OpenGL exercises

Riccardo Salvalaggio

17th of May, 2021

Contents

1 Introduction

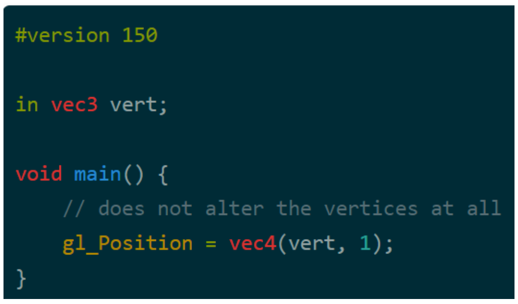
The **OpenGL Extension Wrangler (GLEW)** is what we will be using to give us access to the OpenGL 3.2 API functions. In modern OpenGL, the API functions are determined at run time, not compile time.

GLFW will allow us to create a window, and receive mouse and keyboard input in a cross-platform way.

OpenGL Mathematics (GLM) is a mathematics library that handles vectors and matrices, amongst other things.

Shaders: Shaders are little programs, made from GLSL code, that run on the GPU instead of the CPU. In older version of OpenGL, shaders were optional. In modern OpenGL, shaders are required. What they do depends on types.

- **Types of shaders:**
1. Vertex shaders: transform points.

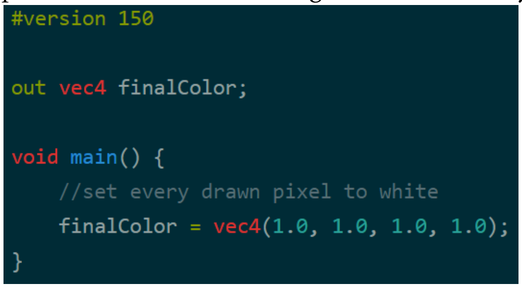


```
#version 150

in vec3 vert;

void main() {
    // does not alter the vertices at all
    gl_Position = vec4(vert, 1);
}
```

First shader: in vec3 vert is just a 3d vetex, gl_position is needed in every shader, in this case just transform the 3d vertex in a 4d one without modifying anything.
2. Fragment shaders: calculate the color of each pixel that is drawn. A “fragment” is basically a pixel, so you can think of fragment shaders as “pixel shaders.”



```
#version 150

out vec4 finalColor;

void main() {
    //set every drawn pixel to white
    finalColor = vec4(1.0, 1.0, 1.0, 1.0);
}
```

First fragment shader: deine output colored variable, vec4(R,G,B,α).

1.1 VBO and VAO

Vertex Buffer Objects (VBOs) and Vertex Array Objects (VAOs) are used to take data from your C++ program and send it through to the shaders for rendering (exchnge data between CPU and GPU).
VBOs are ”buffers” of video memory – just a bunch of bytes containing any kind of binary data you want.
VAOs are the link between the VBOs and the shader variables. VAOs describe what type of data is contained within a VBO, and which shader variables the data should be sent to.

> In this article, we will use a VAO to say “hey OpenGL, this VBO right here has 3D points in it, and I want you to send those points to the ‘vert’ variable in the vertex shader.”

2 Exercise 1

First initialise GLFW for error handler, then to create window *glfwwindowshint*, *glfwCreateWindow* and *glfw-MakeContextCurrent(gWindow);*.
Now we have the windows, we initialise GLEW to access API.
Inside the LoadShaders function, we compile and link a vertex shader and a fragment shader. Inside the Load-Triangle function, we are going to make one VBO and one VAO. Next, we upload some data into the new VBO. It is time to set up the VAO. First, we are going to enable the vert variable in the shader program. The most complicated part of VAO setup is this next function: glVertexAttribPointer:

```
glVertexAttribPointer(gProgram− >attrib(”vert”), 3, GL_FLOAT, GL_FALSE, 0, NULL);
```

Now that the VBO and VAO are fully set up, we unbind them so they don't accidentally get used somewhere else. At this point, the shaders, VBO, and VAO are ready for use. All we have to do now is draw them inside the Render function.
First we clear the screen so that it is completely black:

```
glClearColor(0, 0, 0, 1); // black
glClear(GL_COLOR_BUFFER_BIT — GL_DEPTH_BUFFER_BIT);
```

Next we tell OpenGL that we want to start using our shaders and our VAO:

```
glUseProgram(gProgram-¿object());
glBindVertexArray(gVAO);
```

At last, we can draw that ever-elusive triangle:

```
glDrawArrays(GL_TRIANGLES, 0, 3);
```

This call to glDrawArrays says that we want to draw triangles, starting at vertex zero, and ending after three vertices have been sent to the shader. It will look at the currently bound VAO to determine where to get the vertices from. The vertices will be pulled out of the VBO and sent to the vertex shader. The drawing is finished now, so we unbind the shaders and the VAO just to be safe
The last thing that needs to be done before we can see the triangle is to swap the frame buffers:

```
glfwSwapBuffers(gWindow);
```

Before the frame buffers were swapped, we were drawing to an off-screen frame buffer that was not visible in the window we created at the start.

3 Exercise 2