DH2323 Lab 3 Report

Yan Yu
yan8@kth.se

Summary:

This rendering lab is about rasterization implementation in SDL. I will discuss my processes and the difficulties I faced in this lab assignment.

1. Vertices and Edges

Firstly, I wrote the VertexShader() function that translates 3D positions into 2D coordinates on the screen, according to the following equations:

$$x = f\,X/Z+W/2$$

$$y = f\,Y/Z+H/2$$

Image 1.1

The image above shows where the vertices of the triangles in the scene are displayed on the 2D screen.

Then I added the DrawLineSDL() and DrawPolygonEdges() functions using Interpolate() and VertexShader(). The image below is the output that draws all the edges on the triangles in the scene.

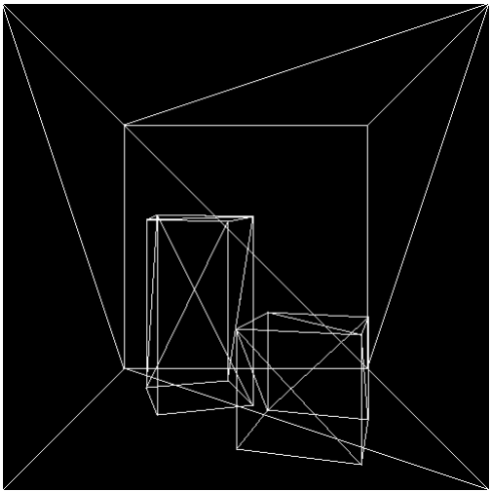

Image 1.2

For camera rotation, I update the yaw and pitch values once control keys are pressed, and the rotation matrices for y- and x-rotations are updated accordingly. Then in VertexShader() I calculate the new camera position as

```
vec3   newP = (v - cameraPos) * Rx * Ry;
```

2. Filling the Triangles

To draw all the pixels inside a triangle instead of just the edges, the program needs the ComputePolygonRows() function, which is the most complicated function in this lab. In the 4th part of the function that finds the x-coordinates in each row of the polygon, I loop through the polygon rows and find the corresponding y-index on the leftPixels and rightPixels arrays. Then I update the x-coordinates on leftPixels and rightPixels. But using this method sometimes gives me a segmentation error when I run the code. I had to loop through the rows in leftPixels and rightPixels to check the conditions for the segmentation error to disappear. This was a strange error about which I'm a bit confused, because I don't see much different in these two methods. Below is the image output for this part of the code.

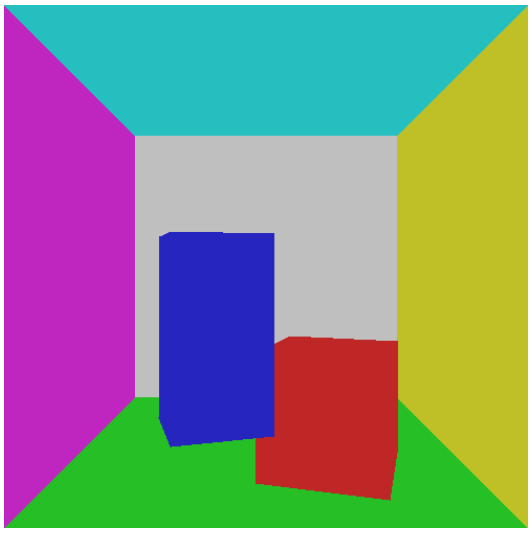

Image 2.1

The depth-buffer part was also challenging because I had to modify existing functions instead of writing functions from scratch, so it is really easy for the code to get messy and miss some statements that update the new variables. Below are some of the failed attempts due to some missing codes regarding the Pixel struct.

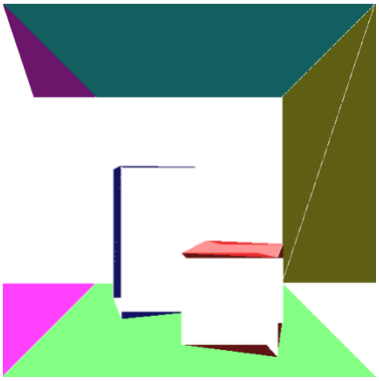

Image 2.2

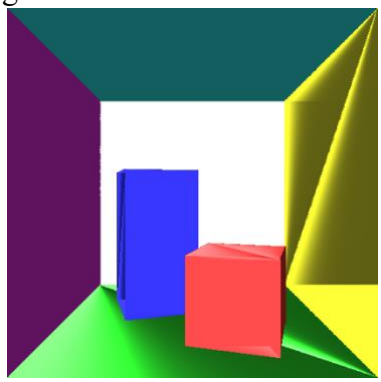

Image2.3

3. Illumination

The lab introduces two illumination methods, per vertex illumination and per pixel illumination. I implemented both, the results of which can be seen below. Initially, the default light power was set too low so that the room was too dark. I spent some time trying figure this out. After I increased the light power to 14.1f*vec3(1, 1, 1), the brightness of the room became more natural.

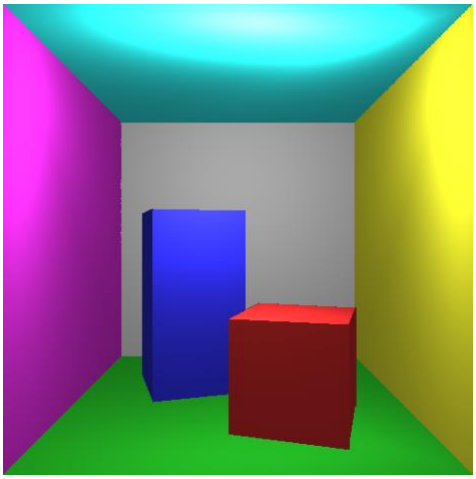

Image 3.1: per vertex illumination

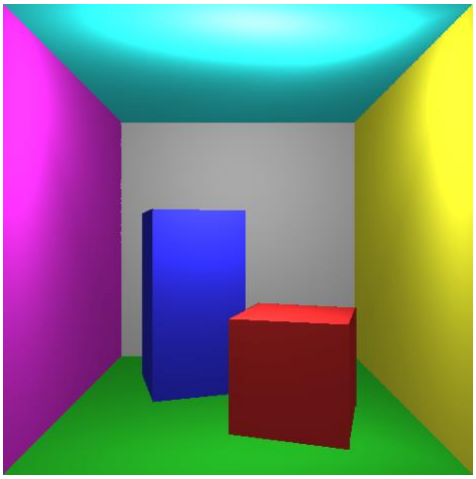

Image 3.2: per pixel illumination

The default outputs of the two methods are very similar and hard to distinguish. But the per vertex method (Image 3.1) looks quite bad when I'm moving the light source around. In comparison, the per pixel render (Image 3.2) responds to the moving of light much more smoothly.

An issue that I have with the render is that I can only move the camera outward, not toward the room. And if I rotate the camera and the edges of the room goes over the screen size, the program will give me a bus error. Therefore I couldn't freely move or rotate the camera inside the room like in the "draw edges" render. I could only obtain an outward movement of the camera similar to the image below. This is a bug that I'm not entirely sure how to fix.

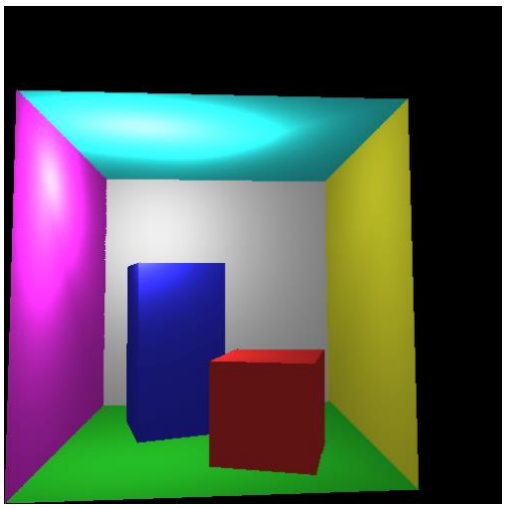

Image 3.3

Overall, this lab was quite challenging and time-consuming to solve. Towards the final part, I had to write down the flow of the rasterizer to better visualize and think through how the different functions and variables connect to each other. There are some minor glitches but the main functions are working by the end. It was a worthwhile attempt that helped me look closer into the rasterization pipeline.