TDT4195 - Visual Computing Fundamentals - Assignment 1

Group 119
Martin Eek Gerhardsen

September 6th, 2019

Department of Engineering Cybernetics

Contents

1 Task 1: Drawing your first triangle

The first two subtasks are done in the code, which will be attached to the submitted report. Some of this code is added in new functions added in the *src* folder, as the files *triangle.cpp* and *triangle.hpp*.

1.1 c)

For the resulting triangles drawn, see fig. 1. They were added next to each other so that it was easy to reuse the same coordinates, and shift them in the x-axis, but could of course be shifted to other places in the code, for a more interesting layout.

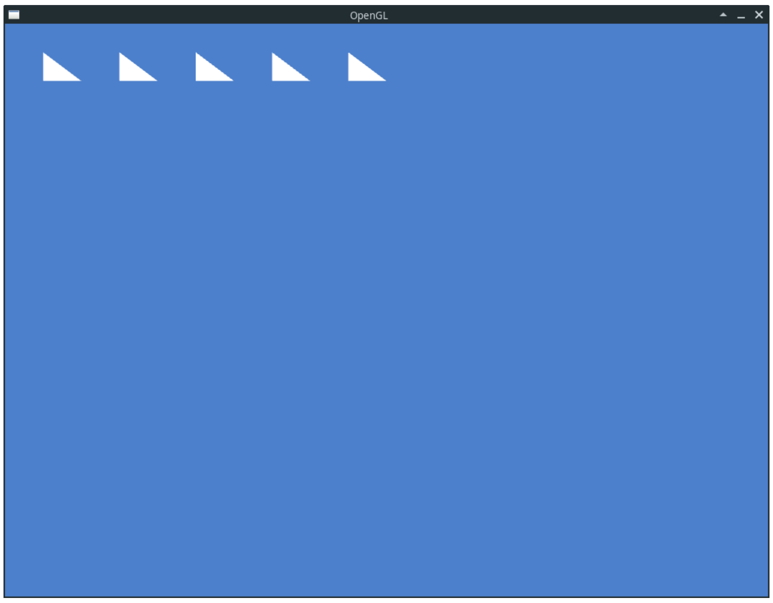

Figure 1: Five triangles drawn

2 Task 2: Drawing your first triangle

2.1 a)

For the resulting triangles drawn, see fig. 2

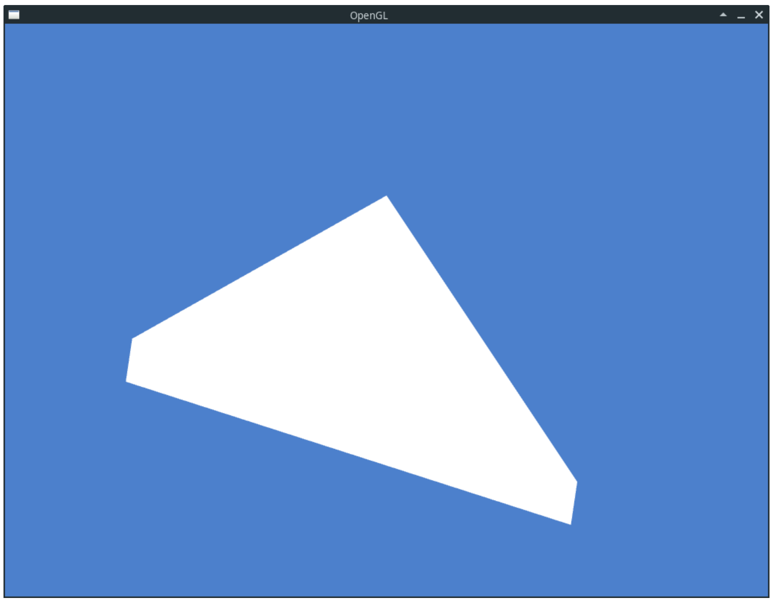

Figure 2: One triangle drawn outside the clipbox

2.1.1 i)

This phenomena is called clipping.

2.1.2 ii)

The problem occurs when coordinates are set outside the clipbox, or more specifically when the coordinate values are outside the range $[-1, 1]$. This problem, with the triangle looking like a poorly drawn cheese, is due to the z-axis being outside the $[-1, 1]$ range, so that part of the triangle "disappears" into and out of the clipbox. We don't really notice changes in the z-axis, as the scene is projected onto a 2D screen in this case, but here it is noticable as the corners now are outside the clipbox.

2.1.3 iii)

The purpose is to remove geometry which is not supposed to be visible in the rendered image.

2.2 b)

For the resulting triangles drawn, see fig. 3 and fig. 4.

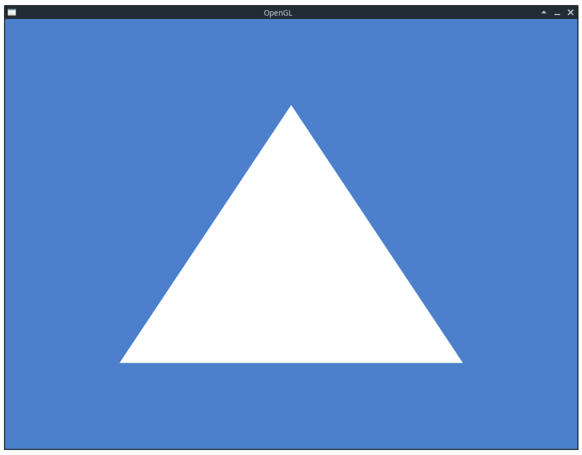

Figure 3: One triangle drawn

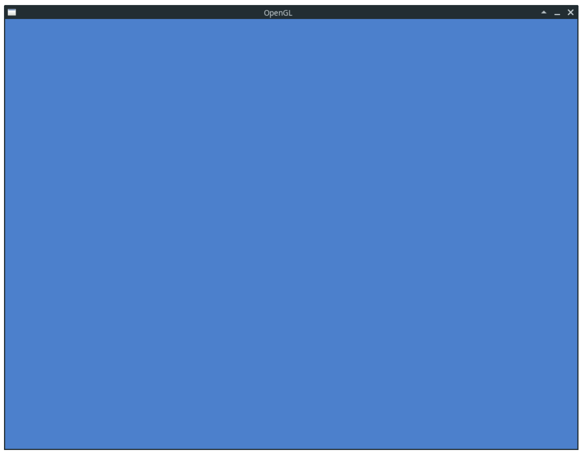

Figure 4: One triangle drawn with indices reversed

2.2.1 i)

The triangle disappears (Face culling).

2.2.2 ii)

We are looking at the back of the triangle. Triangles are defined to have a certain face, following the indices counter-clockwise (unless otherwise specified). There are several reasons why this is a generally good thing, like when constructing 3D-objects, there are usually no reason to care about the inside of the objects, as they should't be viewed anyways, and it is therfore useful to ignore them, and save some time / work.

2.2.3 iii)

As mentioned in section 2.2.2, the rule is that the face of the triangle is shown on the side according to the right hand rule. By laying the fingers (except the thumb) of our right hand along the path of the vertices according to the indices, our thumb will show which side the face of the triangle is.

2.3 c)

2.3.1 i)

The depth buffer needs to be reset, as it is used to compare depth when new pixels are drawn. If the depth buffer is kept, then we will compare against previous values, which would give wrong values.

2.3.2 ii)

In cases where several objects are drawn on top of each other, the Fragment Shader may be executed multiple times.

2.3.3 iii)

Fragment Shaders and Vertex Shaders.

- The Vertex Shader is run once for each vertex drawn. It is responsible transforming the vertices, as well as projecting the scene onto the camera.
- The Fragment Shader is responsible for the colour of the fragments, and is called for every fragment.

2.3.4 iv)

Because many objects will reuse the same vertices, and we can save significant time and energy by rather assigning the same point to different shapes. I.e., when drawing a cube, all the corners will be used several times to draw the full object.

2.3.5 v)

The *pointer* defines the offset in the buffer until it should start reading. If texture has been defined in the same buffer as the coordinates, then this offset must be included so that we aren't reading coordinates when we want to read the texture.

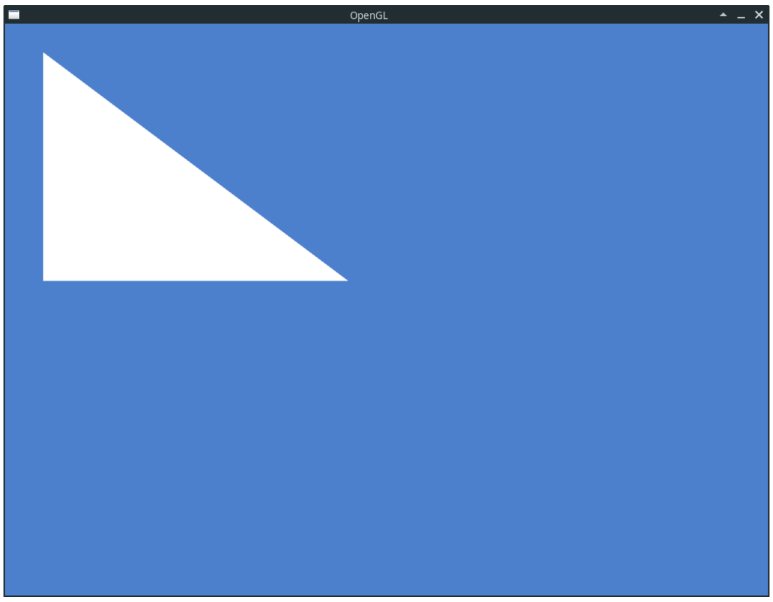

Figure 5: One triangle drawn

d)

2.3.6 i)

The mirror effect was simply achieved by flipping the sign of the x axis and y axis in the Vertex Shader. I was uncertain what the question of this task was (if it was necessary to keep the original before the mirror), but assumed we just needed to transform the original shape. See fig. 6.

2.3.7 ii)

The colour change was achieved by changing the value of the *colour* variable in the Fragment Shader. For fun, I found a function for rainbow colour online (linked in the code), so that the fragment got a nice rainbow colour. See fig. 7. For the complete, combined effect, see fig. 5.

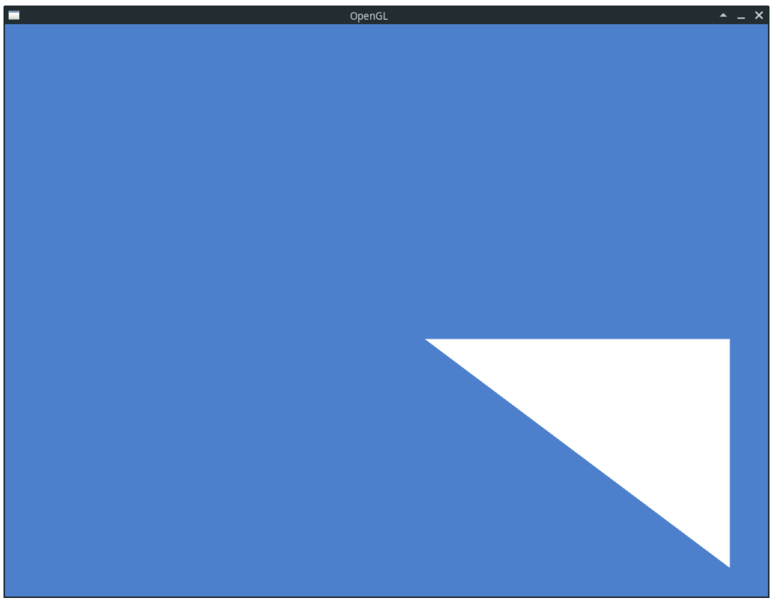

Figure 6: One triangle drawn and mirrored horizontally and vertically

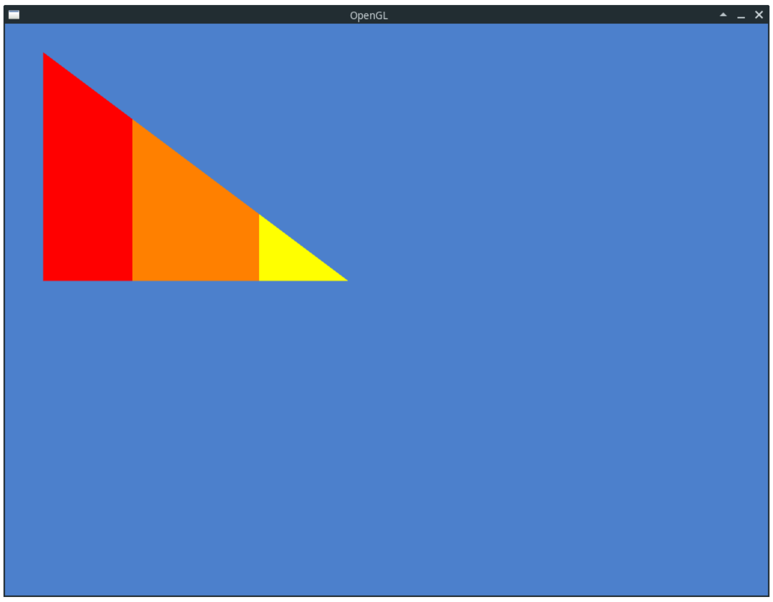

Figure 7: One triangle drawn with changed colour (rainbow)

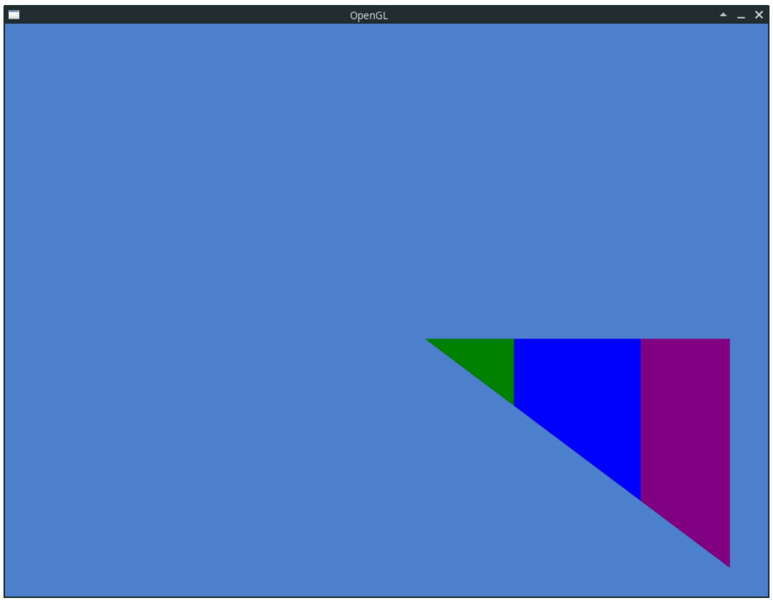

Figure 8: Combined changes

3 Task 3: Drawing your first triangle

3.1 a)

By dividing the x and y coordinates by 10, and then checking if that number is even or odd, we can set the colour to either black or white. If we'd like even larger squares, then we'd divide by a larger number.

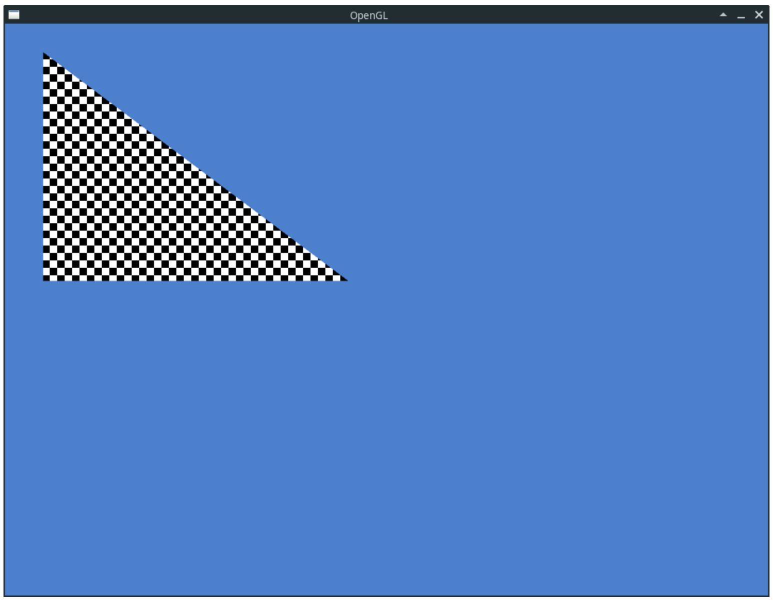

Figure 9: Checkerboard

3.2 b)