Tutorial : CS 475 Example 1

Rohan Prinja

August 1, 2014

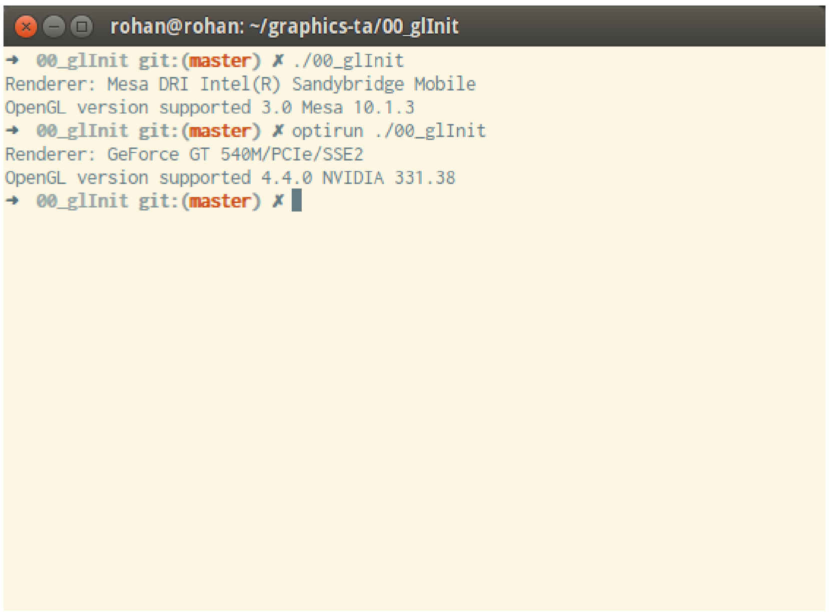
1 About

This is a tutorial for the first code example for CS 475. The code can be downloaded from here. When you untar the downloaded `tgz` file, you will find a cpp file `00_glInit.cpp` and a file named `Makefile`.

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 a single executable file `00_glInit`. To run it, do `./00_glInit`.

When the code is run, it will print the name of the renderer on your laptop and the highest version of OpenGL supported by your laptop. Here is a screenshot of how it should look:



```
rohan@rohan: ~/graphics-ta/00_glInit
➜  00_glInit git:(master) ✗ ./00_glInit
Renderer: Mesa DRI Intel(R) Sandybridge Mobile
OpenGL version supported 3.0 Mesa 10.1.3
➜  00_glInit git:(master) ✗ optirun ./00_glInit
Renderer: GeForce GT 540M/PCIe/SSE2
OpenGL version supported 4.4.0 NVIDIA 331.38
➜  00_glInit git:(master) ✗
```

Note that if you have an Nvidia graphics card, you might need to explicitly force it to run the program. Instead of running `./00_glInit` you should run `optirun ./00_glInit`. See the above screenshot for reference. Without the graphics card, my laptop only supports OpenGL 3.3 and below. With it, it supports OpenGL 4.4 and below.

3 Understanding the code

Now let's understand what exactly the code is doing, line-by-line.

To start with, we include the GLEW library.

```
#include <GL/glew.h> // include GLEW and new version of GL on Windows
```

Many graphics card vendors make extensions to the OpenGL specification by adding functions and constants or slightly modifying the behaviour of core functions. They do this so that OpenGL performs better – that is, it runs faster and consumes less memory. GLEW's job is to find out which extensions to the OpenGL API are supported on the current machine, and load them if possible.

Next, we include the GLFW library.

```
#include <GLFW/glfw3.h> // GLFW helper library
```

GLFW is a utility library for OpenGL. As explained in class, OpenGL is a very bare-bones library, and it does not concern itself with the creation and management of windows. As far as it is concerned, it is your job as a programmer to set up windows and manage their dimensions and so on. It only cares about having someplace that it can draw into.

This is where GLFW comes in. It makes the job of creating and managing windows easy. It also helps in receiving input from the keyboard, mouse buttons and even joysticks.

We need to output stuff to the terminal, so we pull in `iostream`

```
#include <iostream>
```

Now we enter the main function.

```
int main (int argc, char** argv)
{
  // start GL context and O/S window using the GLFW helper library
  if (!glfwInit ())
    {
      std::cerr<<"ERROR: could not start GLFW3"<<std::endl;
      return 1;
    }
```

`glfwInit()` attempts to initialize the GLFW library. Needless to say, it fails if something is wrong with your installation of GLFW, or if you lack some other important component. If it fails, it calls the `glfwTerminate()` function. If it succeeds, it is your job as the programmer to call `glfwTerminate()` before the main program ends.

Next, we give some hints to GLFW about what version of OpenGL we want to use, whether we want forward compatibility or not, and what OpenGL profile to use. If you are not using the OS X operating system, you can safely comment out these lines.

```
  glfwWindowHint (GLFW_CONTEXT_VERSION_MAJOR, 3);
  glfwWindowHint (GLFW_CONTEXT_VERSION_MINOR, 2);
  glfwWindowHint (GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
  glfwWindowHint (GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
```

To understand the meaning of the last line, you need to know the history of OpenGL. When OpenGL 3.0 came out, it was decided for the first time that some parts of the OpenGL API could be marked as deprecated, meaning that in future versions they would cease to be part of the spec. However, in OpenGL 3.1, an extension named ARB_`compatibility` was introduced. If the machine running an OpenGL program supports this extension, it is capable of using the deprecated OpenGL features.

So what do the above lines do? Quoting from the FAQ on the GLFW site:

"The only OpenGL 3.x and 4.x contexts currently supported by OS X are forward-compatible, core profile contexts. The supported versions are 3.2 on 10.7 Lion and 3.3 and 4.1 on 10.9 Mavericks. In all cases, your GPU needs to support the specified OpenGL version for context creation to succeed."

Next, we create a window using GLFW. Note that we don't have to worry about operating system-specific details. GLFW takes care of that for us.

```
GLFWwindow* window = glfwCreateWindow (640, 480, "OpenGL Initialization Example", NULL, NULL
  );
if (!window)
{
  std::cerr<<"ERROR: could not open window with GLFW3"<<std::endl;
  glfwTerminate();
  return 1;
}
```

The first two parameters to `glfwCreateWindow` are the width and height of the window. The third is the window name. The fourth parameter is `NULL`, indicating that we don't want to use fullscreen mode. Ignore the fifth parameter for now (if you're curious, look here).

```
glfwMakeContextCurrent (window);
```

This line tells GLFW to set the context contained by the recently created window as the current context. OpenGL has a context which encapsulates all render state, textures and shaders. You can't use the OpenGL API unless you have a context. When we called `glfwCreateWindow`, it created an OpenGL context. This line tells OpenGL that this context should be made the current one. Remember, OpenGL won't do any of this stuff for us. We are using the helper library GLFW to take care of these details involving the window/context.

```
// start GLEW extension handler
glewExperimental = GL_TRUE;
glewInit ();
```

These two lines initialise GLEW. `GL_TRUE` is an enum that resolves to `true`. `glewExperimental` is a global flag provided by GLEW. If set to true, it enables experimental extensions (along with regular extensions).

```
// get version info
const GLubyte* renderer = glGetString (GL_RENDERER); // get renderer string
const GLubyte* version = glGetString (GL_VERSION); // version as a string
std::cout<<"Renderer: "<<renderer<<std::endl;
std::cout<<"OpenGL version supported "<<version<<std::endl;
```

We read in the GL_RENDERER and GL_VERSION strings into `GLubyte*` (unsigned byte) variables, then print them out. `glGetString` can return a lot of information about your machine. Look here for details.

```
  // close GL context and any other GLFW resources
  glfwTerminate();
  return 0;
}
```

As promised, we call `glfwTerminate()` and exit the main function after returning `0`.