TDT4195: Visual Computing Fundamentals

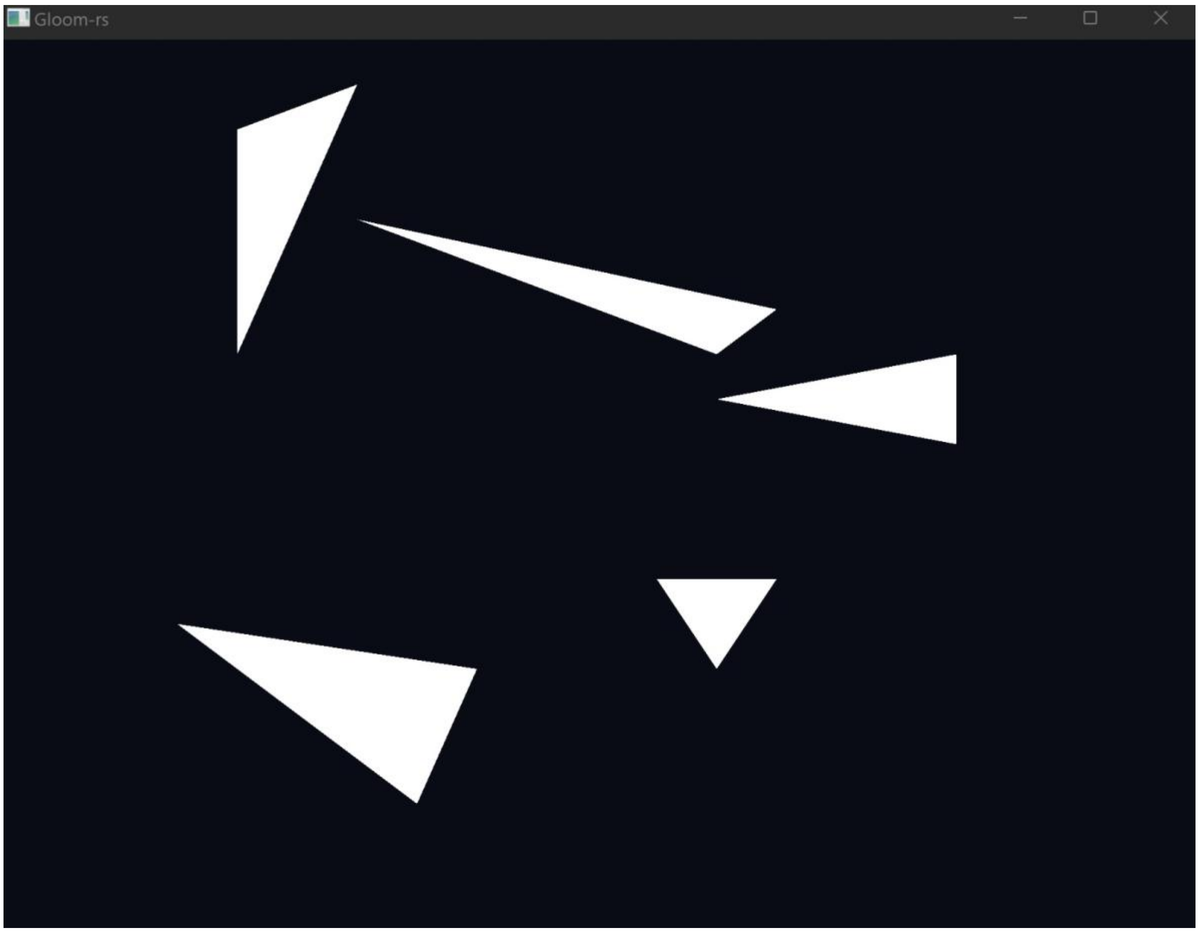
Computer Graphics - Assignment 1

Department of Computer and Information Science Norwegian University of Science and Technology (NTNU)

Caroline Maclaren / 106019 / Marta Almagro / 106011

Task 1: Drawing your first triangle [2.5 points]

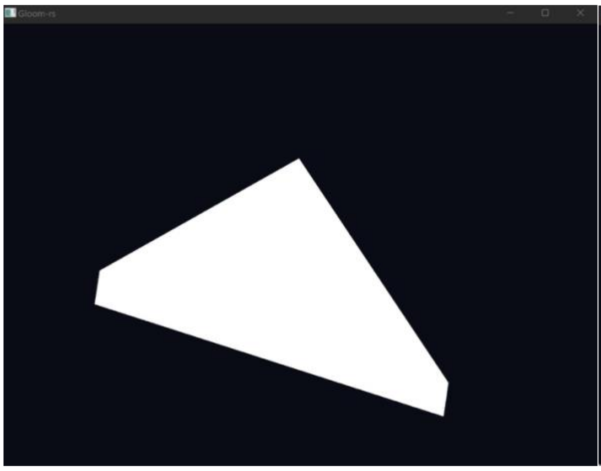
See code

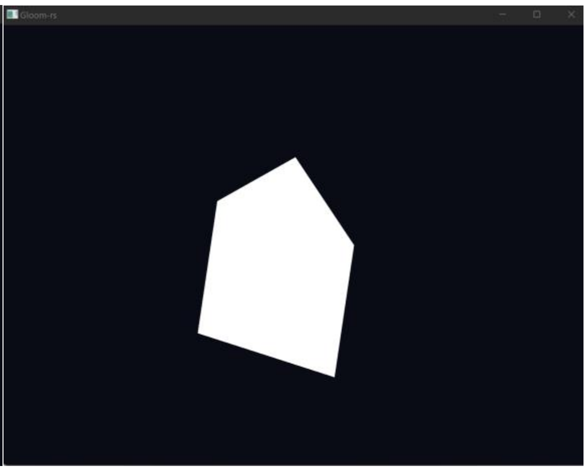


Task 2: Geometry and Theory [1.5 point]

a) [0.4 point] [report] Draw a single triangle passing through the following vertices. Put a screenshot of the result in your report. This shape does not entirely look like a triangle. Explain in your own words:

i) What is the name of this phenomenon? This phenomenon is called clipping, we can observe that the clipping starts after 1. The second picture is when we put the w-axis with three, where the vertex is more outside so we would rasterize a smaller part of the triangle.



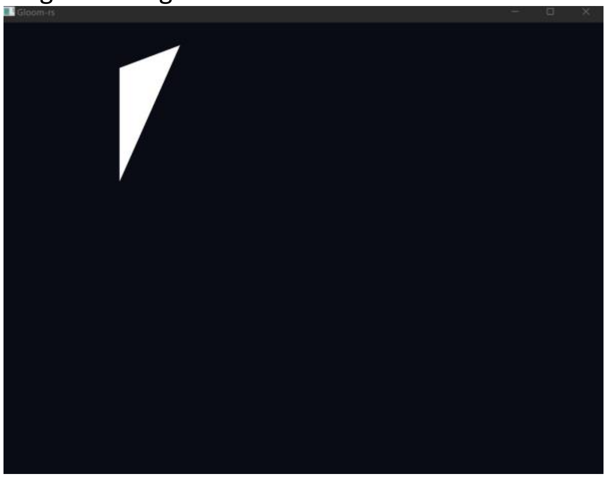


ii) When does it occur? It occurs when we are giving out-of-range values to a display device.

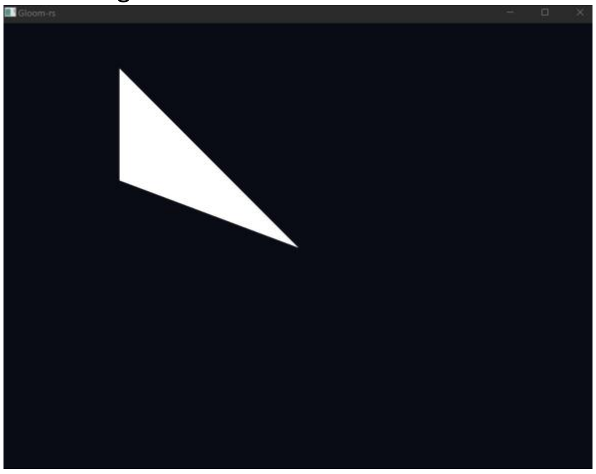
iii) What is its purpose? The purpose is to avoid out-of-range values, therefore, to save computer energy and power to avoid crashing if out of range.

b) [0.4 point] [report] While drawing one or more triangles, try to swap the order in which two of the vertices of a *single* triangle are drawn by modifying the index buffer. A significant change in the appearance of the triangle should occur. Show a screenshot in your report.

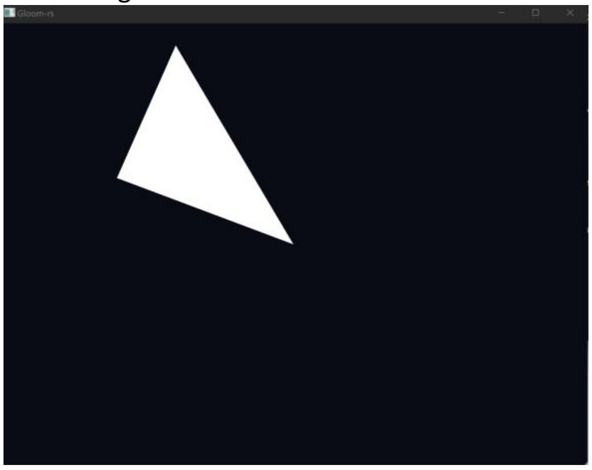
Original triangle:



One change to index buffer:



Two changes to index buffer:



i) What happens? ii) Why does it happen? iii) What is the condition under which this effect occurs? Determine a rule.

Here, we are able to swap two of the vertices of our triangle by changing our index buffer. First, we swap the indices. As a result, this changes where a vertex will actually appear, so it essentially changes every vertex of a triangle when it's rasterized. As such, our triangle looks like a new triangle, where it's rotated counterclockwise. We changed another set of indices too and as expected, the vertices changed. So, we have a different triangle that has rotated and created more of a right triangle. If you change the order of the vertices, it will then change the order of how the vertices are read, thus creating a new triangle that is rotated clockwise or counterclockwise. The rule that is happening here is back-face culling.

c) [0.5 point] [report] Explain the following in your own words:

> **i) Why does the depth buffer need to be reset each frame?** It is necessary to clean it with the function gl::Clear() because then the contents of the previous frame will still be present in the framebuffer, and we would not be able to see just the triangle we want to show. We would see all the one from before.

ii) In which situation can the Fragment Shader be executed multiple times for the same pixel? *(Assume we do not use multisampling.)*

The fragment shader can be executed multiple times for the same pixel when another piece of geometry is drawn over the fragment at a later time while rendering the frame.

For example, when rendering transparent objects, the order of them could affect the final results and how is seen. That is why fragment shader may be executed multiple times for the same pixel as the transparent objects are blended together.

iii) What are the two most commonly used types of Shaders? What are the responsibilities of each of them?

The most commonly shader are the vertex and fragment shaders, which are the ones that we actually have in our folder of shaders. The vertex shader has a function to convert the properties of the vertices that define shapes (in our case triangles) of the 3D model. While the fragment shaders are the responsible for assigning a color to the image and all the properties related to that.

iv) Why is it common to use an index buffer to specify which vertices should be connected into triangles, as opposed to relying on the order in which the vertices are specified in the vertex buffer(s)?

An index buffer is helpful when creating shapes to avoid redundancy and save computer power. If we are using the same vertices several times, we can define the vertices once, and use their index to refer to it. If we rely on the order the vertices are specified, it can use a lot of memory and be inefficient over time.

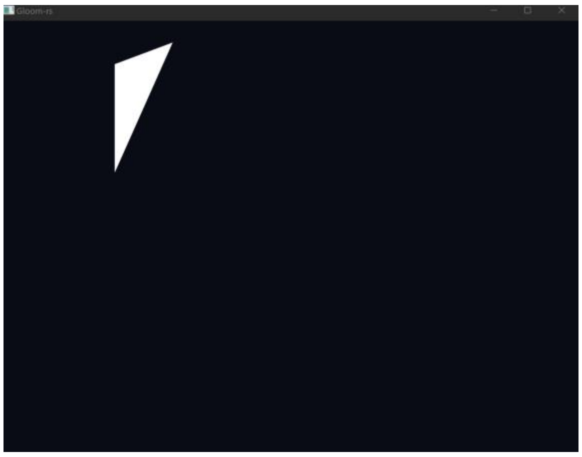
v) While the last input of gl::VertexAttribPointer() is a pointer, we usually pass in a null pointer. Describe a situation in which you would pass a non-zero value into this function.

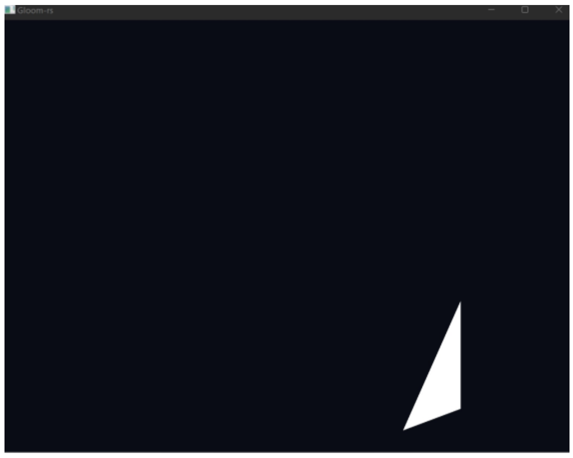
We pass a null pointer because we only have a single-entry type in our buffer. You would pass a non-zero value when you want to use both vertex coordinates and texture coordinates. The texture coordinates would start at a byte bigger than 0. Texture coordinates are used for mapping an image onto a triangle. For example, if we wanted our triangle to be checkered or have some kind of image mapping, we would pass a non-zero value so that we could include both vertex coordinates and texture coordinates into our buffer.

**d) [0.2 point] [report] Modify the source code of the shader pair to:
i) Mirror/flip the whole scene both horizontally and vertically at the same time.**

(Original triangle from part b is what we used)

To mirror/flip the triangle, we edit shader.vert. To mirror/flip we can just scale our shape in the x by -1 and in the by -1.





ii) Change the color of the drawn triangle(s) to a different color.

To edit the color, we edit shader.frag. To change the color to pink, we edited the RGB numbers.