3D Graphics Programming

Lab 1: Rendering a Triangle

Karsten Pedersen
Department of Creative Technology

August 7, 2018

In this unit we aim to use an industry standard graphics API to draw and interact with 3D objects. *OpenGL* as an API is extremely flexible but also comes with quite a bit of complexity. Developing with *OpenGL* can often require some initial time to get used to how it works and also to set up the data used for the rendering.

The first thing we will need to do is to ensure that the project is set up and ready to use *OpenGL*. To just get a window appearing on the screen we will be using *SDL 2* because not only is the code required to do so fairly trivial but also because you are already familiar with it. Other libraries that are also common for this task include *[Free]Glut*, *GLFW*, *SFML* and *Allegro*. Open the provided *Microsoft Visual Studio* project or generate the *CMake* build system. The code inside ***main.cpp*** should be familiar to you by now. It simply opens up the *SDL 2* Window and not much else. You will see that we are not even creating an ***SDL_Renderer*** because we are going to be using *OpenGL* directly rather than using the basic renderer provided by *SDL 2* (which uses either *OpenGL* or *DirectX* underneath depending on platform.

Note:
Microsoft Windows provides a version of *OpenGL* which is extremely old (version **1.2**) and is too inflexible to make what we see today as modern games. Luckily a much newer implementation of *OpenGL* is provided by the graphics card manufacturer's driver (i.e *NVIDIA*, *AMD*, *Intel*, *MESA*). As of writing we are at around version **4.6** which only the very latest hardware supports. However the techniques and functionality covered in these labs was actually provided by version **2.1** which means that your code will work on almost all hardware found in the wild.

For convenience, rather than use a specific header and library (i.e ***GL/gl.h***) from each different vendor's SDK, we instead use a 3rd party library called *Glew* (Open**GL** **E**xtension **W**rangler). This library links the vendor's specific implementation with the platforms implementation at runtime so we can just use *OpenGL* as usual and not worry about the details.

First, lets begin by including the *SDL 2*/*OpenGL* binding layer header and the *Glew* header file into our project.

```
#include <SDL2/SDL_opengl.h>
#include <GL/glew.h>
```

Next we need to create an *OpenGL* rendering context within the created *SDL 2* window and because *Glew* loads the *OpenGL* library and extensions at runtime we also need to initialize it. In your project, just after where you open the *SDL 2* window using ***SDL_CreateWindow***, add the following function calls.

```
if(!SDL_GL_CreateContext(window))
{
  throw std::exception();
}

if(glewInit() != GLEW_OK)
{
  throw std::exception();
}
```

Note:
This must be **after** the call to open the window because otherwise there is no window for *OpenGL* to bind a context to and the call will fail.

With this in place we are now ready to start with *OpenGL*. What we are first going to do to confirm everything is working is change the screen to the color red. The following listing will first set the current *OpenGL* clear color to red, will then actually instruct *OpenGL* to clear the screen and finally it will atomically swap the *OpenGL* memory buffer with that of the screen buffer (to eliminate flicker).

```
glClearColor(1.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
SDL_GL_SwapWindow(window);
```

With that in place, compile the project and run it. You should hopefully see the following.

TODO
Big red OpenGL window

Now we can finally begin using the main essence of *OpenGL*. In 3D graphics almost everything is made up of triangles so lets start with trying to draw one. **Figure 1** should give you an overview of the tasks required to do this.

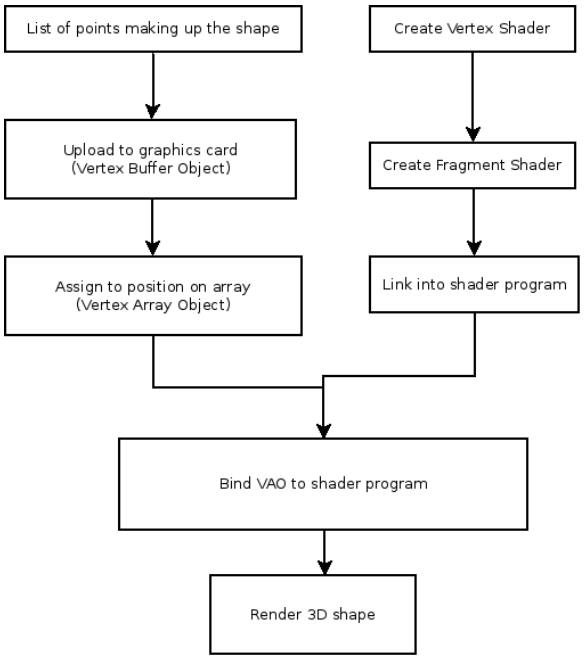

Figure 1: *Diagram showing the series of tasks in order to render a triangle*