Lab 3: Rasterization

Computer Graphics with Interaction DH2323

Author: Niklas Blomqvist, nblomqvi@kth.se
Lab partner: Annika Strålfors

May 26, 2014

Contents

1 Drawing points 1

2 Drawing edges 2

3 Filled triangles 3

4 Depth buffer 4

5 Illumination 5

6 Summary 7

In this lab me and my lab partner worked more individually than in the previous labs. We helped each other with problems and had an ongoing discussion about the lab but did most of the work separately.

1 Drawing points

The vertices of the triangles were first translated into points in 2D and then drawn in the canvas. I decided to wait with implementing the rotation as the lab suggested.

Figure 1: Points representing the vertices.

With a camera position at (0,0,-3.001) and a focal length, height and width of 500 we will be able to see the whole room. This can be calculated by using the methods of lab 2 but with this new angle of view and new distance between the camera and the room.

2 Drawing edges

An interpolation function was then implemented so that lines between the edges could be drawn. The camera movement and rotation was also added. I added so that the camera could both rotate around the y-axis (yaw) and the x-axis (pitch). To do this I just multiplied the two rotation matrices with each other.

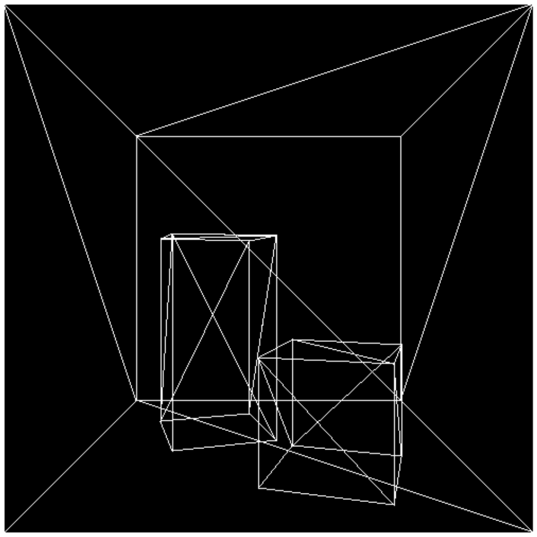

Figure 2: Drawing the edges between the points.

3 Filled triangles

This is where the lab starts to get a bit more complicated and I start encountering problems. The triangles are coloured by drawing lines for each row of the triangle. Two arrays keep track of the start and end position for each row in the triangle. It was hard to decide the indexing in the fourth step of the function 'ComputePolygonRows' but after visualizing an example on paper I could solve it. I also had problems with the interpolation function. I didn't get the same results as the test output in the lab assignment. After some time I noticed that it was because of C++ rounding off differently. By adding 0.5 to every step in the interpolation I finally got the same results (this was removed for the final submission). The speed in this implementation was far greater than that of lab 2.

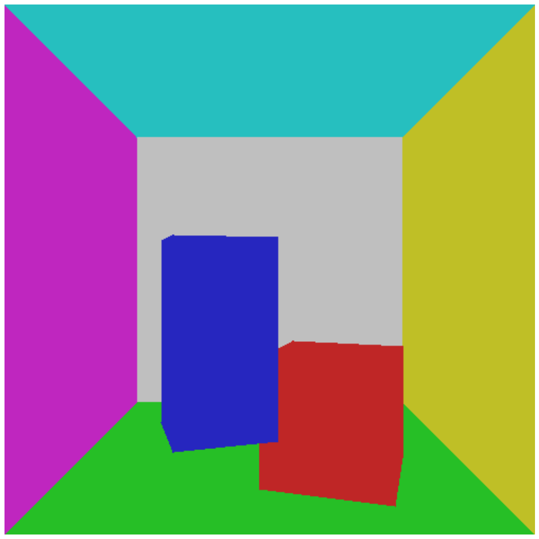

Figure 3: Filled triangles.

4 Depth buffer

A depth buffer solved the problem you can see in Figure 3, that the blue box appears in front of the red. It keeps track of the depth of the triangles, i.e. the distance from it to the camera so that we know which triangle should be drawn.

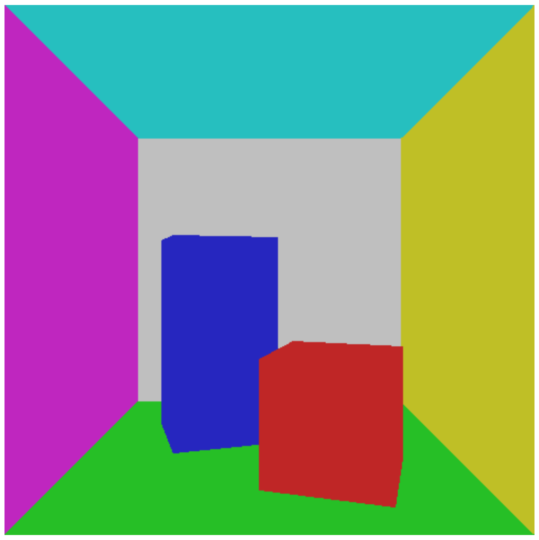

Figure 4: Filled triangles with a depth buffer.

Here, I noticed a problem with the camera movement, that if I moved the camera in some direction the program crashed and I got a segmentation error. The way I solved this was to add some conditions before checking the depth buffer. The indexing of the buffer is 500x500, but I called it with x and y values both lower than 0 and higher than 500. I believe this happens when any part of the triangles is outside of the canvas, and thus, shouldn't be drawn.

5 Illumination

The last part of the lab was about getting illumination. The first approach was to do the illumination computations for every vertex and then interpolate the result. The second approach was to do the computations for every pixel in the canvas. In order to do this a new struct for the vertices was created that would be able to hold its 3D position.

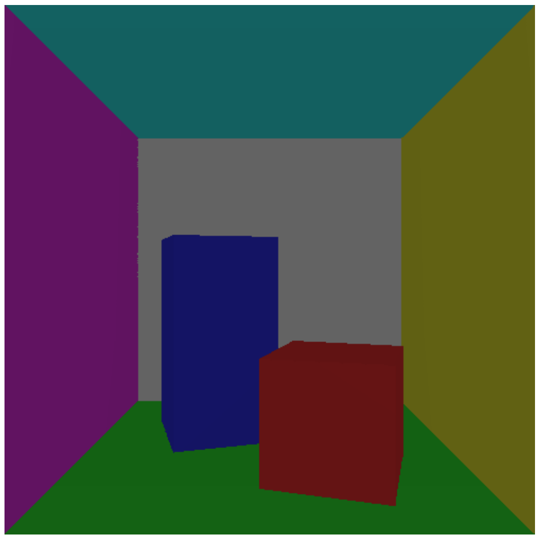

Figure 5: The first approach, illumination computed for vertices and then interpolated in between.

When interpolating the 3D position linearly, the illumination looks a bit bad.

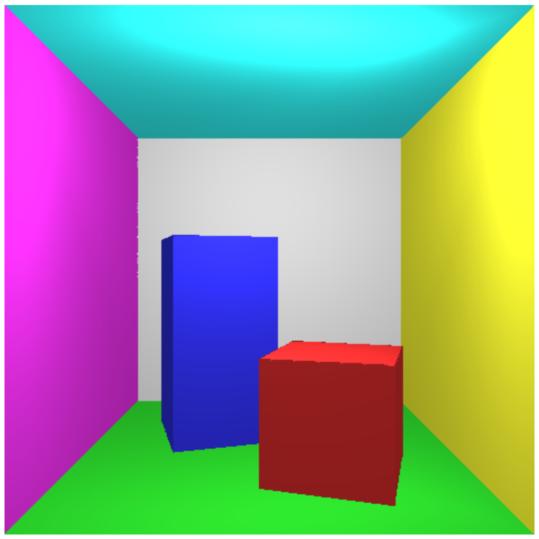

Figure 6: The second approach, illumination computed for every pixel, interpolating 3D positions linearly.

The 3D positions is then interpolated similarly to the depth.

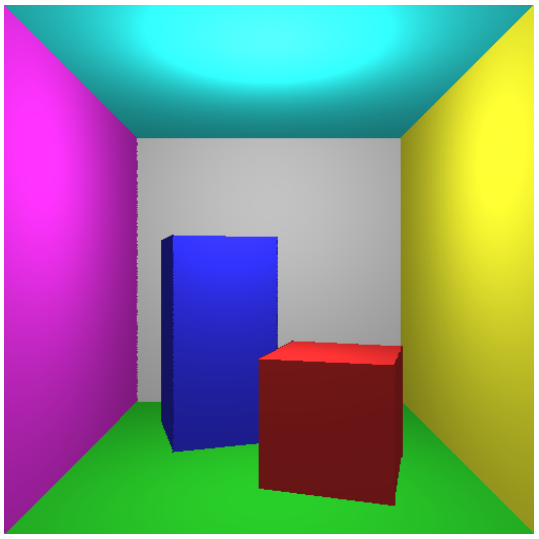

Figure 7: The second approach, using perspective correct interpolation for the 3D positions.

In this last part I noticed something wrong with my program. When I moved in some directions the position of the light decided to move to a specific location. I have no clue why this is.

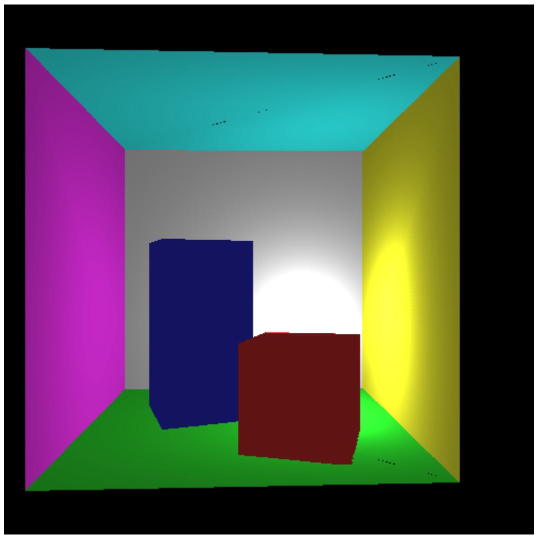

Figure 8: The light position placement after movement in a specific direction.

6 Summary

This lab was a bit messy at times. Mostly because a lot of code was just temporary and was to be altered or completely removed for the upcoming tasks. I also experienced a lot of different bugs that wasen't so easy to understand or solve, like the last one with the movement of the lights position. Overall the lab was fun and educative, but it's not fun when you don't understand why your program isn't working as you want it to.