the way, the fact that there exists a function called `glColor4f()` – as opposed to simply `glColor()` – might make you suspect that other, similar functions also exist, that take arguments of other types! Why not Google around and see if the OpenGL documentation has some help to offer in this regard?

Next, we have a function named `glBegin()` and, a few lines later, a function named `glEnd()`. The parameter to `glBegin()` is an enum named `GL_TRIANGLES`. We already know what this does from the previous tutorial - it tells OpenGL how to interpret a series of vertices. So clearly, `glBegin()` can be passed different enums that tell it how to interpret the vertices that are declared in the `Begin() .. End()` block.

Next, we have a few vertex declarations, and finally, a call to `glEnd()`. And we're done! The remainder of the file is pretty much the same as the modern OpenGL program.

4 Deprecated vs modern OpenGL

You might be wondering why this version of OpenGL got deprecated. It's so simple to use, compared to the modern version, right? If we use deprecated OpenGL, we don't have to bother with shaders, buffer objects, attribute indices and graphics cards!

Here are some reasons why you might want to use modern OpenGL:

1. The modern version is *more performant*. It eliminates *client-side rendering*, including *display lists* and functions like `glColor()` and `glVertex3f()`.
2. By removing many slow functions from the spec, the new spec is *smaller, and more streamlined*.
3. Modern OpenGL is gaining support on all platforms. It is the graphics library of the future.

And here are some reasons why you might want to use the old OpenGL:

1. Your hardware does not support modern OpenGL.
2. You are working on a large codebase written in old OpenGL, and it is too costly to do a rewrite.