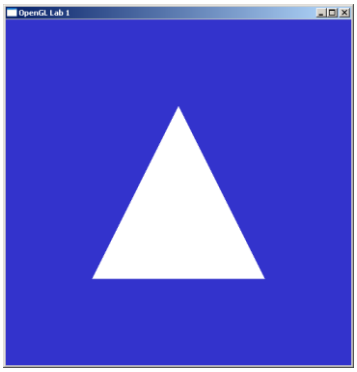
Lab 1 - Basics

Introduction

In this tutorial we will start familiarizing ourselves with OpenGL. To do this, there is a simple OpenGL application that you will study. Make sure you read all the comments, and understand what each command does. If anything is unclear, refer to the OpenGL 3.0 specification, where pretty much everything is described, and in a quite readable way, believe it or not! Even more OpenGL documentation is available in the OpenGL registry. If this fails to clarify ask one of the lab assistants for help, as you will need to understand this for later tutorials.

Your Task



The program we will work with draws a single, white, triangle on screen. Your first task is to **draw a second triangle**. This shall be accomplished by creating a new vertex array object in the *initGL()* function and then draw that object in the *display()* function. The second triangle must not cover the existing triangle, you are otherwise free to place both triangles as you please.

Next, you shall **add a third triangle** to the scene. This time, you shall not create a new VAO. Instead, simply add position and color data to the previous VAO.

Assignment: The glBindBuffer command (for example) has an input parameter called *target*. What possible targets are there and what is the difference between them? Look in the OpenGL specification (§2.9 Buffer Objects) and write your answer here:

_____

_____

Did you try to change the colors of any triangle yet? You'll notice nothing happens as the vertex- and fragment-shaders are not complete yet. Your **third task is to fix this**. Take a look at the *simple.vert* file. The vertex color is declared as the attribute *color* in the beginning of this shader, but from then on it is ignored.

We'll need to pass the vertex color value on to the fragment shader so declare a second output from the vertex shader (before *main()*) like this:

```
out vec3 outColor;
```

Then, set this output variable in the *main()* function:

```
outColor = color;
```

Now, open the fragment shader (*simple.frag*) and tell it to expect a variable *outColor* from the vertex shader:

```
in vec3 outColor;
```

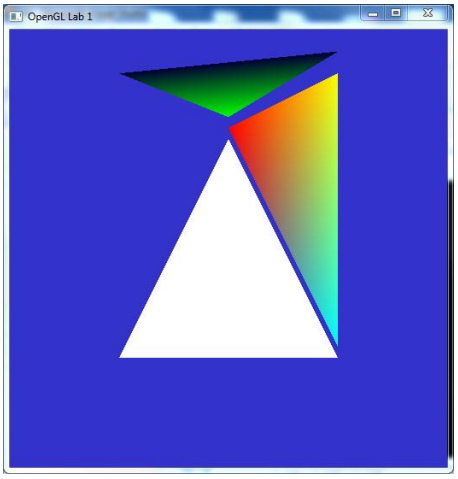
And then change this line that currently sets all fragments to be white:

```
fragmentColor = vec4(1,1,1,1);
```

to instead use the color passed in from the vertex-shader:

```
fragmentColor.rgb = outColor;
```

There! Now make sure your vertex array objects have some fun colors in them and run the program again. The final solution might look something like this:



To reiterate; the purpose of this tutorial is to understand ***how*** the triangles end up on screen. Working in groups it may be a good idea to interrogate one another, let one explain to the other(s) what each line does, the person listening should make sure that he or she is satisfied that the answer really explains what is going on. And again if the OpenGL specification fails to help, ask an assistant!