Lab 2: Ray Tracing

DH2323 Computer Graphics and Interaction

Author: Mohammad Javad Ahmadi

April 04,2017

1. Loading the room

The first step is loading the room model as triangles in our triangles vector. Creating a cubic room with dimensions of x,y,z between -1 and 1. We also let our triangles save some properties such as th triangle’s vertices, normal and it’s color.

2. Camera positioning

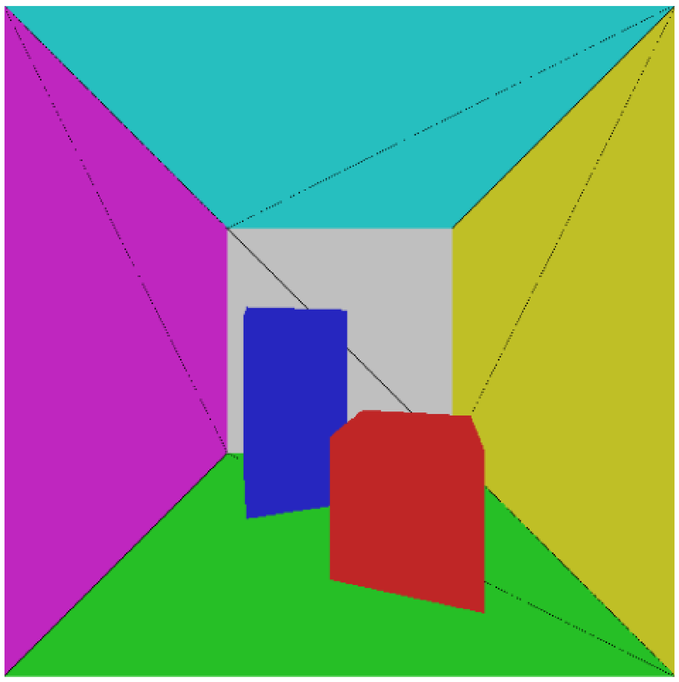
Considering that the dimension of the room is always 2 as mentioned above, therefore we place the camera distance to the room as 2 by choosing the camera position as (0,0,-2) which gives the best outcome and view of the room and the value which I choose for the focal length is screen width over 2 giving me the vertical and horizontal field of view of 90 degrees, considering that the keep the rooms dimension as square.

$$\alpha = 2 \arctan \frac{d}{2f}.$$

3. Ray Tracing

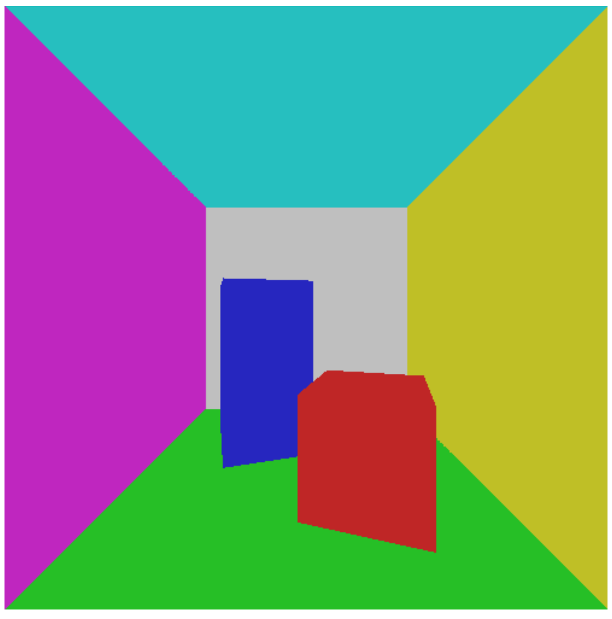
Now we need to trace each ray from the camera to our model and find the nearest intersection which helps us decide which color should be drawn for each pixel.

After finding the closest intersection I were able to get the first visual outcome on the screen which shows the cubic room room but there are apparently some black lines around the corners of our triangles as you can see on the image below.



The cause of this was that in our conditions for checking if the intersection was on the triangle’s plane or not we did not consider some points for example when u= 0 so I changed the condition to the following and it fixed the issue.

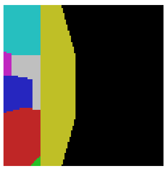
```
((v + u) <= 1 && v >= 0 && u >= 0 && t > 0)
```



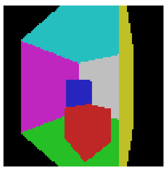
4. Moving the camera

Changing the camera position by updating the cameraPos variable was pretty simple and also we have the option to change the focal length in order to zoom in and out.

What was a bit more tricky was rotating the camera around the y-axis. The problem I encountered here was that the camera was rotating but not exactly correctly as the room was leaving the screen.



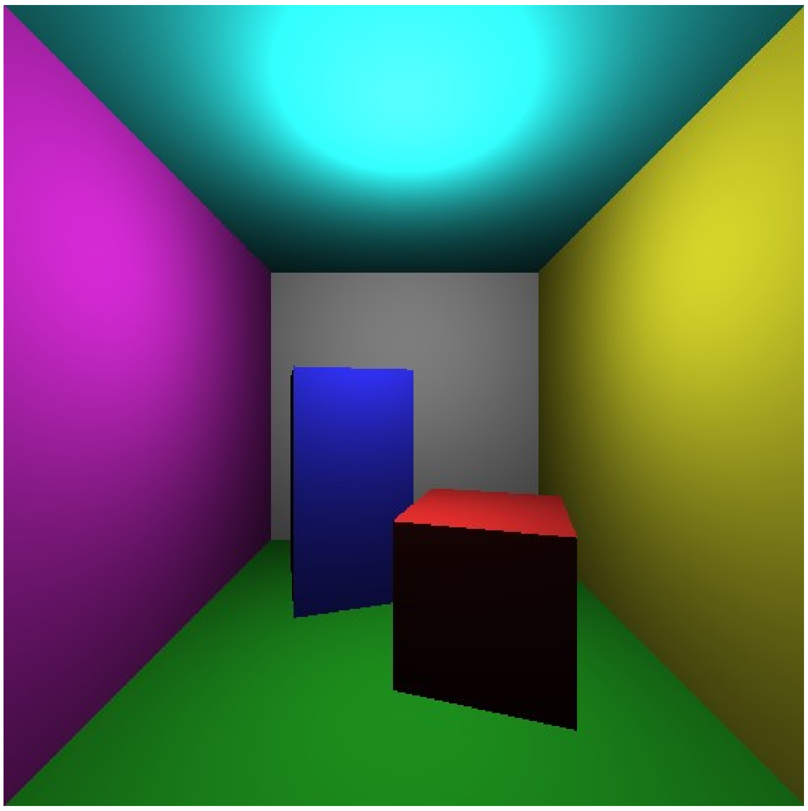
The problem here was that I was actually changing my starting location and the next time the image wanted to rotate it was rotating around the new position and not the starting position which was causing the room to shift off the screen. That was fixed by adding a global variable as the starting position of the screen.



5. Light source

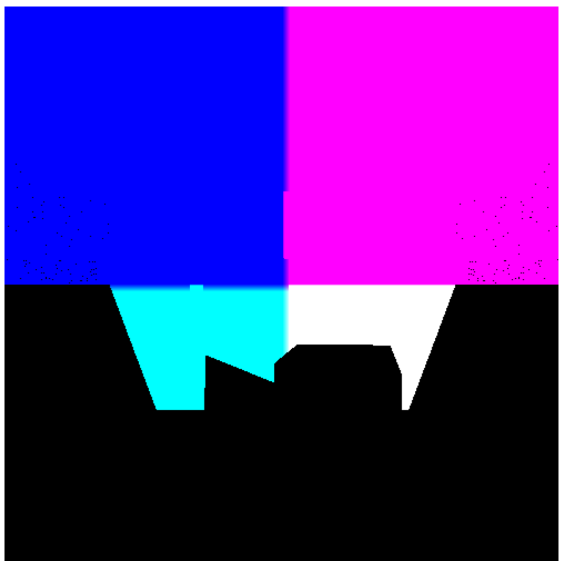
At this part of the lab we added a light source and in order to create our illumination effect we use the following formula to check the power of light which hits any part of surface in our room.

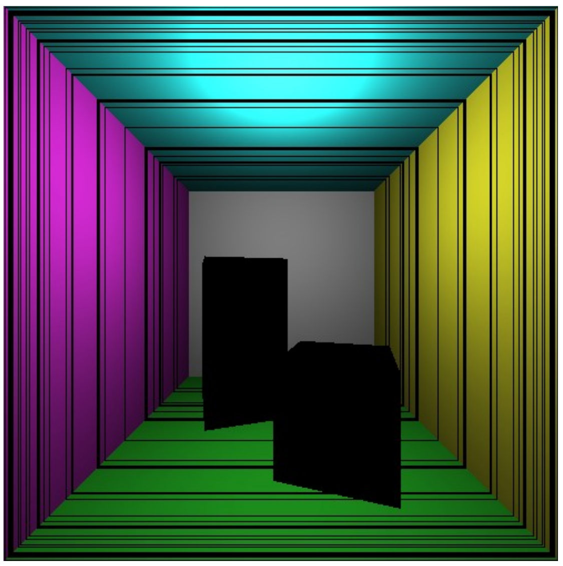
$$D=\frac{P.max(\hat{r}.\hat{n},0)}{4\pi r^2}$$



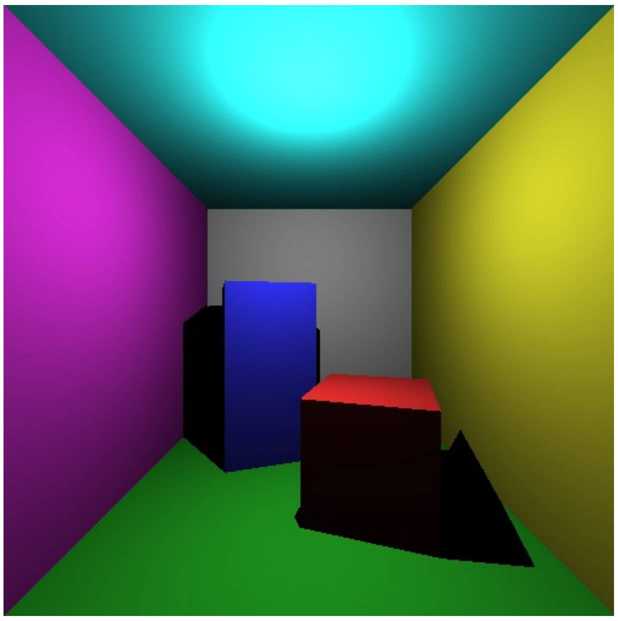
6. Direct Shadow

While finding the intersection of the ray from our surface to the light source in case there is another surface on the way we need to create a shadow, this was done in the DirectLight function. At my first attempt to create the shadow I had strange outcomes like the following.





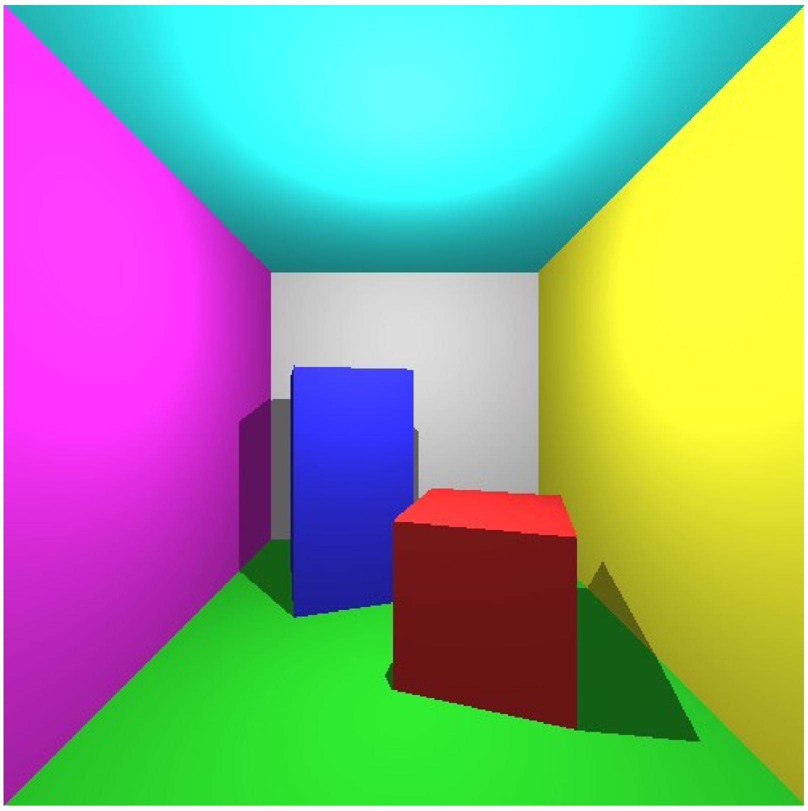
I never figured out what the cause was but after some changes in the code I finally got it to work.



7. Indirect illumination

As you can see in the picture above the shadows are pitch black and it is because we only simulate one bounce of the light with our direct illumination. Therefore we use a global variable as indirect light in order to lighten up the image and make it look like there are more light bounces.

And of course there is one step which I forgot to mention before which is moving the light around in the screen which is done by capturing some keys using SDL and moving our light source’s position around.



Here is the final result of the lab and below is a picture showing how moving the light looks like.