DH2323 Lab 2

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

Method

1. Representing surfaces

This part was pretty simple and straightforward where we essentially just needed to understand the model representations of surfaces as triangles and then copy + paste the given code into our file.

2. Intersection of ray and triangle

Since we, in this lab, use triangles as the representation of surfaces and we wanted to implement ray tracing it was obvious that we had to implement a function that finds intersections between rays and triangles under some given conditions. We learned about the process of finding the closest intersection and made it work in our program. Later we stored some data from the found intersection in a struct.

3. Raytracing

We followed the instructions but our approach was a bit different. The variables focal length and camera position was used in the struct Camera instead of as supposed being global. Then we looped through each pixel in the image plane and cast rays to find intersections and colored them with the color of the surface triangle's color if the intersections were found, otherwise black. This gave us some black lines at first that we later corrected. Pictures of this can be seen in later paragraphs.

4. Moving the Camera

This took us the most time to first understand the theory behind the implementation of this in code, especially with the rotation matrices. We created different templates for rotation matrices in each direction and then a resulting rotation matrix consisting of these initial rotation matrices. It took some time until we got the camera's local coordinate system correct (right-hand rule with z pointing out from the screen).

The rotation was quite straightforward to implement. We had a bug where one of the matrices had a copy-paste error (the values were in the wrong column) which gave us some weird behavior, but it was quickly fixed when confirming the formula from Wikipedia.

5. Illumination

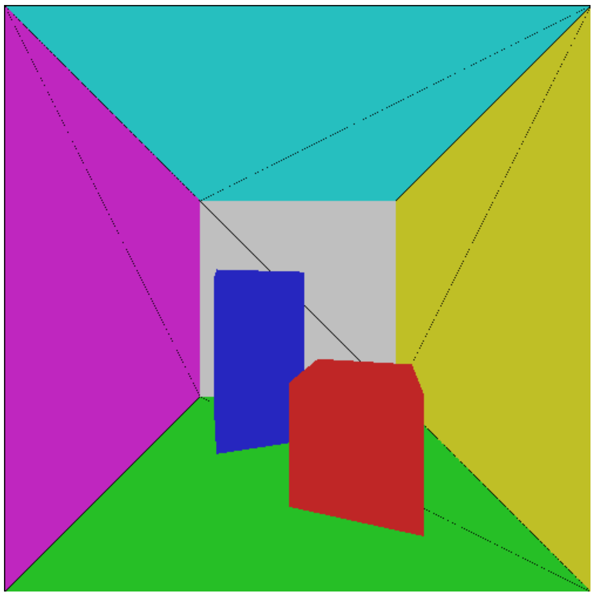
Light source etc.

The initial light (given by DirectLight) was quite quick to implement by just copying the formulas given into a function and using the result to draw our color, instead of using just the triangle's color.

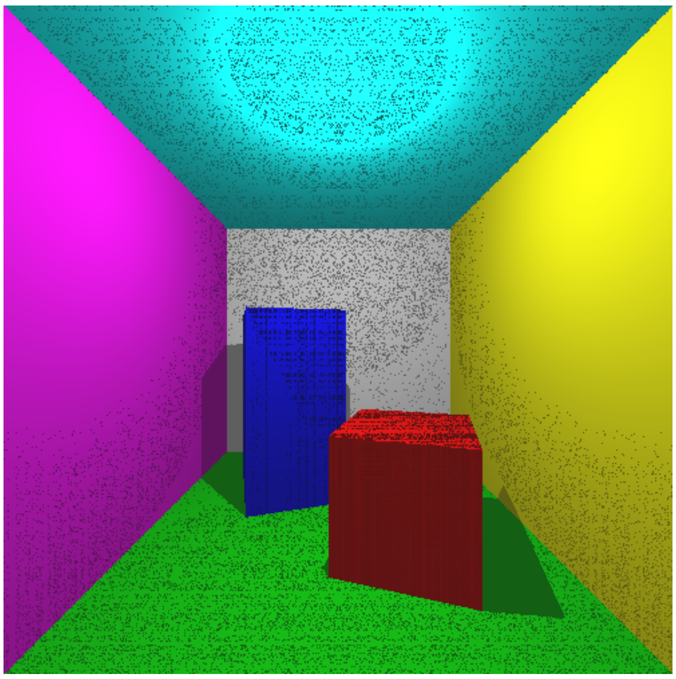
We had some difficulties in getting the correct shadows at first, more can be read under errors and challenges. For indirect illumination we just followed the instructions and formulas.

Errors and Challenges

We had minor issues with some black lines that were solved quickly by allowing t_ray, u, and v to be exactly 0, as well as u + v to be exactly 1.

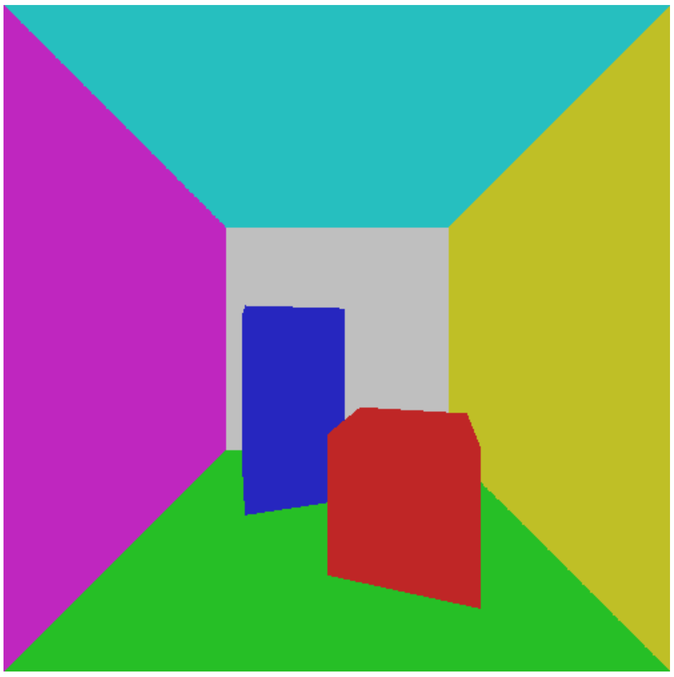


When working on task 5.1 regarding direct shadows we had some initial problems in implementing the shadows due to the if statement checking whether there was a shadow at a specific point becoming true more often than demanded. The result is shown in the picture below. This issue probably occurred due to the fact that we might have had rays self-intersecting caused by small numerical errors. This is called shadow acne and to fix this we took the shadow distance value and added a small number and got the final result shown in the last paragraph.

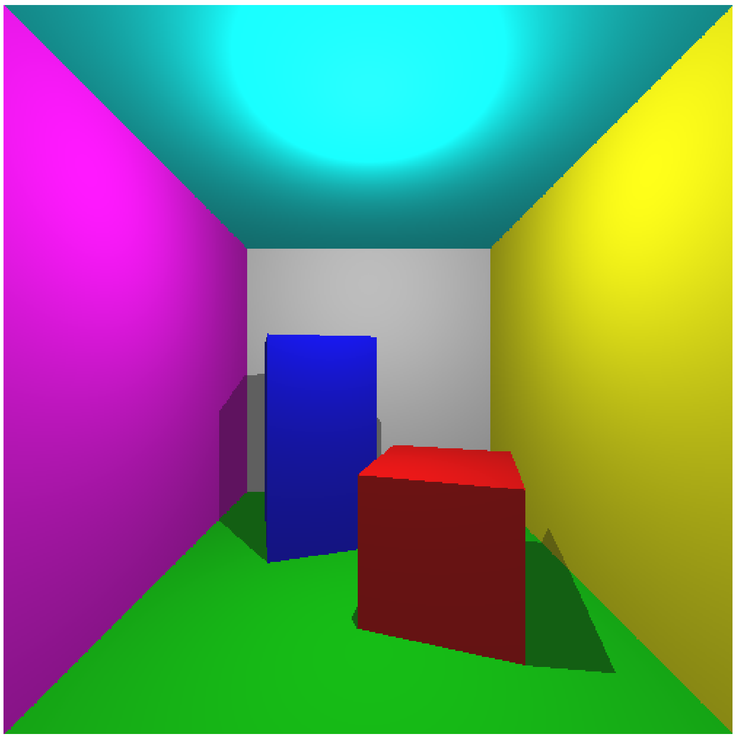


Final Results

After task 3. ray tracing



After the final task the end result became this.



Source code

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