TDT4195 Assignment 2

Olof Ljunggren

September 22, 2023

1 Visual computing fundamentals TDT4195 Assignment 2

1.1 Task 1

Task 1: Per-Vertex Colors [0.5 points]

1.1.1 a: Extend function create_vao to enable texture specification

Done

1.1.2 b: Render a scene containing at least 3 different triangles, with diffrent vertex colours

For every fragment in a triangle OpenGL interpolates the colour value depending on the three verticies colours. This is done automatically by a mathematical formula which describes every point inside the triangle as a linear combination of the verticies.

1.2 Task 2

Alpha Blending and Depth [0.5 points]

1.2.1 a: Drawing triangles at diffrent depth.

1.2.2 b: Swap colours and depth of the triangles.

1. Swap colours, what happens with the blended colours?
    - What happens is that the last drawn triangle have larger effect on the colours. The blending is not all the same when we draw the same triangles in diffrent order. Effectively swapping the colours means swapping the order the colours are drawn.
2. When does it occur?
    - Similarly as for the colour swap the order of the z makes a diffrence. There is no blending effect when the triangle with lowest z (closest to the viewer) are drawn first. Then it will override the other colours since they dont fulfill the depth test. This is where the depth buffer is used.

1.3 Task 3

The Affine Transformation Matrix [0.7 points]

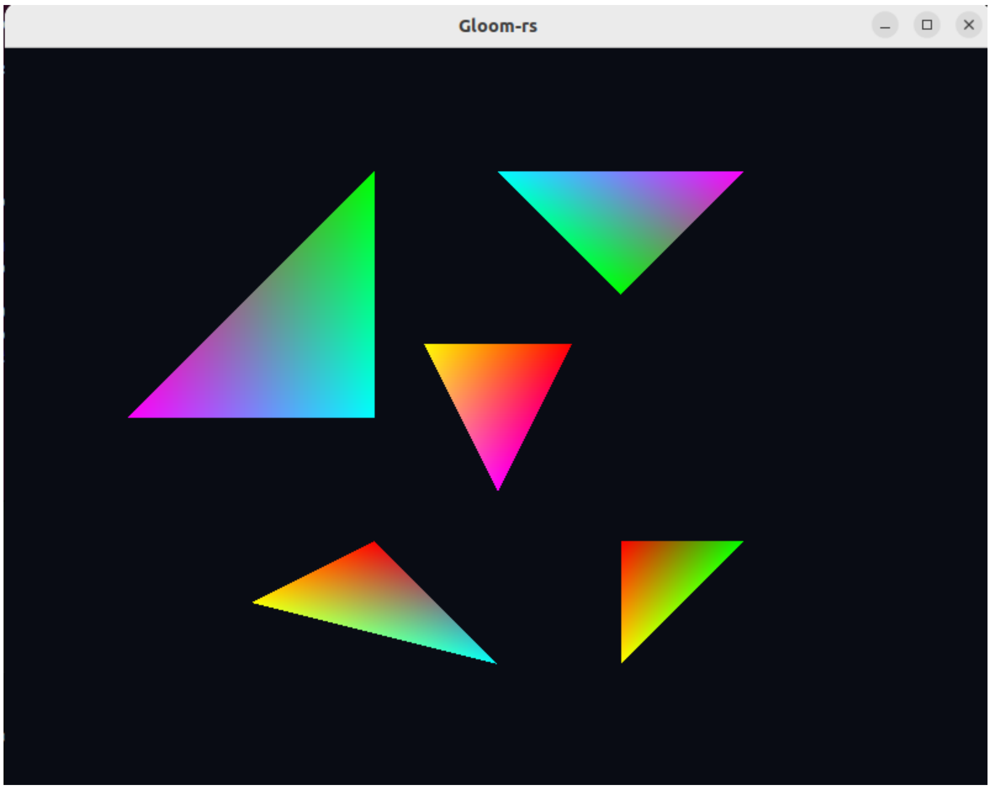

Figure 1: Five triangles with interpolated colours.

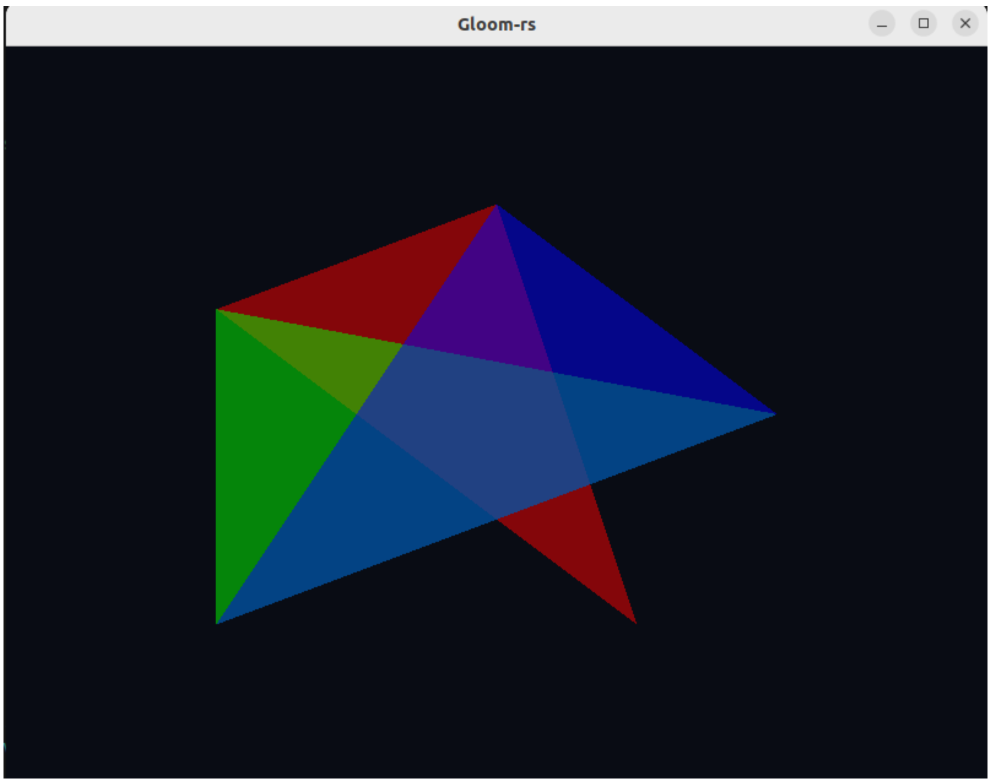

Figure 2: Swapped colours example 1.

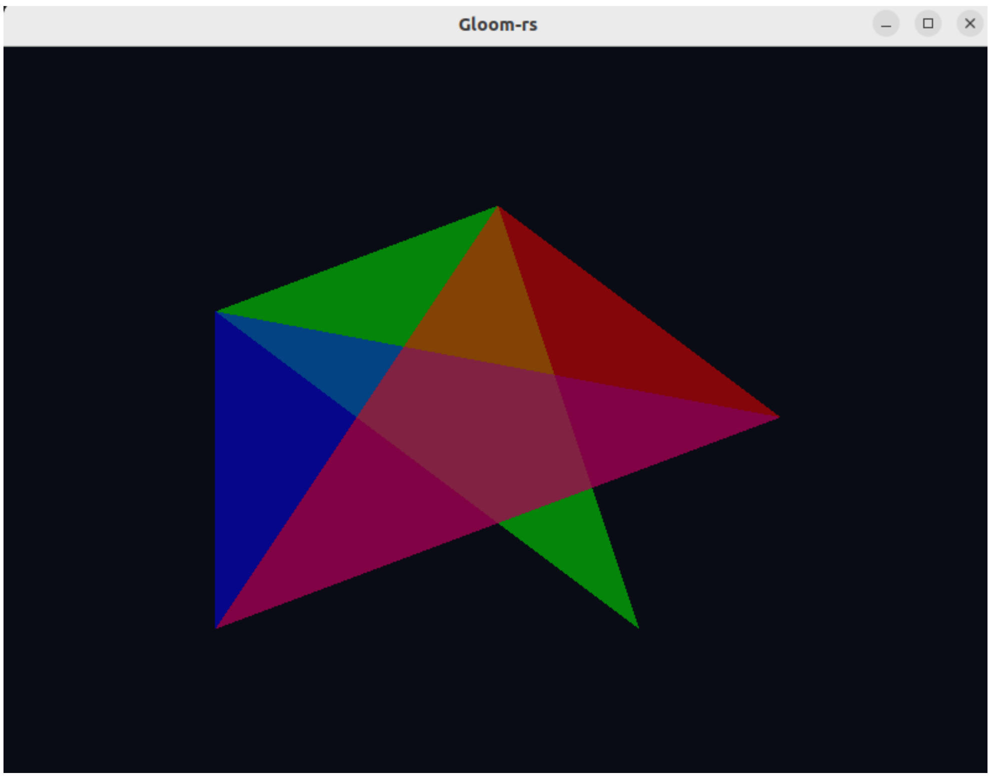

Figure 3: Swapped colours example 2.

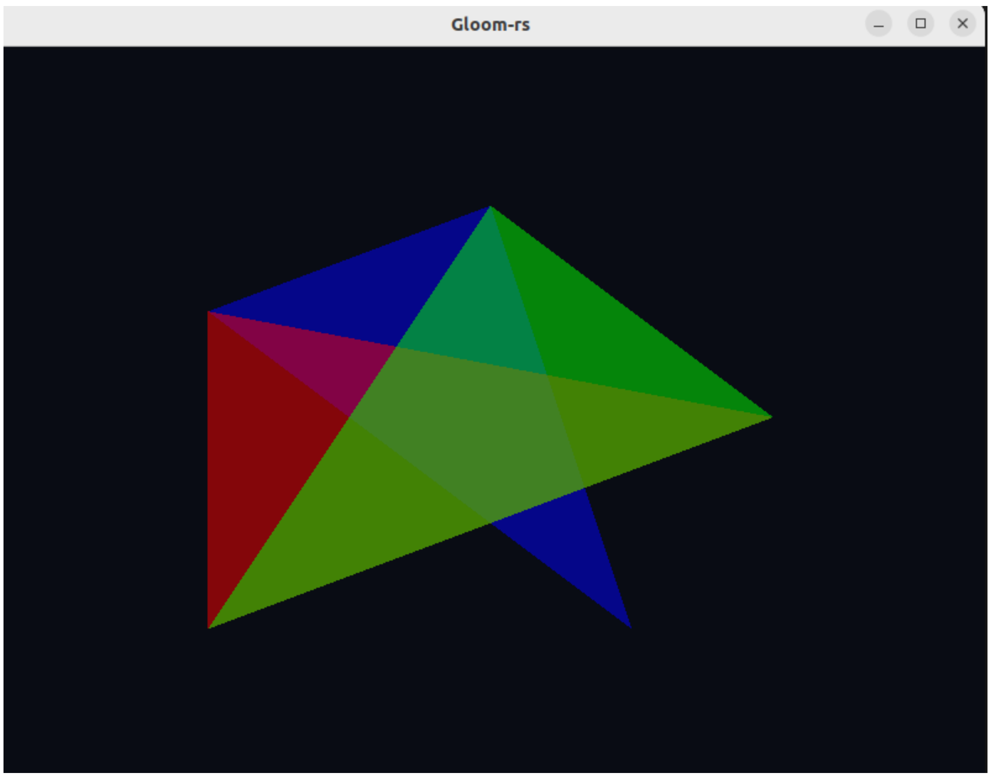

Figure 4: Swapped colours example 3.

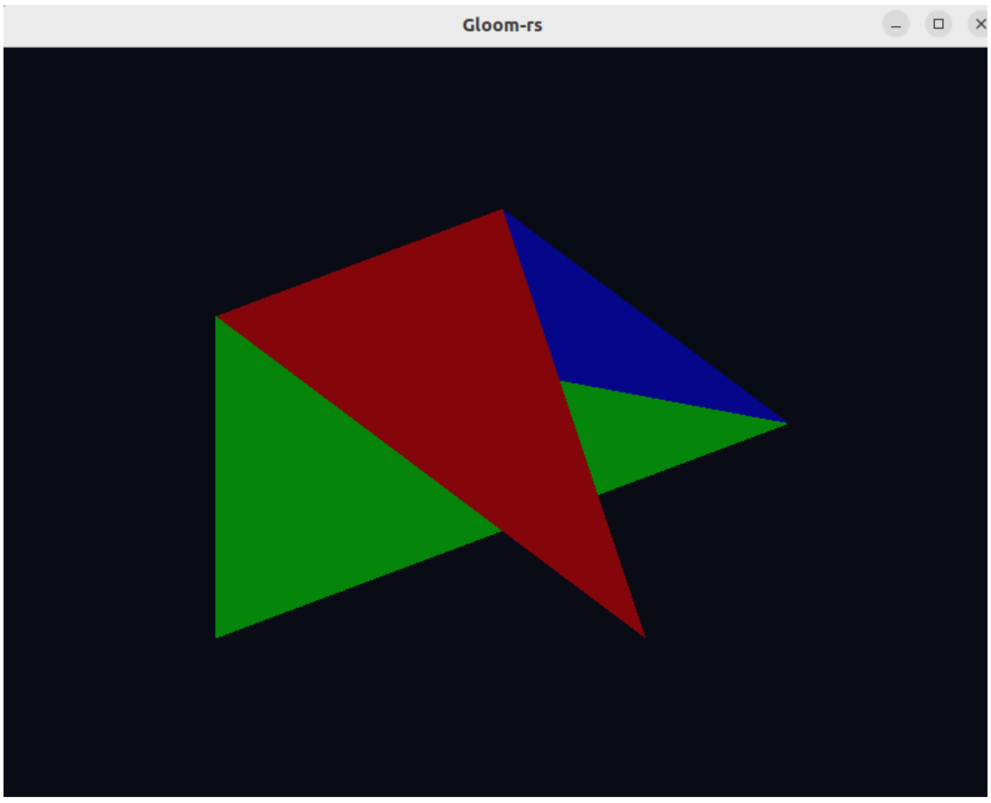

Figure 5: Swapped z example 1.

1.3.1 a: Modify the vertex shader so that the coordinates is being multiplied by a 4x4 matrix.

Done

1.3.2 b: Change matrix varaibles to describe their function in the transformation matrix.

In the matrix:

$$\begin{pmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The “a” corresponds to a scale for the x-axis. Every x-value gets “a” times larger. Similarly the “e” scales the y-axis. Negative values on these also mirror the axis. The “c” is the part translating the x-values in 3d homogenous coordinates. Correspnding value is “f” to translate the y coorinates. The last two ones, “b” and “d” are shearing effects. “b” is shearing the x-axis with respect to every y-value. “d” is shearing the y-axis with respect to every x-value.

1.3.3 c: Why can you be certain that none of the transformations observed were rotations?

This is because rotations needs at least two parameters to be changed. If we only change one parameter we can’t get that needed symmetri in the matrix.

1.4 Task 4

Combinations of Transformations [3.3 points]

1.4.1 a: Pass by matrix as an uniform to the vertex shader.

Done

1.4.2 b: Apply a perspective projection onto the scene.

Done. I also set the near clipping plane to 1 and the far to 100. Corresponds to -1 to -100 since the perspective projection is flipping the z-axis.

1.4.3 c: Implement a camera which can move x, y, z and rotate around x (pitch) and y (yaw).

1. I created 5 variables to store camera data. Position for x, y, z and the two rotations, yaw and pitch.
2. To implement a controller for movement and rotation I used the built in handler. I used wasd to move up, down, left and right. Space to move backwards and left shift to move forward. The arrow buttons were used for rotations, up and down corresponds to the pitch movement which I also capped to 90 degrees. I also used the delta_time variable.
3. For every new frame I created these needed matrices to construct the entire tranformation. I then multiplied in the following order: Perspective * Rotation * Translation * Scale * Identity.

1.5 Task 5

Optional Bonus Challenges [at most 0.3 points]

1.5.1 a: Make movement more intuitive.

This task was accomplished by multiplying some sinusoidal terms when updating the positions.

1.5.2 b: Testing different interpolation qualifiers.

We can see the first one appears more smooth. This is because it is perspective corrected and calculated by barycentric interpolation. This is done by dividing the triangle into three subtriangles and using normalization. Basically the perspective works because the colour of each vertex is normalized by the depth for the smooth qualifier.

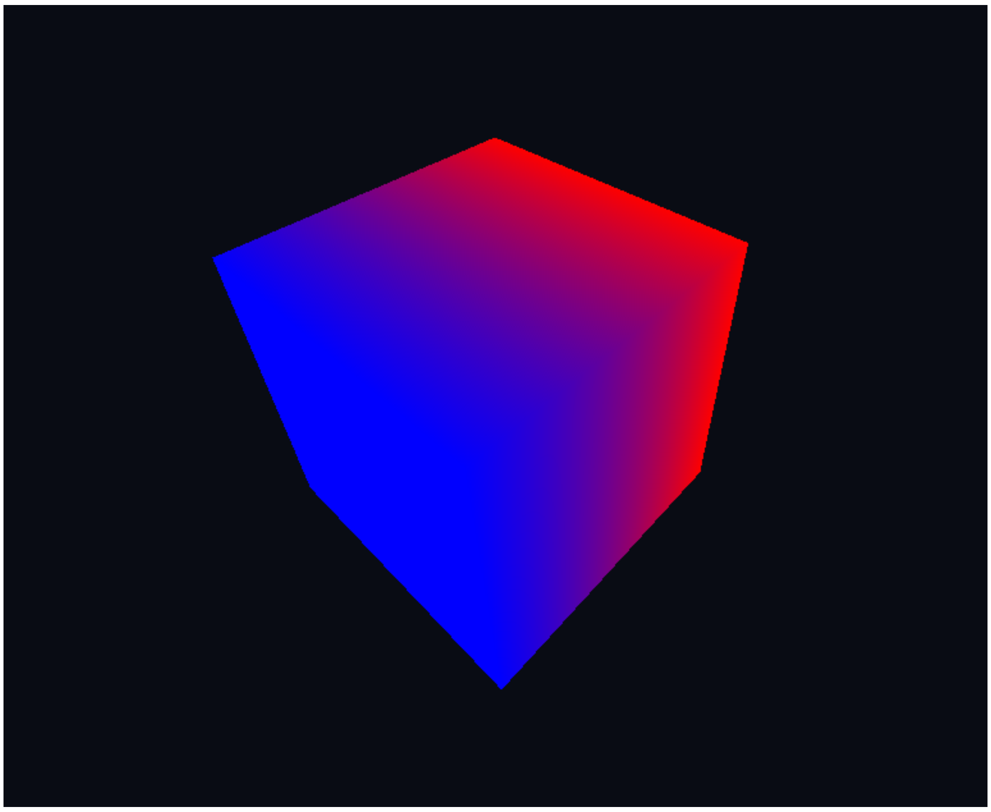

Figure 6: Smooth interpolation.

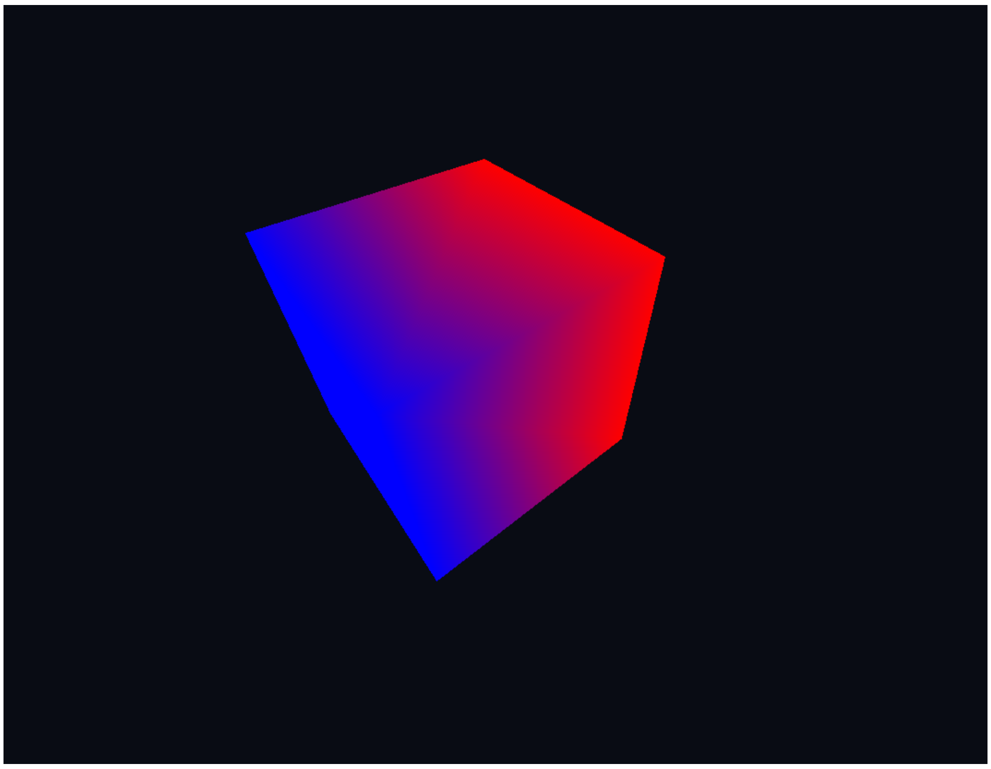

Figure 7: Noperspective interpolation.