DH2323 Computer Graphics with Interaction
Lab 2 : Raytracer

Viktor Collin
<`vcollin@kth.se`>
19880316-0277

Simon Österman
<`simost@kth.se`>
19880205-0156

May 13, 2013

Total pages: 3

1 Introduction

The purpose of this lab is to learn how to build a raytracer and use it to render an image of a 3D environment that consists of triangular shapes. The lab also includes light models, shading and camera movement.

2 Assignment

The assignment is to implement a raytracer which can create images given a 3D-model by tracing the rays reaching a simulated camera. The implementation should be done in several steps, each with new functionality or other improvements. The final implementation should include a mobile camera, simulated light-sources as well as light reflection for diffuse surfaces.

3 Method

3.1 First implementation

As a first step we built our raytracer to send a ray thru each pixel on the screen and fining the closes intersection with the predefined model of the room, then it render that pixel to be the color of the intersected shape. If the ray did not hit any shape we render the pixel to be black. See the output of this in figure 1.

3.2 Camera movement

Next step was to implement an interactive environment were the user with the help of the keyboard should be able to move and rotate the camera. Rotations is implemented by increasing or decreasing an angle and then updating the rotation matrix `R`. movement in the forward and backward directions is then done by multiplying `R` with a movement speed in the `z`-direction. The rays sent thru each pixel is then sent in the direction the camera is facing.

3.3 Direct light

To introduce shading we introduced a light source in the ceiling of the room. When the ray from the camera hit a shape if the normal of that shape is facing away from the light there is no light that hits the surface. See the output of this in figure 2a. We also sends a new ray from the intersection point towards the light source to see if there was an object shadowing that pixel. If so the pixel was painted black otherwise the amount of light was calculated based on the distance from the light. At first this implementation did not take in to consideration the color of the shape that was hit by the first ray. See the output of the shading in figure 2b. We also implemented a possibility for the user to move the light source in a similar way as moving the camera but instead of rotating it moves in the `x`-direction. To take in to consideration the colors of the boxes and walls we just multiply the color with the direct light. See the output of this in figure 2c

3.4 Indirect light

Instead of making the light bouncing and defusing on each surface witch is a complex task we just overlay the direct light with a uniformed light for all pixels. This is a much simpler way an the result is acceptable realistic. We just add a constant light to the direct light before color computation. the result can be seen int figure 3.

4 Result

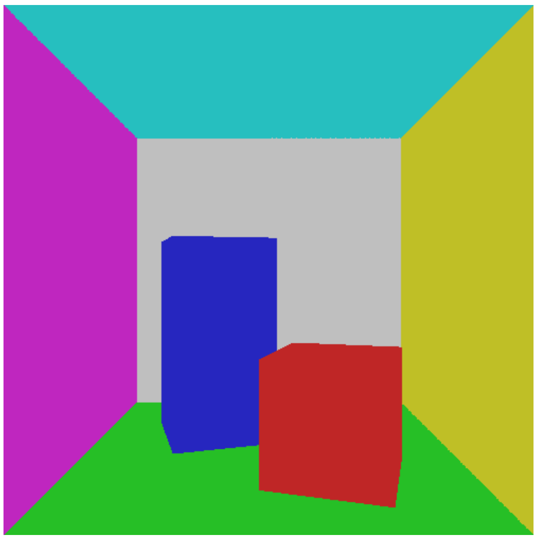
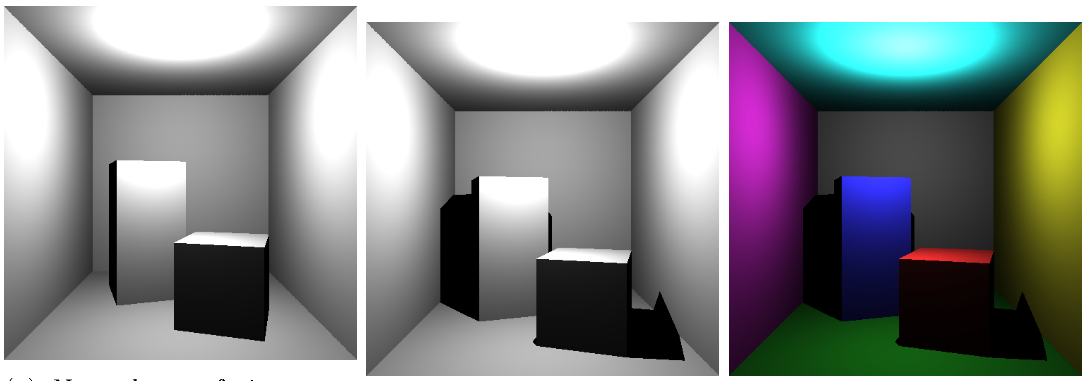


Figure 1: Output from first run



(a) Normals are facing away from light source (b) With shadows (c) With shadows and colors

Figure 2: Progress of the direct light model

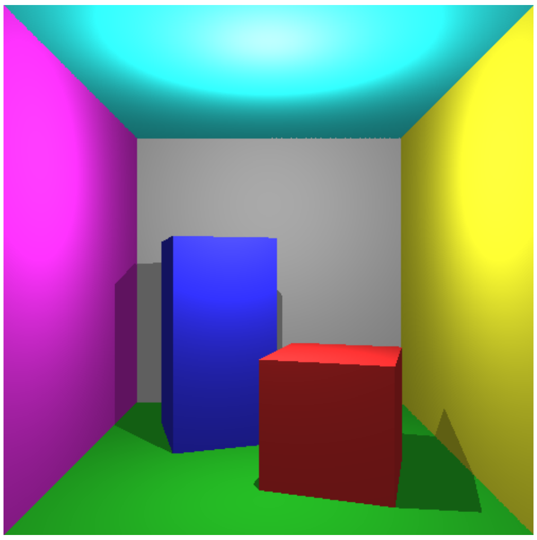


Figure 3: Output from finished program