TDT4195: Visual Computing Fundamentals

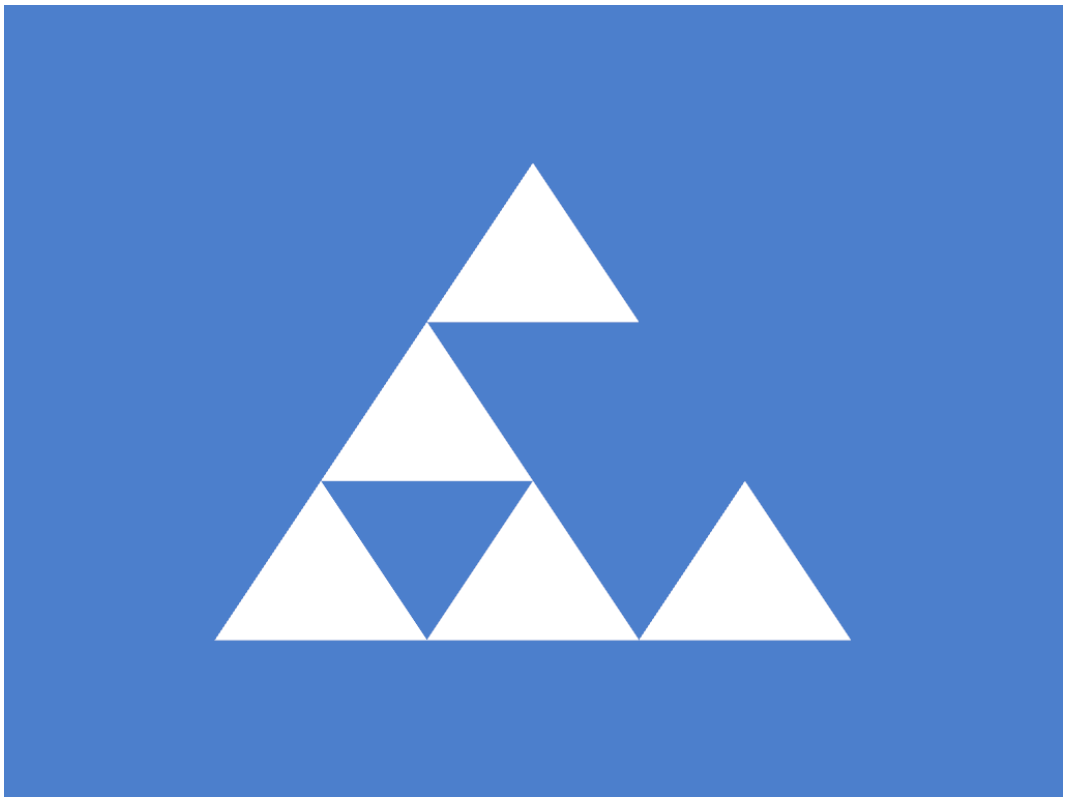
Computer Graphics – Assignment 1

Deadline: 06.09.2019

Name: Niklas Sølvberg

Task 1

Screenshot from 1c). The pattern is "missing" a triangle in such a way that it can be mirrored (later in the assignment) and not look identical to the original.



Task 2

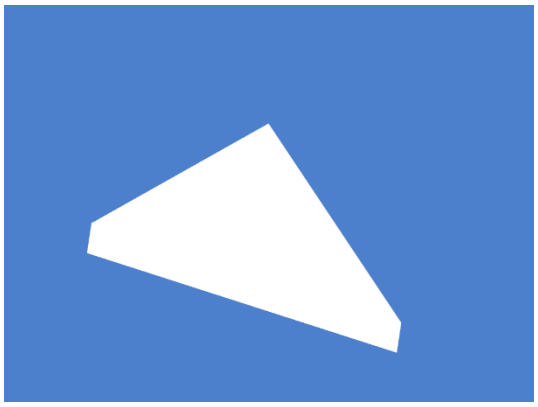


Figure 1

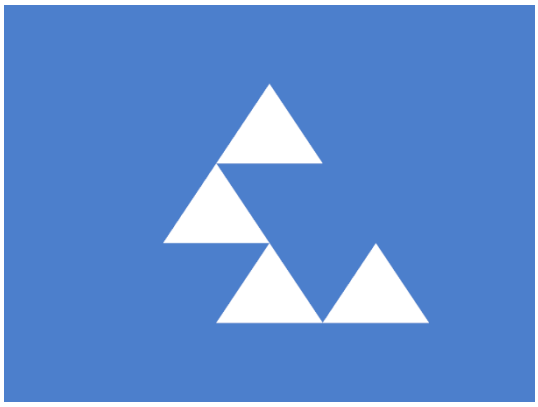


Figure 2

a) (Figure 1) Described at the beginning of task 2.
   i) This phenomenon is called clipping.
   ii) Clipping occurs when you try to render something outside of the clipbox. In this situation the z-value (depth) is the one extending out of the clipbox (z-coordinate of right and left corner is -1.2 and 1.2 respectively, which is slightly outside the clipbox), and therefore parts of the triangle is prone to clipping, such that just the parts within the clipbox is rendered.
   iii) The purpose is to avoid rendering polygons that wouldn't be visible ("not visible" is equivalent to "outside the clipbox").

b) (Figure 2) The triangles I described myself for task 1c, but with one difference: the lower left triangle has its vertices noted in a clockwise order, in contrast to all the other triangles.
   i) What happens here is that the one triangle that had its order of vertices in the index buffer changed, has disappeared.
   ii) This happens because the triangles will have a front- and back-facing side. Back-facing sides shouldn't be visible to us; therefore, it isn't rendered.
   iii) The condition for it to be back-facing is quite simple, and has to do with the winding order; which means it will be back-facing whenever the vertices in the index buffer is written clockwise, and front-facing whenever the winding order is counter-clockwise. Example: let's consider the top triangle in figure 2. Let's name the vertices A (lower left), B (lower right), and C. If this triangle is noted, in the index buffer, as either [A,B,C], [B,C,A], or [C,A,B], it will be front-facing and visible. If it is noted as [A,C,B], [C,B,A], or [B,A,C], it will be back-facing and hence not rendered.

c)
   i) The depth buffer keeps track of which z-value is smallest (so we know what should be drawn). If it isn't reset every frame, that means the old (after last reset) smallest z-value is still present (and can thus be the one to be rendered, even if it isn't supposed to be in this frame), and therefore the buffer needs to be reset every frame.
   ii) The fragment shader can be executed multiple times for the same pixel if two primitives are overlapping. These primitives will share some fragments, and at some

point, one may be discarded due to depth-testing. This is a situation in which the fragment shader has been executed multiple times per pixel.

iii) The two most commonly used types of shaders are fragment shaders and vertex shaders.
- Fragment shader: has the responsibility to determine the colour of each fragment.
- Vertex shader: has the responsibility of moving, scaling and rotating individual vertices, and for projecting the scene onto the camera.

iv) An index buffer lets you reuse a vertex for other triangles without the need to include them multiple times in the vertex buffer. As an example, in task 1, I used just ten vertices to draw 5 triangles. If I'd want to not use an index buffer, I would need five more vertices, or 15 more entries in the vertex buffer, and by scaling this problem up it would become obvious that an index buffer is the way to go.

v) You would pass a non-zero pointer to `glVertexAttribPointer()` if you'd want to add an offset to where to start read the data in the array buffer.

d)

i) I achieved the effect (figure 3) of mirroring vertically and horizontally by going into the `simple.vert` shader and change `gl_Position = vec4(position, 1.0f);` to `gl_Position = vec4(-position, 1.0f);`.

ii) I achieved the effect (figure 4) of changing the colour of the triangles by going into the `simple.frag` shader and change `color = vec4(1.0f, 1.0f, 1.0f, 1.0f);` to `color = vec4(0.8f, 0.4f, 0.8f, 1.0f);`.

Figure 3

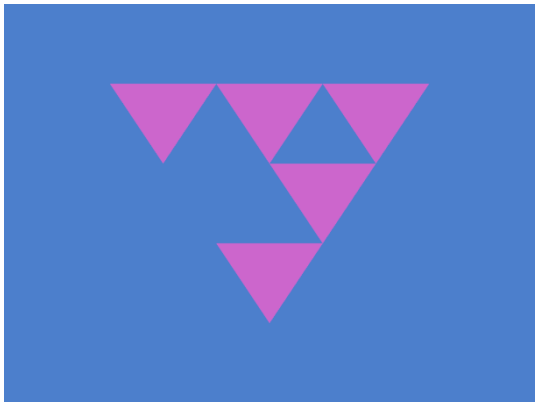


Figure 4