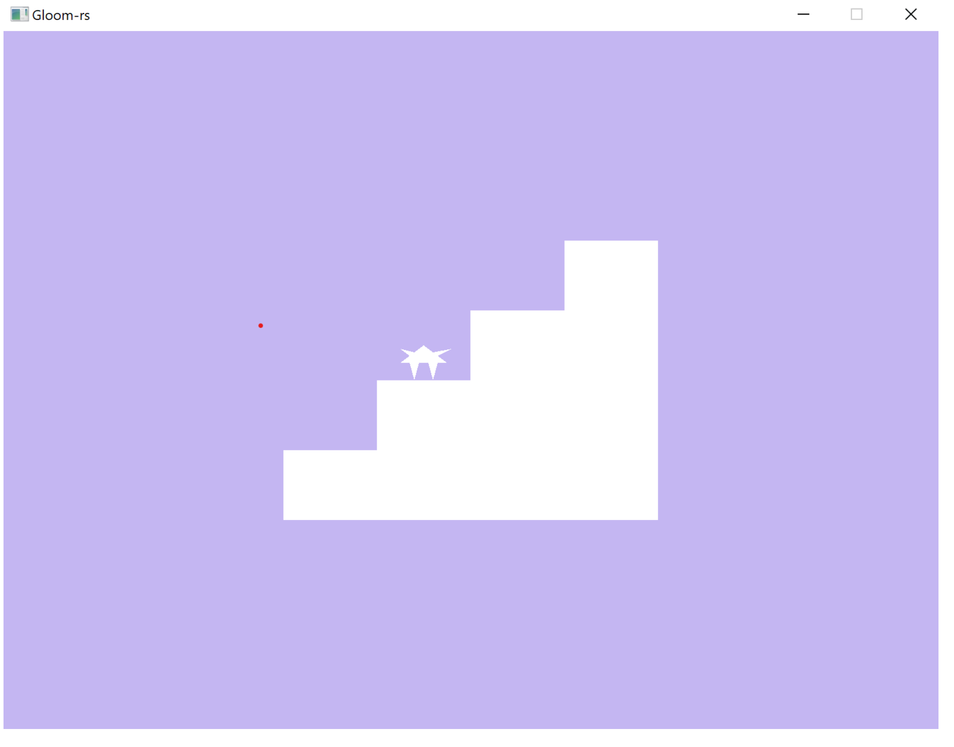
# Øving 1
TDT4195

Even Lauvsnes

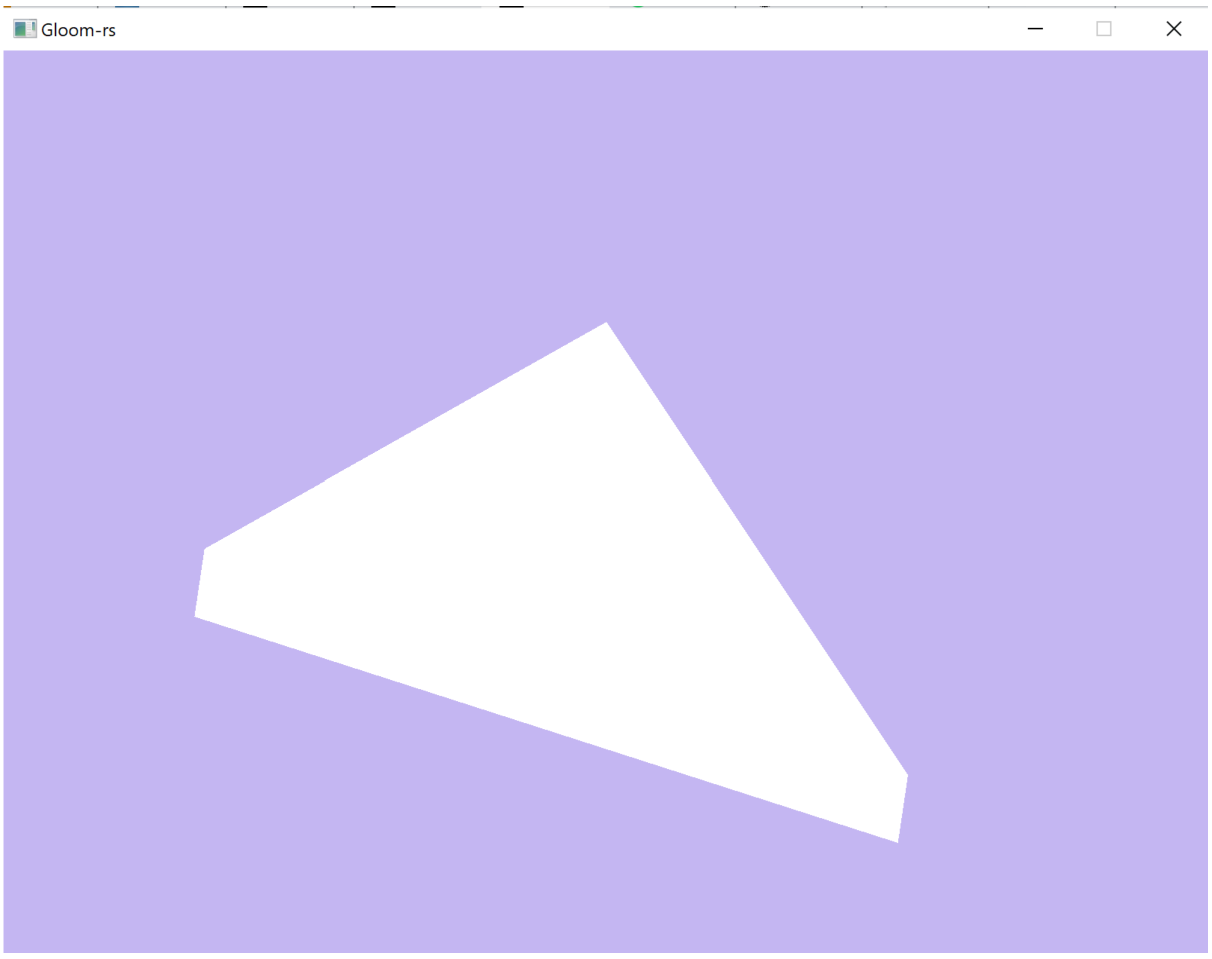
## Oppgave 1)



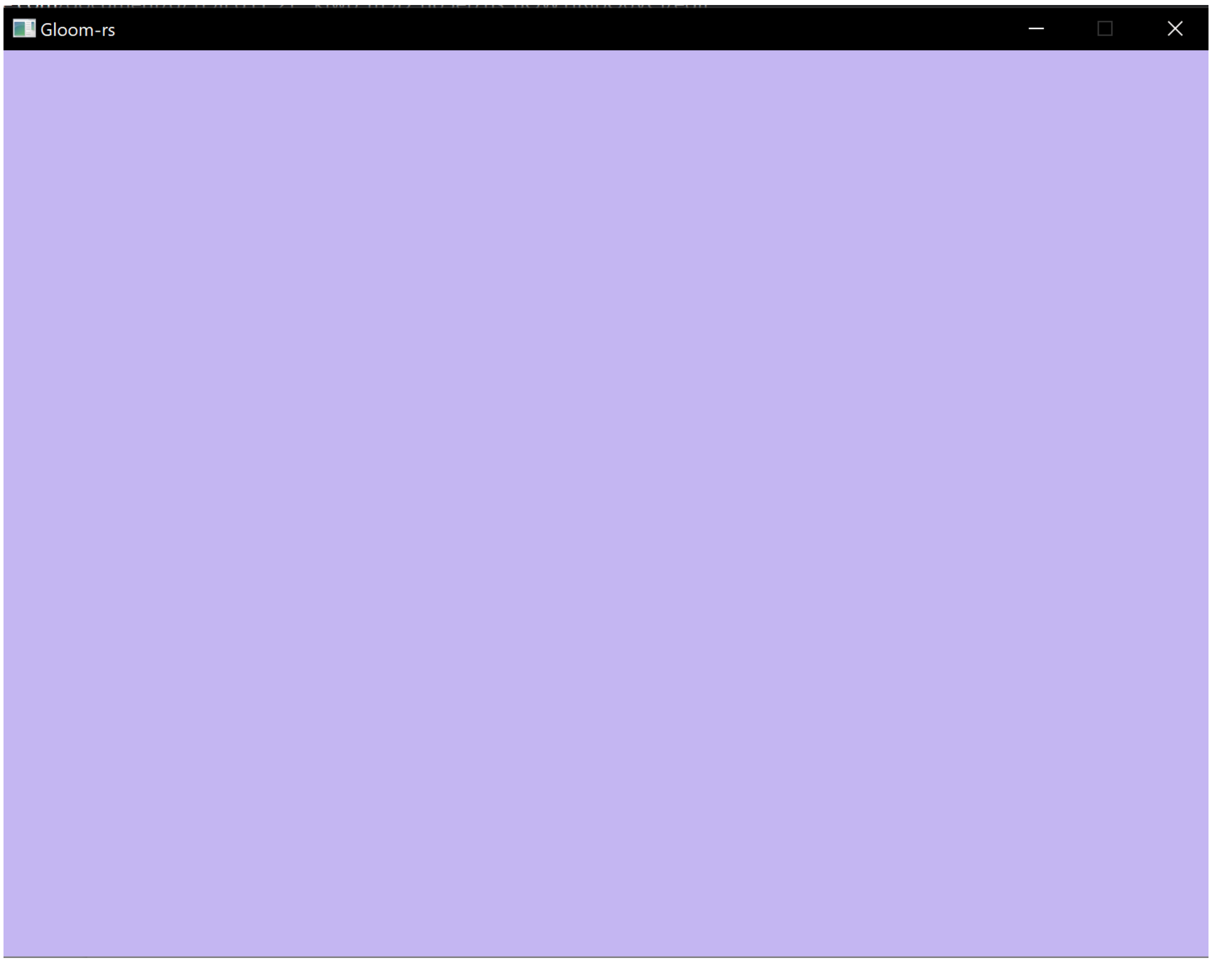
Oppgave 2)
i) This phenomenon is called clipping.

ii) Clipping occurs when you try to draw something on the screen and some of the values are out of the range for your display device. In this example, the z-coordinate for the first and third vertex is out of range for the displayed image, which in our handout code, has a 1x1x1 limit.

iii) The purpose of clipping the image to fit the displayed frame are several. The main purpose is so that the correct part of the figure is shown. This means projecting a 3D object to a 2D surface. The first part of the clipping is finding which surfaces to render to be able to give the correct image to the viewer. Another purpose is to relax the pixels which are not shown from the particular angle or in the specific frame. This will lead to less computation and less use of memory for the computer.



b)



i) We see that no triangle is rendered.

ii) This happens because we give openGL the indices in the wrong order, causing the rendering to fail.

iii) This  phenomenon is called backface culling and occurs when you switch the indices from, for instance [0,1,2] to [2,1,0]. By doing this we instruct the computer to render the triangle clockwise, which openGL interprets as the backside of the object, causing it not to render, whereas by rendering counter-clockwise, openGL interprets it as the face of the object, hence rendering it. The rule we can make from this example is to always give the input indices on the object we want to render in a counter-clockwise order. In this way, openGL will render all the surfaces that are visible to the viewer, and not the ones on the non-visible side, causing less computer computation and use of memory.

c)
i) We need to reset the depth buffer for each frame in case the viewer's perspective has changed or the objects have moved. Therefore the rendering will be able to adapt to which object is closer and should be rendered in front or if the viewer's angle has changed.

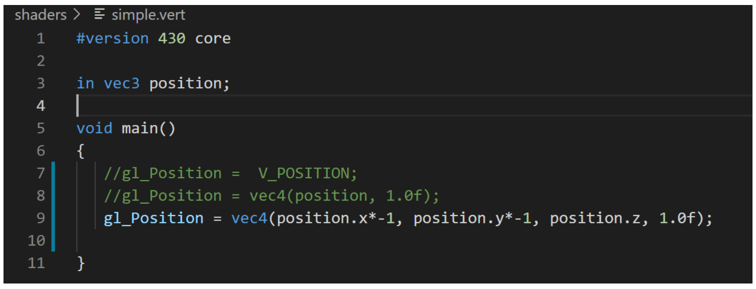
ii)The fragment shader can be executed several times in one pixel if more than one triangle shares the same pixel.

iii)The most common types of shaders are the vertex shader and the fragment shader. The vertex shader's responsibility is to process each individual vertex. It is fed data from the VAO with information about how to process the vertex. The purpose of the shader is to transform from a 3D position into the 2D-surface, and it has properties of position, colour and texture. An important notice with the vertex shader is that it is a 1 : 1 mapping from input to output of the vertices.
The second most common shader is the fragment shader. This shader comes along later in the graphics pipeline. This shader is within the rasterization step, where pixels between the vertices are calculated and coloured.

iv) The reason that we use index buffers is so that we don't have to create several vertices on the same position. Suppose you were to draw a square consisting of two triangles. By using an index buffer you can manage with just 4 vertices, instead of 6.

v) If we were in a situation where we wanted to use the index buffer on parameters that describes texture, for instance, we could send in another parameter for the *pointer* value. (Byte where texture coordinate starts, e.g. 12 in a common x,y,z,u,v coordinates.)

d)
i) I mirrored the scene by simply multiplying the x and y coordinates with -1 in the vertex shader, like this:

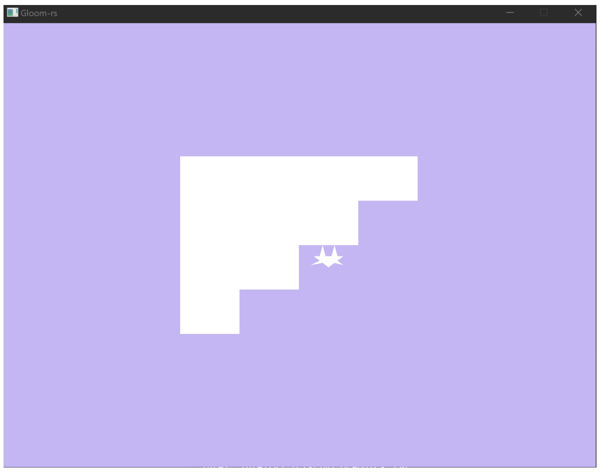

shaders > simple.vert

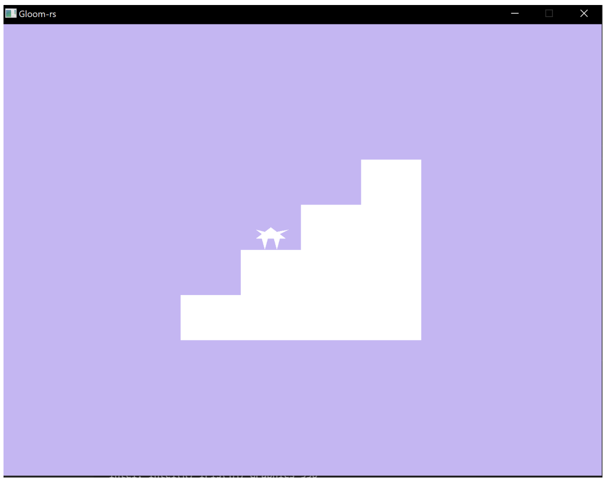
```
#version 430 core

in vec3 position;

void main()
{
    //gl_Position =  V_POSITION;
    //gl_Position = vec4(position, 1.0f);
    gl_Position = vec4(position.x*-1, position.y*-1, position.z, 1.0f);

}
```





ii) This was accomplished by modifying the values of the vector in the fragment shader, using normalised RGB decimal values.