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 15, 2018

Contents

1 Introduction

In real life, we have different objects such as spheres, cubes ... etc. What's mutual 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 the techniques which helps in producing different images for different environment variables. It is widely used in producing films, video games, animations and many other areas. 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 types 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 isn't very practical for ray tracing.
- **Image Plane**: referred to as **window frame** in our implementation. This plane has a set width and height and it is divided into small squares, where each square covers 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 a certain position and direction. The camera is used to determine the way of looking to the scene elements. From the eye or camera, view rays are produced which travel through the image plane and determine the way the object is rendered.

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. Despite the implementation not being trivial, the main idea is quite 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

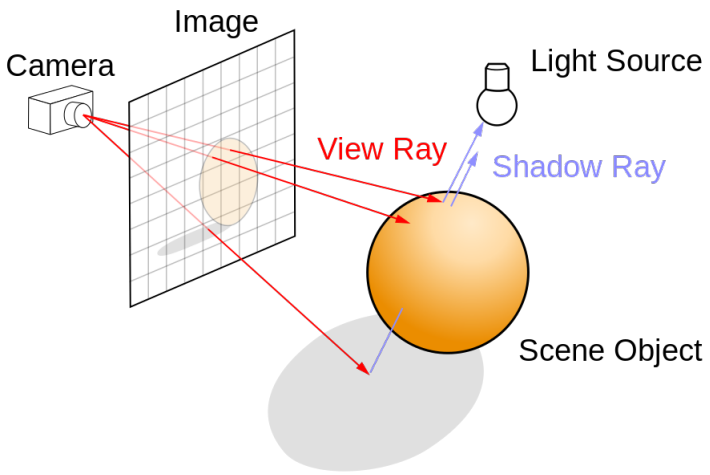

Figure 1: Ray Tracing's Scene Elements [1]

or not, and return the colour of that pixel.

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 colour 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, there are many different problems to be addressed in this program. These problems can be summarised 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, colours. Furthermore, adding different lights at different positions with different colours.

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 functionality in our 2D web interface supports the drag and drop feature. In other words, the user can define the position by clicking on the object, moving it in any direction, and from the 3D view, the user can view the scene.

1.2.2 Ray tracing Engine

The other part of our system is the back-end 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 following:

- A scene with the following objects:
  - **Plane**: This object is mainly used to create the walls in the scene. And it has a position, and a vector called normal which decides the orientation, in addition to one of the supported materials.
  - **Cube**: Each cube has a size, a position, and is made up of one of the supported materials.
  - **Sphere**: Each sphere has a radius, centre point, and is made up of one of the supported materials.

  The **supported materials** are:

  - **Phong materials**: which follows the phong theory such as chalk, metal, plastic. The difference between those materials is the values of diffusion, specular coefficients and the light colour influence.
  - **Flat material**: which is the simplest material, that will return the colour 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 colour of each pixel of that object.

  Each of these materials can have a colour chosen by the user, or just use the default colour, if null.
- A scene with multiple light sources at different positions and colours.
- A scene with perspective camera. Each camera has a position, direction of how to look at the image plane, and distance from that plane.

2 Review

2.1 Brief History of Ray tracing

Ray tracing is one of the popular methods of rendering. What is Rendering? Rendering is the act of generating an image (usually photorealistic) from a 2D

or 3D model. There are three familiar techniques to image rendering and they are rasterisation, ray casting and ray tracing. These three techniques are quite similar but are differentiated by their representation of the optical effects such as reflection intensity which in turn makes the processing speed vary differently (Wald, Slusallek, Benthin, & Wagner, 2001). This means that ray tracing would produce the most photorealistic image while also taking the most time to compute while rasterisation would produce the most basic image.

The earliest known Ray tracing system was calculated by hand by Richard Hoyt in the 1950's while working on vehicle shotline at the Ballistic Research Library known today as DARPA (Klopcic & L. Reed, 1999). In the 1960's, physicists coined the name "ray tracing" by plotting on paper the path taken by rays of light starting at a light source and passing through the lens in the design of lenses (Glassner, 1991).

Computer Graphic researchers at the university of Utah thought it would be a good idea to apply this simulation of light physics to create images but the computers of the 1960's were too slow to produce images of better quality than those made by using cheaper image rendering techniques hence the idea of ray tracing was dropped for several years.

Over several years, as computers grew more powerful, it became imperative to simulate the real physics. Several optimised algorithms were implemented and this led to computers being able to simulate various kinds of optical effects. Ray tracing have evolved greatly since then and is now one of the most powerful technique used in image rendering. Ray tracers have become both faster and more efficient and can even produce hyper-realistic images that are indistinguishable from the real world. We will Discuss some Ray tracers in detail below.

2.2 Overview of Ray Tracing

2.3 Review of Related Works

2.3.1 Nvidia Optix

Not a Ray tracer

2.3.2 BRL-CAD

2.3.3 OpenRL

3 Requirements and design

3.1 Mandatory Requirements

The beginning of the project saw the group discuss and decide what features were essential for the running of the ray tracer, that is to say, creating and developing a running system, consisting of the front-end, which is the client side, and the back-end, which is the server side. The bare minimum requirements for the entire system are as follows:

- Allow users to create spheres and cubes with specific parameters' values. Those parameters are:
  - The size of the object.
  - The position of the object.
  - The material of the object; determining its colour and translucency.
- Allow users to assign values to the parameters of the environment. The parameters include the following:
  - Position of the camera.
  - Size of the output image and the window frame.
  - Filename of the output image.
  - Colour of the background.
  - Light's colour and position.
- Render multiple objects in the same scene, with at most one light source.

3.2 Optional Requirements

If time allowed, the group established additional requirements. The requirements specified in this category will extend the basic functionality of the ray-tracer, thus distinguishing our product from others. The optional requirements are as follows:

- Allow users to add multiple lights to the scene in different positions, colours and intensities.
- Allow users to create other objects such as polygons.
- Allow users to choose different materials, such as plastic, glass, etc. for the objects.
- Improve the colours of the output image, by using different techniques such as sampling.

- Improve the performance of our system, by applying the multithreading concept, if the systems takes a long time while rendering the scene.

Due to time constraints we were unable to implement the use of polygons. However we successfully achieved the other optional requirements.

3.3 Methodology

The methodology the group decided to adopt was the Agile software methodology.

3.4 Architecture

This project has a two-tier, client-server architecture.

1. The front-end client is served up by the server. The front-end has a user interface that allows the user to input their shapes and it will also dynamically render an approximation of the input to visualise what the user is about to ray trace. The front-end will primarily be coded in HTML, CSS, JavaScript and AJAX.

2. The back-end has a number of controllers that will dynamically process the content specified by the user and generate a ray traced image. The back-end is built in .Net, using WebApi2 to run the local server.

MVC

4 Implementation

4.1 Front End

4.2 Back-end

4.2.1 Unit Testing

5 Team work

5.1 Process

Weekly meetings At the start of each meeting standup style sync up: Everyone speaks, says what they did, what still needs to be done, calls out any blockers that we can go over in the follow on meeting. Discussion: Talk about what the next steps are, what could be improved, what has blocked us in the previous sprint. Try arrive at outcomes - outcomes made into tickets Next steps: Illicit next steps both from discussion and from product roadmap

5.2 Communication

Communication is one of the most important factors when it comes to working in a software engineering team. For our project we decided to utilise a number of

different structured and unstructured communication channels. There exists a large array of great project management and issue tracking tools such as JIRA, Trac or Trello, however, we decided that given the brevity of our project and the relative low number of backlog items (especially at the start), that it would be best to keep our ticket system as close to the codebase as possible. As a result of this, we decided to make use of the inbuilt GitHub issue tracking system. This system does not provide the same calibre of project management tools as some of the forementioned products, however, it does enough for our purposes. We decided to use the GitHub milestone system as a way of scheduling issues for our sprints and implemented a branch naming convention such that we would refer to the issue we're addressing in the name of the branch - {issue number}/{name of branch}. This simple mechanism has worked tremendously because it ensures that on one hand all branches are addressing exactly one tasks or feature and on the other hand, that all tasks and features have been recorded in an issue.

Another way we structured our communication was by the use of pull requests whenever pushing code to master. By raising a pull request and having another team member review it we not only improved our code quality but also spread knowledge of the code base and cross-polinated ideas for features or more efficient solutions. Standout examples of a pull request flow can be seen here: TODO: Cite some nice PR from github

Not all communication can be done over issue summaries and pull requests and therefore we scheduled weekly meetings to go over what has been achieved, what needs to be discussed and what is the plan for the following sprint. One could say that we condensed our daily scrum meeting, sprint planning, backlog refinement and sprint review all into one weekly meeting. This is not the textbook way of running an Agile project but given that we are not working on the project full time and that these meetings were kept as brief as possible, it still alligns to the core principles of Agile. During the meetings we would discuss the general trajectory of the project and what we want to accomplish in the next seven days, before breaking off into our sub teams and deciding on the exact issues we needed to create for the week ahead. This way of running a project has both advantages and disadvantages which is covered in section TODO: Cite section that covers this. We mostly met in person but on occasion, when we needed to, we used video conferencing tools such as Appear.In and Gruveo to connect with eachother.

During the course of the sprints, our communication did not fade. We built on some ideas from Extreme Programming and met frequently in our work units to engage in pair programming while working on issues during the sprint. To facilitate the smooth operation of the team we used the lowest friction medium of communication for all of us which was a WhatApp group where we have discussed everything from meeting times, status of different issues or even specific lines of code in pull request.

5.3 Tools

Things that enabled us Visual Studio, Visual Studio Code, Git, Things that kept us on track Travis, Mocha, Istanbull, Things that connected us Appear.In, Gruveo, Whatsapp, Facebook,

6 Evaluation

Considering the mandatory requirements that were previously set out, all of them have been accomplished. On the other hand, our optional requirements have not been completed. However, if the deadline was not a factor, they would have been completed as most of them were quite similar to the mandatory ones.

As mentioned earlier, the team was divided into two. This was a success as each team member decided whether they wanted to work in either the front-end or the back-end depending on their own skills,desires or work experience. This was crucial as everyone choose the area they were more comfortable working with which allowed less issues to arise in the future. Each sub-team also had at least one member who had work experience in the particular field. David had experience in the front-end, whereas Othman had experience in the back-end. This was important as both of them were able to lead their sub-team and any problems would be solved efficiently. It also gave the rest of the team members a look into the future as both of their experience helped the team visualise and learn how different code practices and tools are used in the workplace.

Focusing on the mandatory requirements first seemed logical as they were the functionality required for the software to work. Once those were accomplished, the optional requirements would be prioritised. This process went well as we did manage to finish the mandatory requirements. Most of the mandatory tasks were divided in the sub-teams which lead to both individual work and working in pairs. Critical tasks such as SVG implementation on the front-end or creating different shapes in the back-end, was either developed through pair programming or each team member had a small task to do in them, for example one team member would create a cube and the other a sphere. This helped us participate in every aspect of the project and improved our skills such as team work and communication. This also lead to coding problems being solved quickly as two minds are quicker and more knowledgeable than one. On the other hand, small tasks that were also important such as fixing a bug or creating forms, were done individually as not much time was needed for them.

Communication was a major downside that could have been improved between the two sub-teams. This only came to our attention a few days before the first presentation while we were trying to integrate both front-end and the back-end together. It was a small issue at first which became big due to the fact that: variable names were not matching e.g spelling colour in the British way or American way; or the variables that were supposed to be hard coded either in the back-end or front-end e.g the back-end team assumed that the user would input where the camera would be pointing at. While the front-end thought that

the back-end team would have it set as default and the user would not be able to change that as it made sense to always have the camera aiming at the middle of the walls with the user being able to change the position of the camera. This led to some confusion and a lot of changes in the code which wouldn't have been an issue if communication was more effective since the beginning. Communication became better once these issues came out and were pointed out to us but there was still some lack of effective communication.

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.