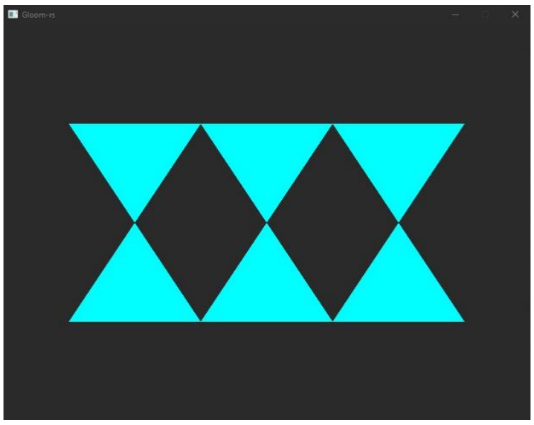
TDT4195 – Assignment 1

Task 1

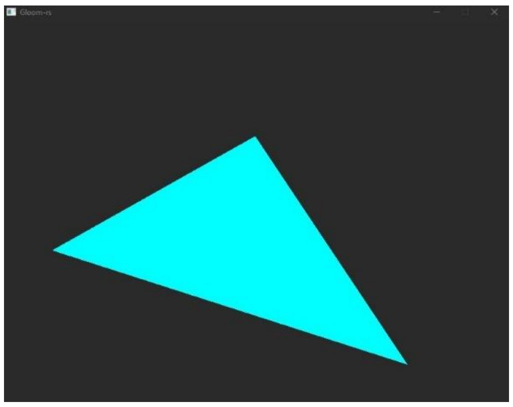


Task 2

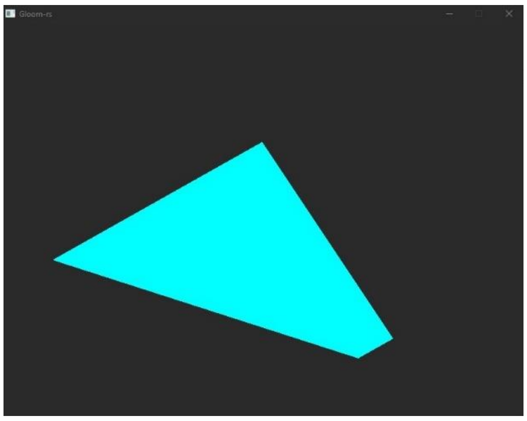
2.a

2.a.1

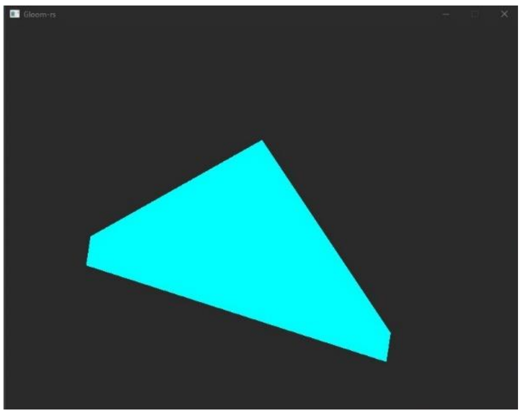
When we set all *z* values to 0.0, we get this shape:



If we set the first *z* value to -1.2 instead of 0.0, we get the shape below. Why, because the first *z* value is 1.2, which is outside the clip box, so not everything is rendered. This phenomenon is called **clipping**.



When we set the first *z* value to -1.2 and the last to 1.2, we get this shape. Notice that both the first and the last *z* indexes are > 1.0, so they are not rendered. Therefore, the triangles edges seem "cut off".

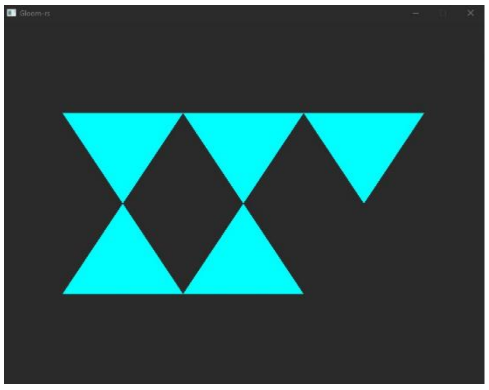


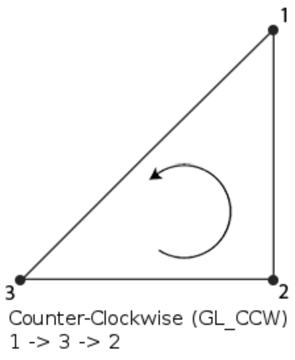
2.a.2
Clipping occurs when we try to render somethiWeng outside the clip box (clipping volume).

2.a.3
OpenGL uses clipping to save performance. It does not need to render things that are not visible.

2.b
2.b.1
The affected triangle disappears.

2.b.2
The triangle disappears because of **culling**. The ordering of the coordinates matter! We need to draw the triangle in a counterclockwise matter. If we draw the triangle in a clockwise matter, we are looking at the triangle from its back, so it is not rendered.





2.b.3
As mentioned, the rule is counterclockwise. This can be changed in OpenGL.

2.c
2.c.1
The depth buffer should be cleared for each frame to remove the depth values from the previous frame. When we redraw a frame, we want to use depth values (*z* indexes) from this

render, not the previous. If we forget to clear it, some pixels may not be rendered because they are “behind” some pixels from the previous frame.

2.c.2

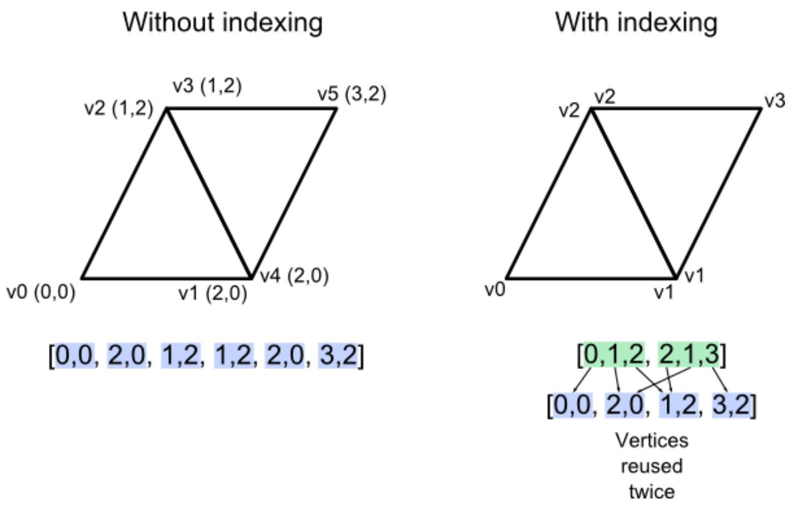
As the OpenGL paper said: “The Fragment Shader is executed once for every single fragment.”. Typically used for complex scenes.

2.c.3

- Vertex shaders - responsible for transforming (translating, scaling, rotation etc.) each vertex. It is run once for each drawn vertex.
- Fragment shaders - responsible for the colour of each fragment. A fragment is a pixel that OpenGL attempts to draw. It has a depth; therefore, it can be behind another fragment which leads to it not being drawn.

2.c.4

Memory usage saving. Triangles often share the same vertices, so we can reference them instead of storing them redundantly.



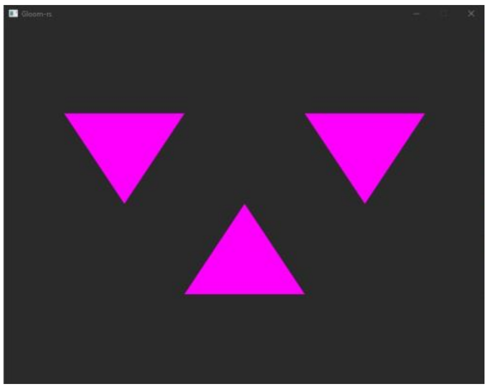
2.c.5

The pointer parameter defines the number of bytes until the first value in the buffer. If we only use a single entry-type (e.g. floats) we can pass in the null pointer. However, if we have multiple entry-types (e.g. floats and integers), we need to specify which index they start from.

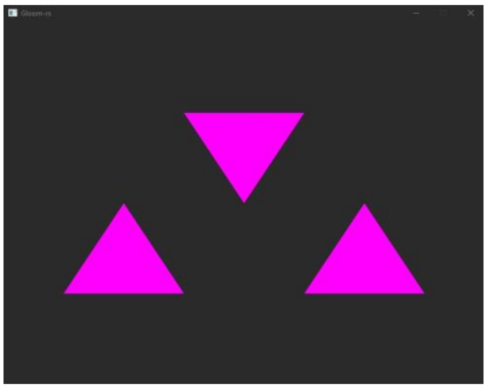
2.d

2.d.1

Since my image was symmetrical, I had to adjust it:



Now we reverse the *x* and *y* coordinates in the vertex shader:



2.d.2

Swapped red and green values:
color = vec4(1.0f, 0.0f, 1.0f, 1.0f); instead of color = vec4(0.0f, 1.0f, 1.0f, 1.0f);