DH2323 Lab 2 Report

Yan Yu
yan8@kth.se

Summary:

This rendering lab is about ray-tracing implementation in SDL. I will discuss my processes and some challenges in solving three of the main problems in this lab assignment, including the closest intersection function, camera position and shading.

1. Closest Intersection

First I had to write the function that detects the closest intersection of a ray on a set of triangles. I followed the method in the lab instructions, which represents a point in triangle as a vector plus ratios of its two edges:

$$\mathbf{v_0 + ue_1 + ve_2 = s + td}$$

The variables u, v, and t can be derived using the inverse matrix of A. In the closestIntersection() function, I first set closestIntersection.distance as infinitely large, and for every triangle check if u, v, t are within ranges for an intersection to occur, and if t (the distance from starting point to intersection) is smaller than the current closestIntersection.distance, update the intersection.

```
if (x.x>=0.0001 && x.y>=0 && x.z>=0 && (x.y + x.z)<=1){

        if_intersect = true;

        if (x.x < closestIntersection.distance){

           closestIntersection.position = start + x.x * dir;

           closestIntersection.distance = x.x;

           closestIntersection.triangleIndex = i;

        }

     }

  }
```

```
return if_intersect;
```

One thing I noticed is that if I set t > 0 (x.x in the code), there are some problems with the shadows, as shown in *Image 1.1* below. I spent a lot of time trying to figure this out, and the shadows only work properly after I set something like t >= 0.0001, which produces the proper shadow effect in *Image 1.2*. It could be some interference with the camera but I'm not sure what the exact cause of this problem is.

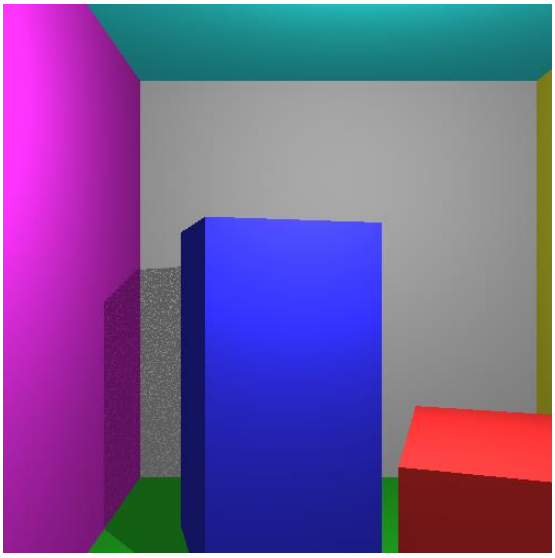

Image 1.1

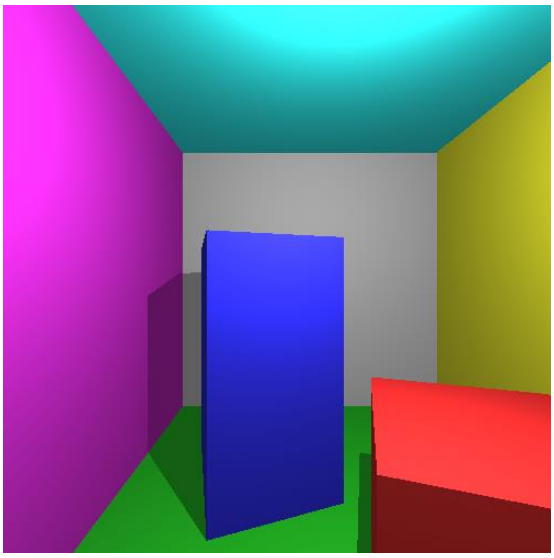

Image 1.2

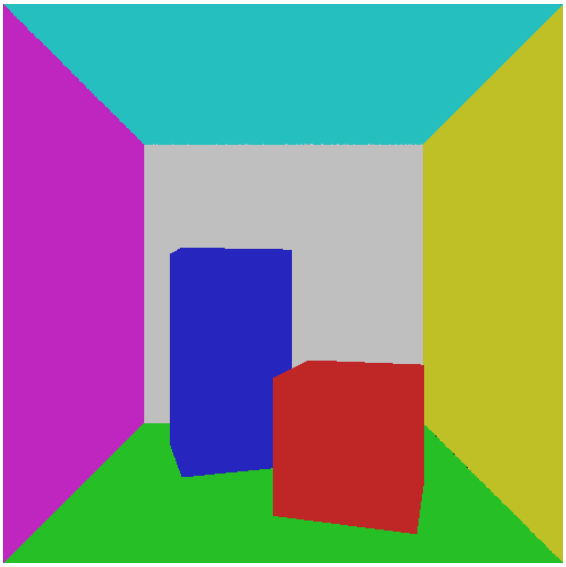

Image 1.3

Image 1.3 above is the rendering result of the triangles without shading.

2. Camera Position

I set the focal length to be the same as screen height. Because the x, y, z dimensions of the room are from -1 to 1, the camera needs to be placed at a distance behind z = -1 to be able to see the whole room. The field of view depends on the ratio of the dimension of the room and focal length:

$$\text{field of view} = 2 * \arctan(d/2f)$$

The camera's distance to the screen (z = -1 plane) depends on the focal length's ratio to screen height:

$$d' = f / (\text{height}/2)$$

For f = screen_height, d' = 2. So the camera position should be (0, 0, -3) for the scene to be displayed on the whole screen.

Then I wrote the Update() functions that updates the camera position according to keyboard input.

```
If (keystate[SDL_SCANCODE_LEFT])

    {

        // Move camera to the left

        yaw = -amount2;

        R = mat3(cos(yaw), 0, sin(yaw), 0, 1, 0, -sin(yaw), 0, cos(yaw));

        cameraPos = R * cameraPos;

    }

If (keystate[SDL_SCANCODE_RIGHT])

    {

        // Move camera to the right

        yaw = amount2;

        R = mat3(cos(yaw), 0, sin(yaw), 0, 1, 0, -sin(yaw), 0, cos(yaw));

        cameraPos = R * cameraPos;

    }
```

Above is my code for camera's rotation around the y-axis. Every time the LEFT or RIGHT key is pressed, the camera position is multiplied by the y-rotation matrix. At first I had the matrix multiplication code outside of the if statements. As a result, the camera kept rotating when I only pressed the key once. I was able to fix it after I gained a better understanding that the rendering is frame-based.

3. Shading

In the shading part, I first set the light source, and obtained the surface normal *n* and normalized the directional vector *r* for an intersection. Then the amount of direct light on that intersection can be calculated as:

$$R = \rho * D = (\rho * P \max(\hat{r} . \hat{n}, 0)) / 4\pi r^2$$

Next, I wrote an if statement that checks if there is any intersection starting from that surface point to the light source. If there is, return color black, otherwise return the reflected amount of direct light. After considering both the direct and indirect lights and multiplying them to the triangle colors, I got the final rendering result below.

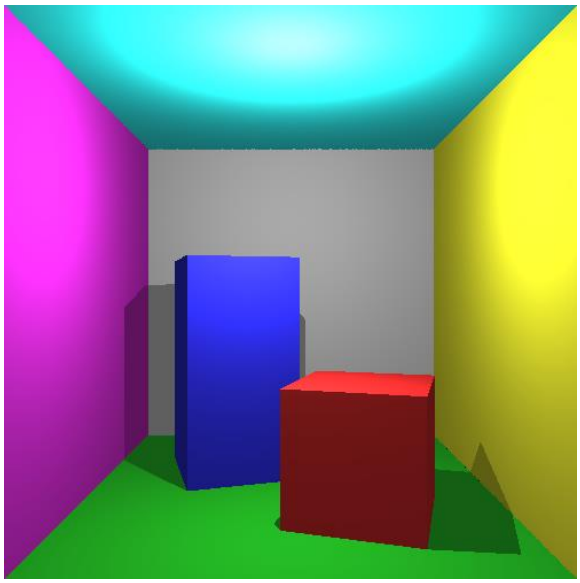

Image 3.1