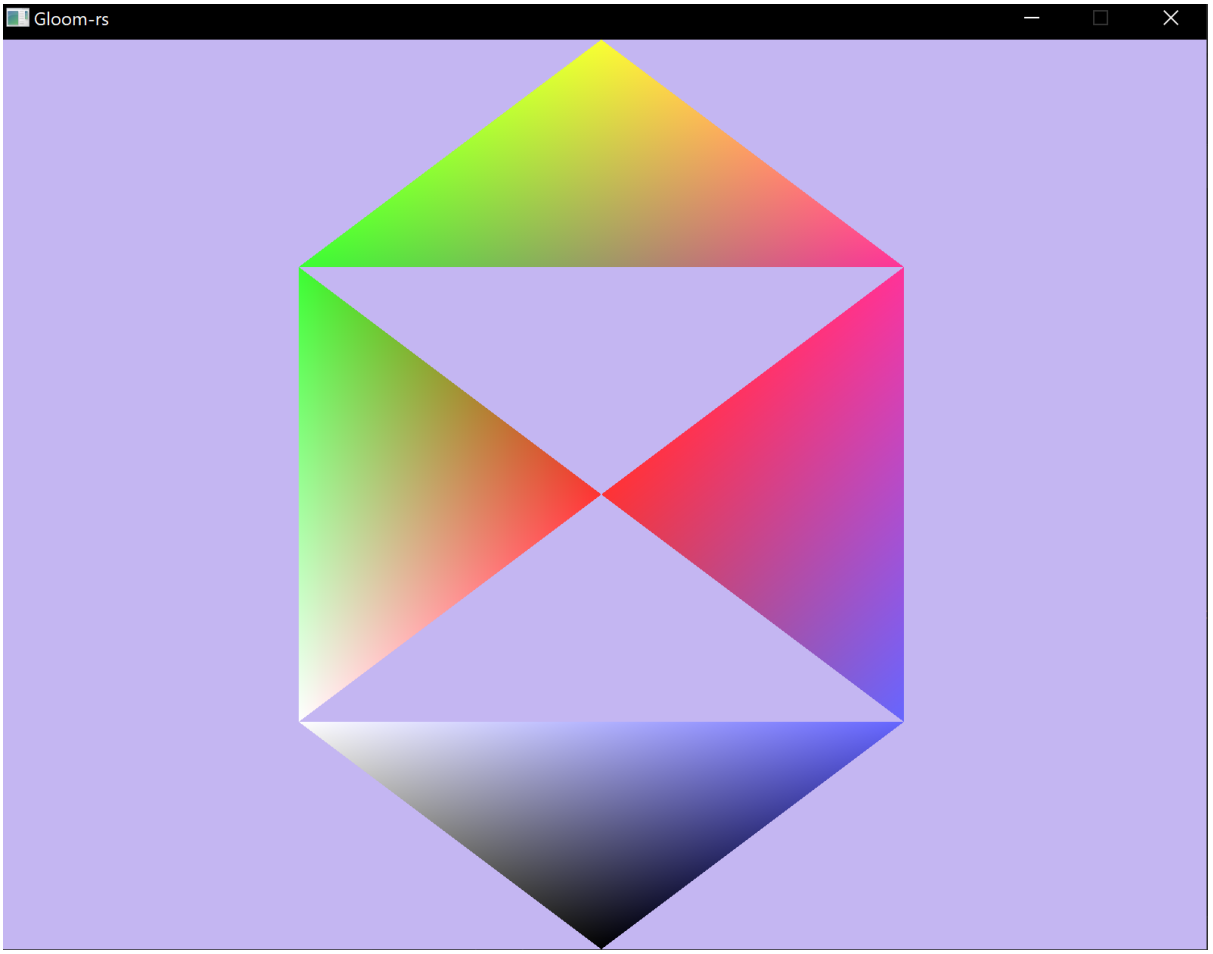
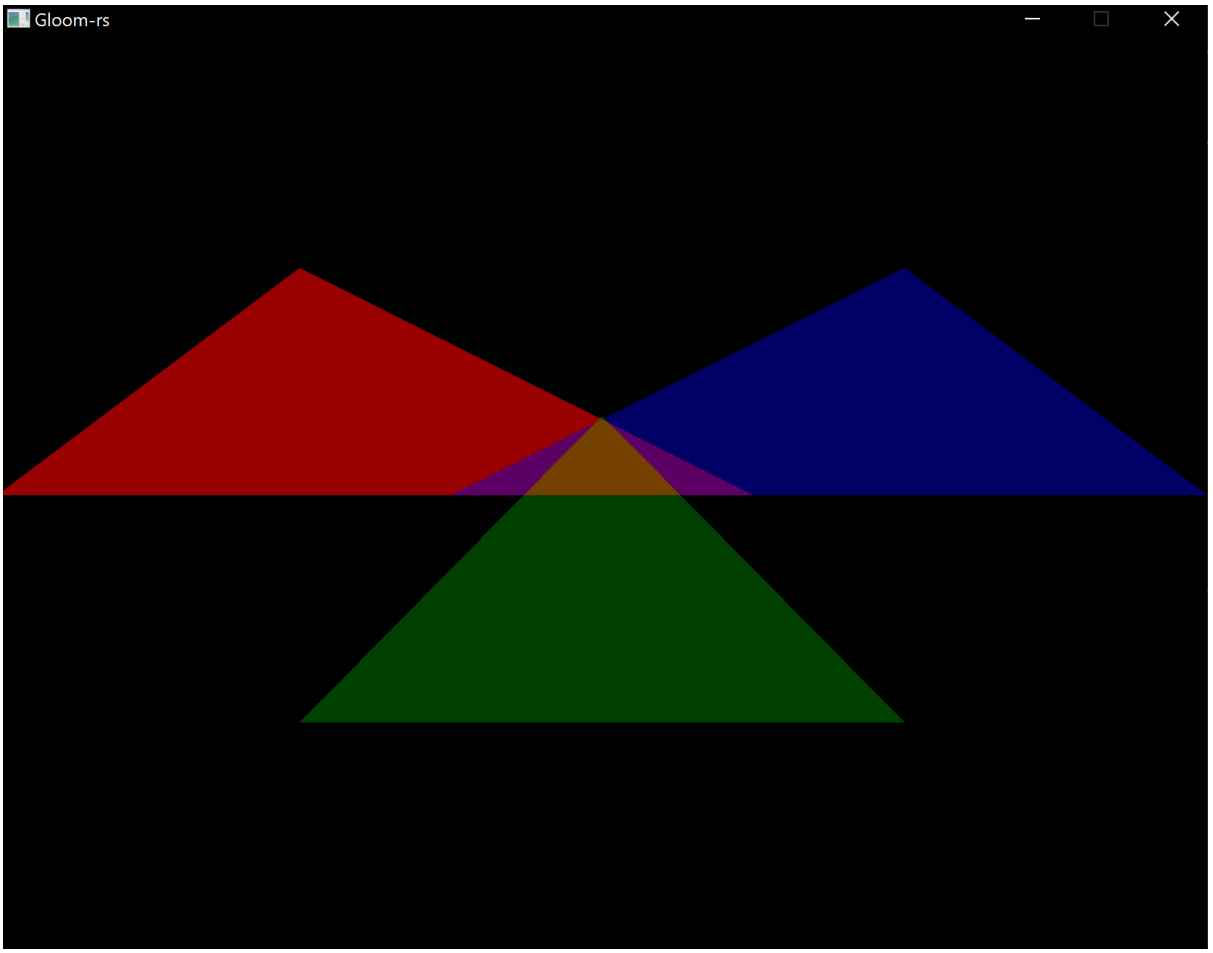
Øving 2

Even lauvsnes

1) b

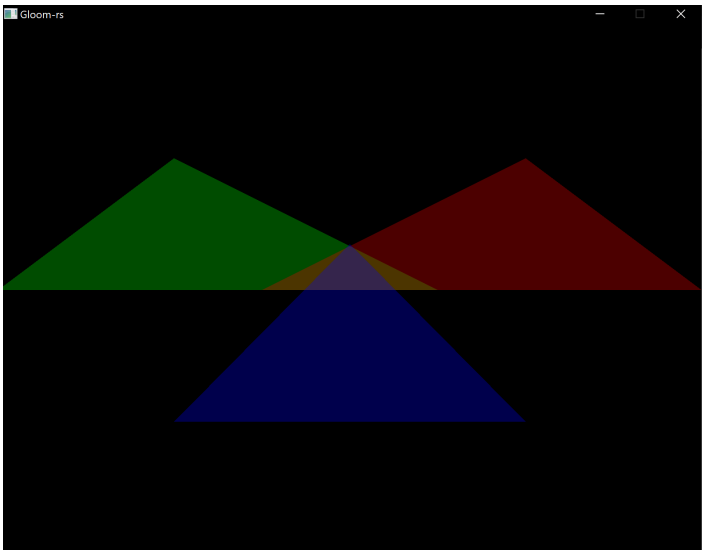


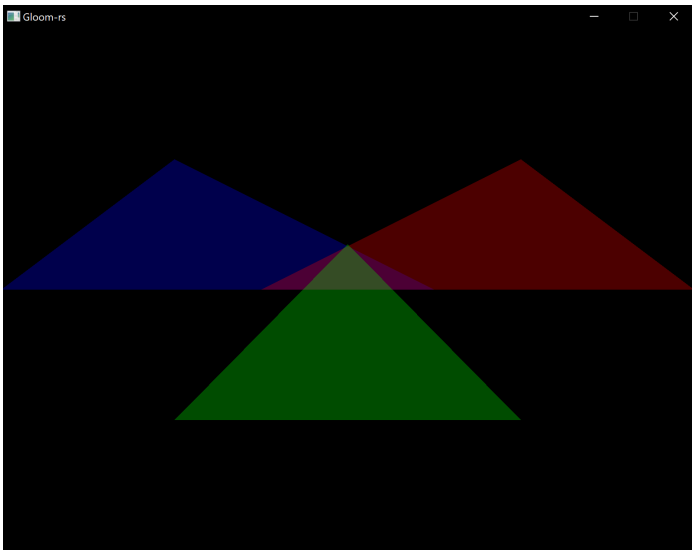
2 a) Changed the canvas color to make it a bit nicer to display.

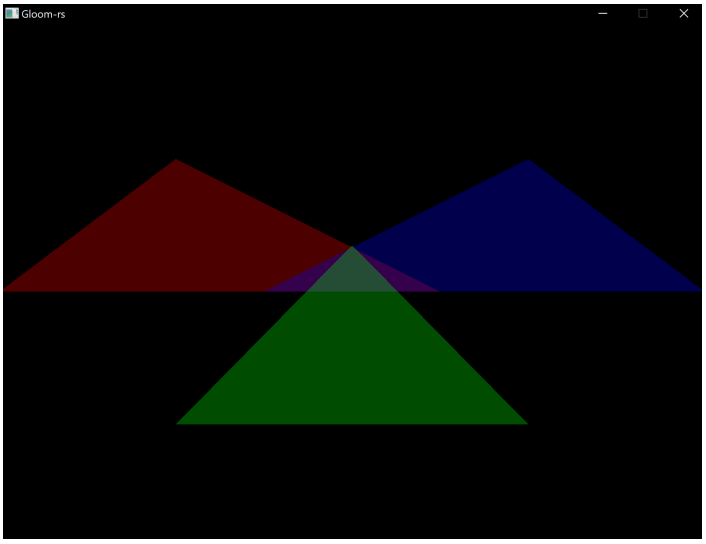


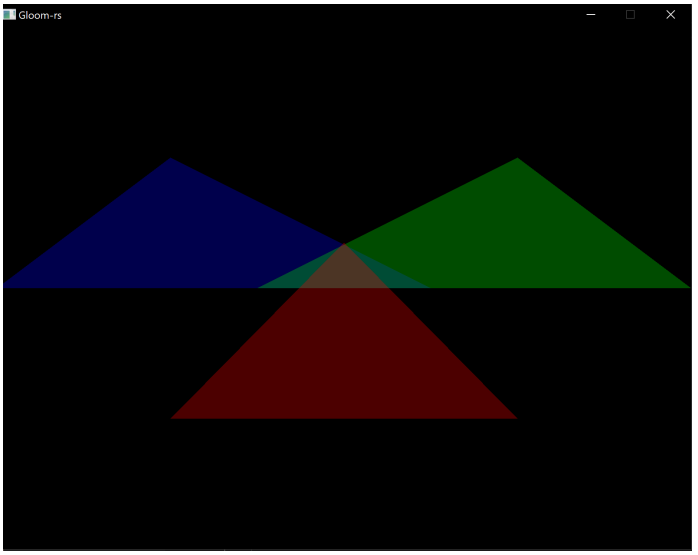
b)

i)









We observe that the triangle that is rendered on top, is the most dominant. This is caused by gl::BlendFunc(), which takes in aplha and aplha_minus_one as argument, and uses the formula:

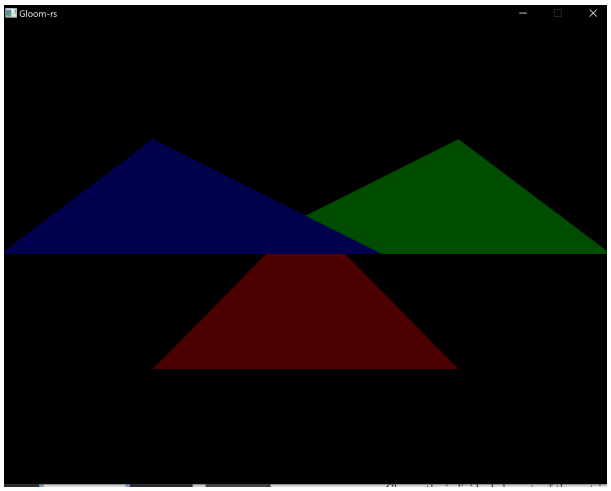
ColorNew = ColorSource · AlphaSource + ColorDestination · (1 − AlphaSource)

Let us look at the screenshot in the top right corner. The blue triangle is rendered first, then the red one on top. The color of the blue one will be(α = 0.3 for all colors):
blue * 0.3

Rendering red on top will then give us:
ColorNew = red*0.3 + blue*0.3*(1-0.3)

ColorNew = 0.3red + 0.21blue, which gives us a pink(ish) color. This screenshot is more dominated by red, conturary to the bottom left, where the rendering order is red-blue, giving us a more blue dominated color(purple).
This is the reason that the triangle which is rendered on top, has the most dominant color.

ii)



We observe that there is no blending in the colors. To figure this out, we can look at the equation from the blending function again.

ColorNew = ColorSource · AlphaSource + ColorDestination · (1 − AlphaSource)

In this particular example, the blue triangle is drawn first. Since there is no ColorDestination in the canvas when the triangle is drawn, ColorNew = ColorSource*AlphaSource. When the next triangle is drawn(green) the same happens. Since this triangle has a depthbuffer < depthbuffer_blue, the blending function does not work. This phenomena occurs again because there is no ColorDestionation, meaning the ColorNew = ColorSource * AlphaSource.

3)

b)

a: Will lead to scaling in the x-direction. The new coordinate: $V_x = X_{old} * Scale$

b: Will lead to shearing in the y-direction. The new coordinate: $V_x = X_{old} + Y_{old} * scale$

c: This will lead to translation in the x-direction. New coordinate: = $V_x = X_{old} + scale$

d: Shearing in the x-direction. The new coordinate: $V_y = Y_{old} + X_{old} * scale$

e: Scaling in the y - direction. The new coordinate: $V_y = Y_{old} * scale$

f: Translation in the x-direction. The new coordinate: $V_y = Y_{old} + Scale$

All of the transformations above are results of matrix multiplication and the result was taken from the following calculations, here with f) as an example.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \sin(y) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix} = \begin{pmatrix} v_x \\ \sin(y) + v_y \\ v_z \\ 1 \end{pmatrix}$$

Elapsed.sin() was used to visualize and is referred to as *Scale.*
The following code setup was used:

```
loop {
    let now = std::time::Instant::now();
    let elapsed = now.duration_since(first_frame_time).as_secs_f32();
    let delta_time = now.duration_since(last_frame_time).as_secs_f32();
    last_frame_time = now;

    let matrix = glm::Mat4::from([
        [1.0,0.0,0.0,0.0],
        [0.0,1.0,0.0,elapsed.sin()],
        [0.0,0.0,1.0,0.0],
        [0.0,0.0,0.0,1.0]]
    );
```

```
#version 430 core

layout (location = 0 ) in vec3 position;
layout (location = 2 ) in vec4 color;
layout ( location = 3) uniform mat4 transformation;

out VS_OUTPUT{
   vec4 color;
} OUT;


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

}
```

c)

We can be certain that none of the transformations were rotation because a rotation needs to transform along several axises at the same time. Since the task asked to only change one coordinate at the time, we can be certain that none of the transformations were rotations.