DH2323 Lab 3

Ted Klein Bergman, tedber@kth.se
Sruti Bhattacharjee, srutib@kth.se
Elias Alkvist Cetin, ecetin@kth.se

Method

1. Transformations

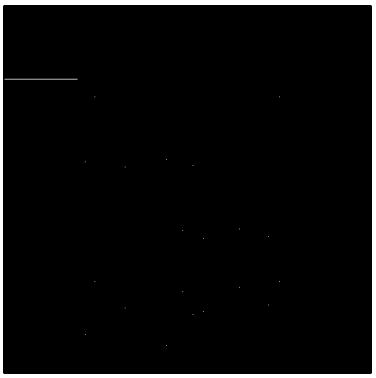
Due to having done lab 1 and 2, it was pretty straightforward what had to be done, since we already had done projection of points from 3D to 2D. We were also familiar with the translation part but we waited with implementing the rotation as the instructions said we could.

2. Drawing Points

The drawing of points wasn't too difficult but initially, we had issues with getting the points outside of our screen window which caused the program to crash. We later fixed this by clamping the position of the received projected vector in VertexShader to keep them inside of the screen. We could have just not rendered those points, but either way, the result would have looked wrong without any clipping.

3. Drawing Edges

We took the given interpolation function and at first, we wanted to just see how the output would look when rendering a line on the screen. The picture below shows the straight-line we rendered. After, we continued with implementing our given triangles and got an incorrect result shown under error and challenges. The scene was rendered and at a first sight it looked correct, but the scene was seen from behind, we fixed this and the fixed version is seen under final results.

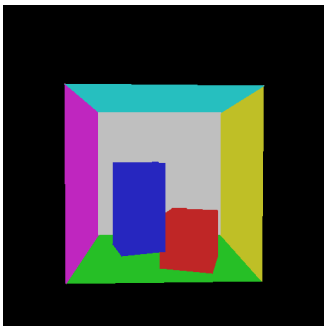
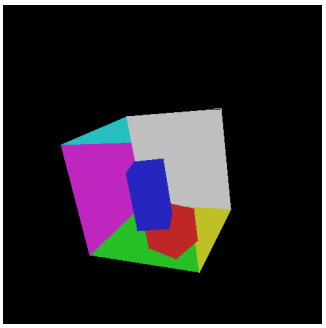
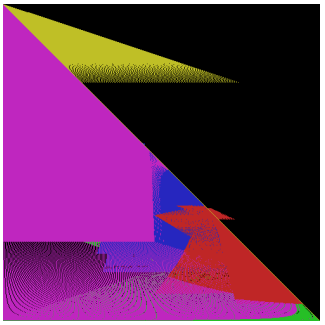


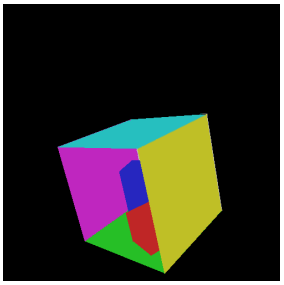
4. Filled Triangles

This took us some time to figure out exactly how to get the compute polygon rows function to work properly. The difficult part was the fourth step in the instructions, namely to loop through all edges and update the positions of the x-coordinates.

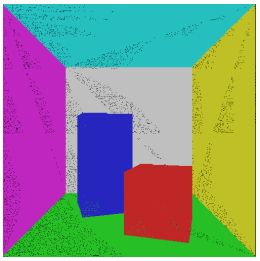
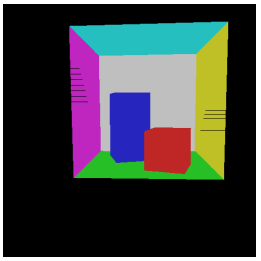
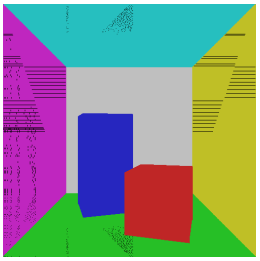
5. Depth Buffer

Since the filled triangles task gave us the scene where the blue rectangle was in front of the red, we had to implement a depth buffer. This was done to get it correctly and the final result is seen in the last picture below. We had some minor issues seen in picture 3, where we don't know what happened.




6. Illumination

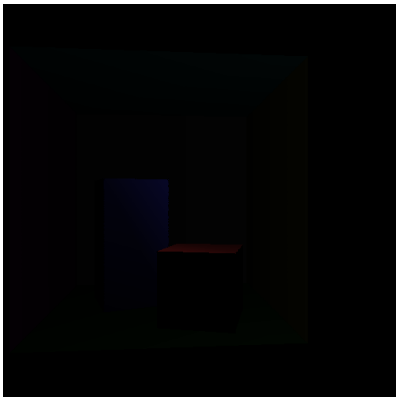
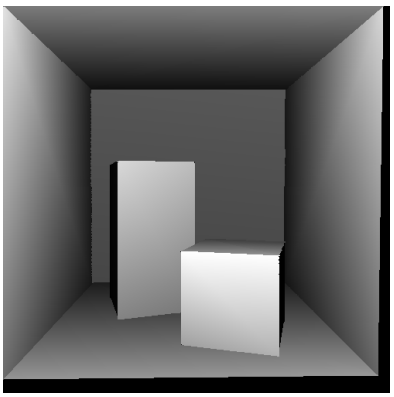
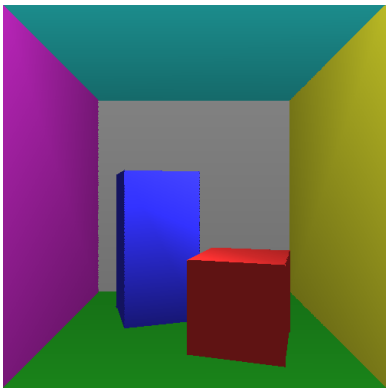
We had a problem with the interpolation when we added the additional fields in the Pixel struct. Since we no longer could use our first interpolation method, which was:

```
float step = delta / float(n-1);
float current = start;
for (int i = 0; i < n; ++i) {
   result.push_back(current);
   current += step;
}
```

We opted for more traditional interpolation algorithms. However, none of these worked, as they all once in a while skipped a pixel due to floating-point precision errors. We wrote some tests which confirmed this. These were the one's we tried:

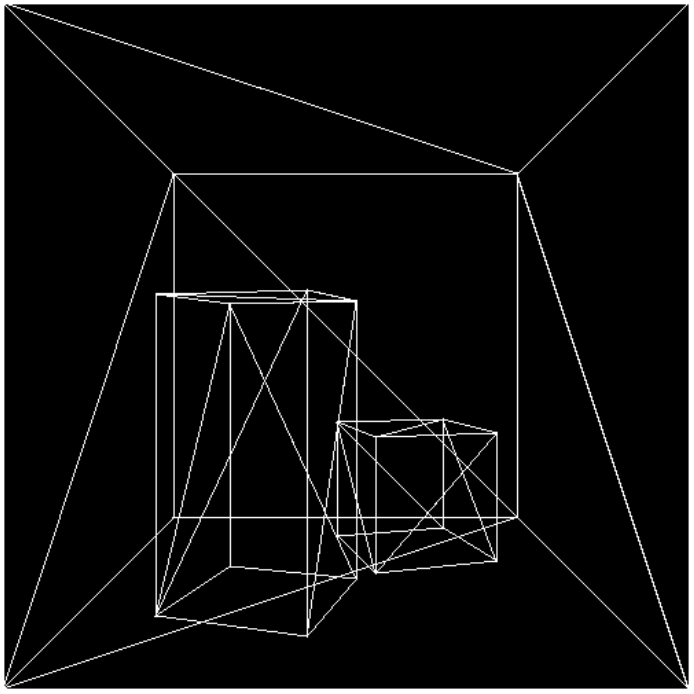
```
float lerp1(float a, float b, float t) {
   return (1.0f - t) * a  +  t * b;
}
float lerp2(float a, float b, float t)
{
   return a + t * (b - a);
}
float lerp3(float a, float b, float t)
{
   return b - (b - a) * (1.0f - t);
}
```

In the end, we decided to go back to a variant of our first technique.

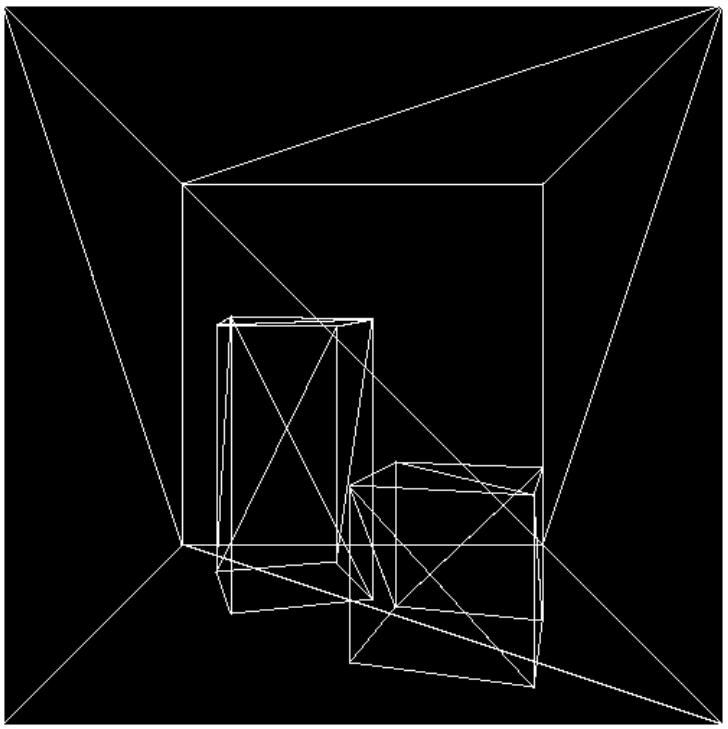
After the interpolation was working again, we had the problem of the light being too dark, almost black. This was an issue throughout the lab so we just increased the power of the light source.

Errors and Challenges

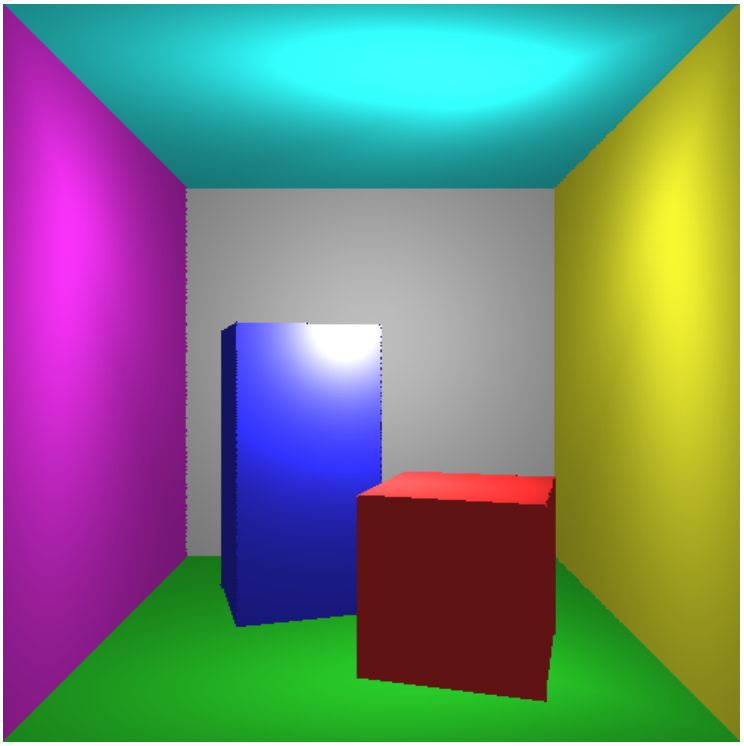


We received this picture after task 3, Drawing edges. It is slightly incorrect since we are seeing the picture as if we are behind it. This was fixed by changing the focal length to be negative and also the camera position's z-value to be positive.

Final Results



Correct result after task 3, Drawing edges.



Final result with illumination

Source code

https://github.com/Naxaes/DH2323-Computer-Graphics