King's College London

7CCSMGPR

Group Project

Group of Seven Report

Authors:

Number	Name	Email
1756850	Othman Alkhamra	othman.alkhamra@kcl.ac.uk
1739256	David Benicek	david.benicek@kcl.ac.uk
1770922	Aadam Bari	aadam.bari@kcl.ac.uk
1425704	Hitesh Mankani Vinod	hitesh.mankani_vinod@kcl.ac.uk
1755013	Timothy Obimma	timothy.obimma@kcl.ac.uk
1783087	Nagarjuna Vaddiraju	nagarjuna.vaddiraju@kcl.ac.uk

supervised by
Dr. Laurence Tratt

March 7, 2018

Contents

1 Introduction

In real life, we have different objects such as spheres, cubes ... etc. What is common between them is that they are all geometry objects. However, those objects might appear in different ways, based on their environment. For example, a glass sphere placed in an environment with many light sources will react in a different way than a metal sphere. And so, it is clear that different factors such as the material type, light sources and many other factors, will produce many different outputs.

Ray tracing is one of those techniques which helps in producing different images for different environment variables. It is widely used in producing films, video games, animations, ... etc. It can be applied on any object and not just geometry objects.

1.1 What is the Idea behind Ray Tracing?

Before discussing the idea behind ray tracing, and explaining its algorithm. Let's begin with listing the basic required elements to run the ray tracing algorithm. Those elements will be in one container which will be called **scene**. The elements of the scene are:

- **Object(s)**: a scene can contain one or many objects. Each object can be one of the previously mentioned objects or any other object, which exists in real life. For example, an object could be a sphere, triangle, tree, car, building ... etc.
- **Light source(s)**: The algorithm of ray tracing itself doesn't require having a light source. However, a scene without any light source, won't make the use of ray tracing effective.
- **Image Plane**: which is called **window frame** in our implementation. This plane will have some width and height, and it will be divided into small squares, and each square will be covering a number of pixels on the objects of the scene.
- **Eye or Camera:** In order to render the scene, it is mandatory to have an eye or camera, with some position and some direction, to detect the way of looking to the scene elements. From that eye or camera, rays will be produced, and will go through the image plane.

Figure 1 shows an example of a scene with the previously listed elements. Now, and after discussing the elements of the scene, it is the time to describe the algorithm of ray tracing. The idea is so simple, in each scene there will an eye or camera, from that camera many rays will be produced and go across the small squares in the image plane, for each ray we need to check whether it intersects with any of the objects or not, and return the color of that pixel.

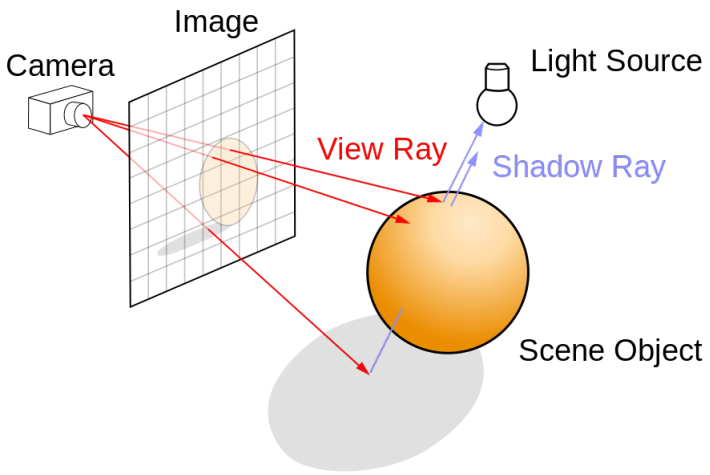

Figure 1: Ray Tracing's Scene Elements [1]

The algorithm itself is really simple. Nevertheless, it is previously mentioned that environment variables play a big role in the calculations. And so, returning the color of the pixel, which was hit by the the ray, won't be that simple; since it will require additional computations related to the reflections, shades ... etc.

1.2 Our Solution

Although the idea behind ray tracing is simple. But, there are many different problems to be addressed in this program. These problems can be summarized as below:

1.2.1 Web Interface

In order to define the different elements of the scene, a web interface is developed to allow users enter the different values for the scene elements. Using this interface, the user can add different objects to the scene, with different materials, positions, sizes, colors. Furthermore, adding different lights at different positions with different colors.

The challenge here is that objects must be in 3D. And so, we developed a web interface with two options. One option is the 2D option, where the user needs to define an element by providing its values in the top and side views. The other option, is the 3D view, where the user can add elements directly in 3D format.

All the functionalities in our web interface support the drag and drop feature. In other words, this means that the user can define the position by clicking on the object, and moving it in any direction.

1.2.2 Ray tracing Engine

The other part of our system is the backend engine, where all the calculations are done based on the input received from the web interface. This system was developed using .NET C#. It includes a post API, which will receive the request from the client, and it is able to render a scene with the followings:

- A scene with the following objects:
  - **Plane**: This object will be used to create the walls in the scene. And it will have position, and a normal which will decide the orientation, in addititon to one of the supported materials.
  - **Cube**: Each cube will have a size, a position, and one of the supported materials.
  - **Sphere**: Each sphere will have a radius, center point, and one of the supported materials.

  The **supported materials** are:

  - **Phong materials**: which will follow the phong theory such as chalk, metal, plastic. The difference between those materials are the different values of diffusion, specular coefficients and the light color infulence.
  - **Flat material**: which is the simplest material, which will return the color of that object regardless to any other environment variables and parameters.
  - **Glass material**: this material is the most complicated one, which will do many different calculations in order to calculate the color of each pixel of that object.

  Each of these materials can have a color chosen by the user, or just use the default color, if **nothing** is received.

- A scene with multiple light sources at different positions and different colors.
- A scene with perspective camera, this camera will have the position, the direction of how to look at the image plane, and the distance from that plane.

References

[1] An overview of the ray-tracing rendering technique. https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview. Accessed: 2018-03-07.