Ray Tracer Implementation

Design Laboratory Project

Aditya Singh

Indian Institute of Technology Kharagpur

November, 2022

Abstract

Ray Tracing is a very popular algorithm used in computer graphics for environment illumination. It can be used to simulate various optical effects to make our graphical scenarios more realistic and visually appealing. In this work, I will develop a custom python ray tracer software with a UI that enables the user to add/remove objects and alter the position, color and light parameters. Through this implementation, we will then generate various scenarios and observe how our ray tracer will illuminate these environments.

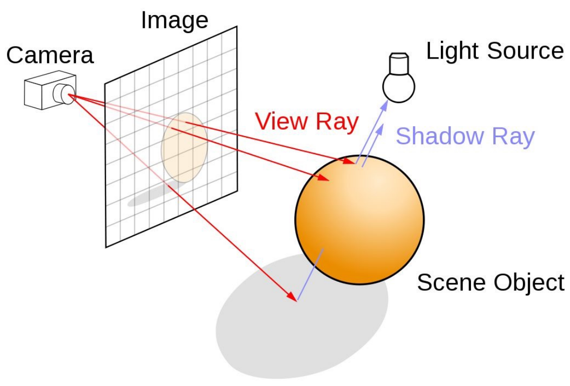


Figure 1: Ray tracing

1. Introduction

This report is a detailed description of my work as part of my design laboratory project (CS59001). My objective for this project was to implement a simple ray tracer in Python. I created a simple UI through which the user can add spherical objects in a pre-designed environment and shine light on them from a single point source.

2. What is Ray Tracing?

In 3D computer graphics, Ray tracing is a method of graphics rendering that simulates the physical behavior of light to produce incredibly realistic lighting effects. It is capable of simulating a variety of optical effects, such as reflection, refraction, soft shadows, scattering, depth of field, motion blur, caustics, ambient occlusion and dispersion phenomena (such as chromatic aberration). It is a very computationally heavy technique and requires very capable hardware to perform it efficiently.

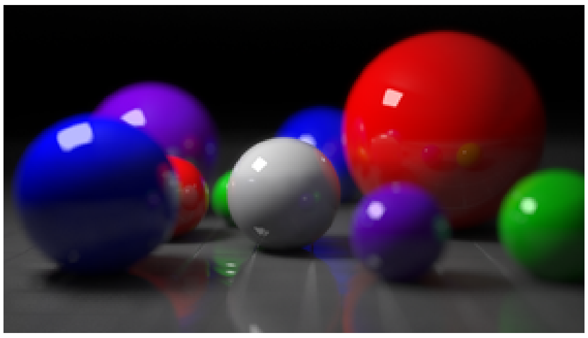


Figure 2: Illuminated scene

Ray tracing is popularly used for various applications like environment illumination, animation, lighting architecture, etc. Many modern video games nowadays come with a ray tracing graphic option available as it makes the game world look more realistic and aesthetically pleasing. In the image below, we can see how enabling ray tracing causes the in-game light to bounce off the floor.

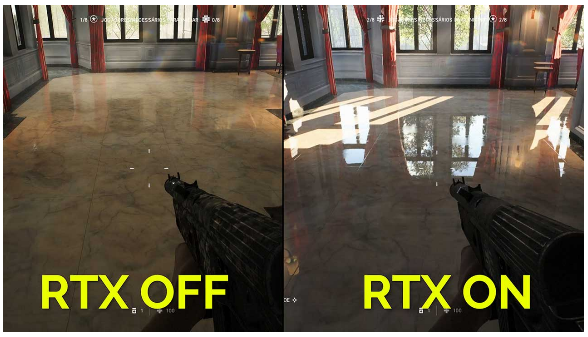


Figure 3: Ray tracing effect comparison

3. Project Implementation

To implement this project, I had to work on 2 main components : the UI and the ray tracer. The UI is gonna handle the change in the environment parameters and settings, and based on those parameters, the ray tracer is gonna generate the final image and display it.

3.1. Ray Tracer

Our ray tracer is going to trace a certain number of light rays from a point source shining light on a pre-designed scene while taking in account various factors like reflections, diffusion, shadow casting, etc. Our scene is going to consist of the following

- A checkerboard flat surface to check for reflections
- A certain number of spherical objects
- A point light source emitting light in all directions
- An "eye" or a camera to observe the scene
- A screen through which our camera is looking. We have to calculate the color for the pixels for this screen

When a light ray falls on a surface, it bounces off in certain directions. When this reflected ray reaches our eye, we can see the object. This is how illumination takes place in real life. However, this is very inefficient to implement for our ray tracer since most of the reflected rays won't reach our eyes. Hence, we will do the reverse process, staring from our camera and only considering those rays as they are the ones reaching our "eye".

We follow the algorithm given below :-

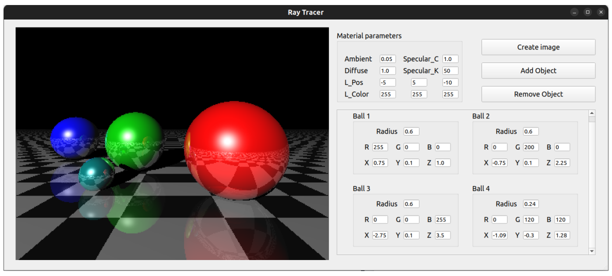
- Associate a black color to every pixel P of the screen
- Trace all possible rays from the camera towards all pixels of the screen and do the following for each
- If a light ray from the camera goes towards P then calculate the intersection point of the light ray with any object X in the scene
- If there are no obstacles between X and the light source, calculate the color of the intersection point and associate it to P
- If there is an obstacle, that means that light cannot reach this point (shadow region)

3.2. UI design

For our interface, we are going to be using PyQt5 designer to create a simple and easy to understand UI. Our UI will display all the light, color and position parameters. It will contain a region where our final image will be displayed, an add/remove object button and a create image button which will start the ray tracing process.

4. Final Look and Results

After implementing the ray tracing function and the UI, we finally complete our application and the final look is shown below.



Material parameters

Ambient	0.05	Specular_C	1.0
Diffuse	1.0	Specular_K	50
L_Pos	-5	5	-10
L_Color	255	255	255

Ball 1
Radius 0.6
R 255 G 0 B 0
X 0.75 Y 0.1 Z 1.0

Ball 2
Radius 0.6
R 0 G 200 B 0
X -0.75 Y 0.1 Z 2.25

Ball 3
Radius 0.6
R 0 G 0 B 255
X -2.75 Y 0.1 Z 3.5

Ball 4
Radius 0.24
R 0 G 120 B 120
X -1.09 Y -0.3 Z 1.28

Figure 4: Final Look

On the right pane, we can see all the light parameters including position, color, diffusion and ambient coefficients, etc. We also see the attributes of all the spherical objects currently present in the scene. The user can modify their radius, color and position as well as add/remove more objects using the add and remove buttons provided. To show the final illuminated scene, the user has to click on the "Create image" button which calls the ray tracing function on the current scene. The ray tracer takes around 15 seconds to generate the image which is relatively slow as it takes time to trace all possible rays. However, we get a detailed illuminated scene displayed on the left. In the example shown above, we can see the intricate details like the shadows casted by the spheres, their reflections on the ground, the reflection of the checkerboard pattern on the glossy surface of the spheres, the shading of the sphere surface where the light is hitting, etc.

Hence, our ray tracer is generating highly detailed illuminated scenes in a relatively short period of time.