

Final Report

|  |
| --- |
| School of Computing  Faculty of Engineering AND PHYSICAL SCIENCES |

<Effective rendering curve primitive in ray tracer>

<Hanmun Hwang>

< BSc Computer Science >

**<2023/24>**

**<** *COMP3931 Individual Project* **>**

The candidate confirms that the following have been submitted*:*

*<As an example>*

|  |  |  |
| --- | --- | --- |
| **Items** | **Format** | **Recipient(s) and Date** |
| *Final Report* | *PDF file* | *Uploaded to Minerva (DD/MM/YY)* |
| *Scanned participant consent forms* | *PDF file / file archive* | *Uploaded to Minerva (DD/MM/YY)* |
| *Link to online code repository* | *URL* | *Sent to supervisor and assessor (DD/MM/YY)* |
| *User manuals* | *PDF* | *Sent to client and supervisor (DD/MM/YY)* |

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)

© <Year of Submission> The University of Leeds and <full name of candidate>

# Abstract

*<Concise statement of the problem you intended to solve and main achievements (no more than one A4 page)>*

*Drawing simple geometries, such as sphere, triangle and etc, on the window is quite simple as long as we know some vertices which are forming them. When it comes to a realistic camera imaging, there are a number of considerations like “How to capture 3D appearances into 2D screen?”, “How does the interaction between ray and objects represent?” and so on.*

*In 3D graphic, the technique called ‘Ray-tracing’ is known for solving such doubts.*

# Acknowledgements

*<This page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by others to the project.>*

*Note that it is not acceptable to solicit assistance on ‘proof reading’ which is defined as “the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the text”; see*

https:://www.leeds.ac.uk/secretariat/d`ocuments/proof\_reading\_policy.pdf

# Contents

1 Introduction and Background 1

1.1 Ray-Tracing 1

1.1.1 Basic principles 1

1.1.1.1 Ray 1

1.1.1.2 Rendering pixel by pixel 1

1.1.1.3 Intersection 1

1.1.2 Shading 1

1.1.2.1 Diffuse and Specular Reflection 1

1.1.2.2 shadow computation 1

1.1.2.3 Reflection and Refraction 1

1.1.2 Camera model 1

1.1.2.1 Diffuse and Specular Reflection 1

1.1.2.2 shadow computation 1

1.1.2.3 Reflection and Refraction 1

1.2 Bezier curve 1

1.2.1 Mathematical Concepts 1

1.2.1.1 Geometric properties 1

1.2.1.2 Bernstein Polynomials 1

1.2.1.3 Bezier basis Matrix 1

1.2.2 The De Castelão Algorithm 1

1.3 Complex primitives by Curve primitvies 1

2 Methods 2

2.1 Development tool 2

2.2 Ray-Tracer development 2

2.2.1 Mathematical Concepts 1

2.2.1 Mathematical Concepts 1

2.2.1 Mathematical Concepts 1

2.3 Implementation of Bezier curve 2

2.3.1 Segment ray-intersection 1

2.2 Mesh model by curve 2

3 Improvement/Optimisation 2

3.1 Anti-alising 2

3.1.1 Theory 1

3.1.2 Result and comparison 1

3.2 Bounding Volume Hierarchy (BVH) 2

3.2.1 Theory 1

3.2.2 AABB 1

3.2.3 Method 1

3.2.4 Result and comparison 1

4 Results and Evaluation 2

5 Discussion 2

References 3

Appendix A Self-appraisal 4

A.1 Level 2 Heading with ‘heading 2’ Style 4

A.1.1 Level 3 Heading with ‘heading 3’ Style 4

A.1.1.1 Level 4 Heading with ‘heading 4’ Style 4

Appendix B External materials 4

B.1 Level 2 Heading with ‘heading 2’ Style 4

# Chapter 1

### Introduction and Background Research

## Ray Tracing

Adjusting a sequence of pixels for rendering common geometries in window is not enough to generate a realistic image. When taking a photograph with a camera, does the image appear as a two-dimensional flat plane? No, while the photograph itself is a two-dimensional representation, modern cameras are designed to capture depth, texture, and perspective, which give the image a sense of three-dimensionality. How about taking a picture at night without a flashlight? It will likely turn out black. Raytracing is a computer graphic technique that emulates the way light reflects and refracts in the real world, providing a more realistic and believable environment.

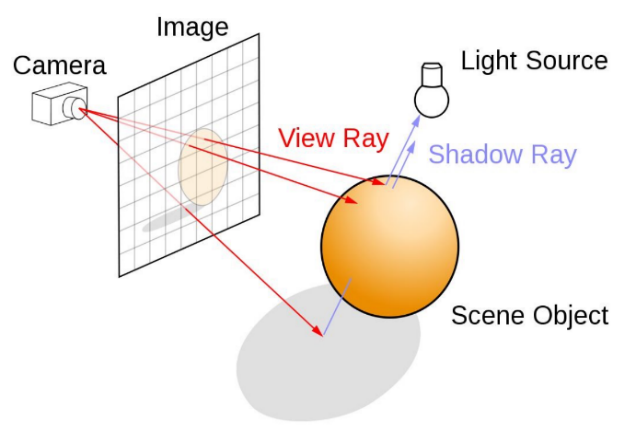


Figure 1

As we know Intuitively, this technique is literally “Tracing a ray”. What comes in our thought when we see a thing is to stare at something but from the perspective of computer graphics or physics, it is of ‘light detection’. Rather than a specific substance or wave from the eye to detect an object, light scattered in all directions is reflected or refracted and enters our eyes. When we chase the ray reversely, it is possible to explain where the source of ray is and how an object looks like once detected something. In other words, by tracing the light that reaches the human eye back, it becomes to mimic how the object should appear in that situation(see Figure 1).

In 1979, Turner Whitted firstly introduced a method for computing ray reflections and refractions using ray tracing techniques, marking a significant advancement in the creation of realistic images in computer graphics. This approach laid the groundwork for simulating complex optical effects such as shadows, reflections, and refractions, thereby enhancing the visual fidelity of computer-generated imagery. His work enables developers to simulate the lighting of a scene and its object with diverse effects.

### Basic principles

#### Ray

A ray symbolizes essentially a beam of light. Just as our eyes perceive the surrounding world by receiving Rays emitted from various sources, a ray-tracer uses ‘ray’ to simulate the interactions between light and virtual models, presenting them through the viewport or camera. Rays enter the camera from the respective pixel, calculating and simulating the path that light takes. The ray can be mathematically represented as a point defining the ray’s starting position in space, along with a directional vector indicating its direction.

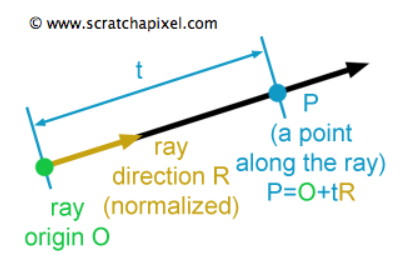


Figure 2

P : A position along the line in 3D

O : Ray origin

: Ray direction

t : Ray parameter that scales

The parameter t in this equation can either be positive or negative. If t is positive, the point P value in front of the ray’s origin point. The other way if t is negative, the point P is behind the ray origin. In general ray-tracer, the intersection points between ray and surfaces always be in front of the ray’s origin that means we do not consider situations occurring behind the ray's origin.

#### Rendering pixel by pixel

In a similar manner like the natural propagation of light from sources, ray-tracer casts primary rays(employing the formula 1.1.1.1 ) from camera origin into the scene, rendering pixel by pixel based on the geometries these rays intersect with. To determine the direction of each ray, a virtual viewport or pixel frame is utilized, facilitating the precise tracking of each ray's path(see figure 3).

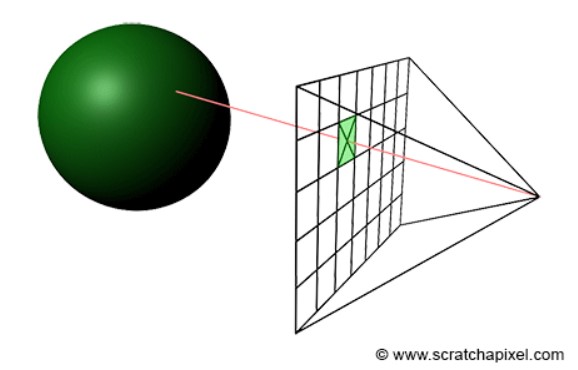


Figure 3

A primary ray passes through each pixel on the grid frame by which it requires to know the coordinate of middle within each pixel. For calculating the middle of a pixel, unit u and v vectors are used in this ray-tracer(see figure 4).

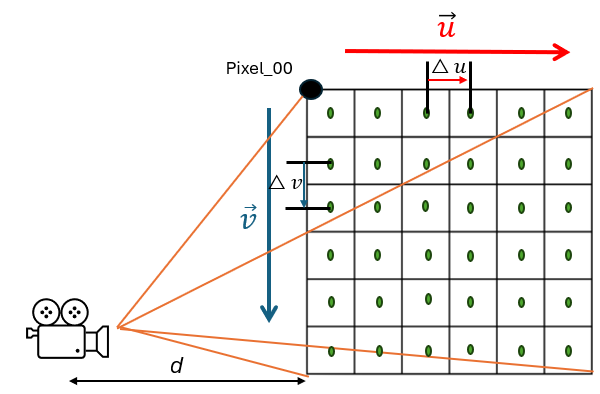
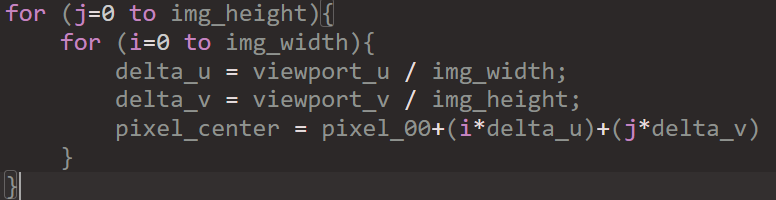


Figure 4



#### Intersection

We now turn our attention to the critical concept of ray-intersection, the process by which rays determine visible surfaces by intersecting with scene geometries. In this part, a sample geometry is needed to explain and calculate whether a ray hits or not. Sphere is a decent option because it is relatively simple which is best. Looking back in mathematic class, we have learnt the equation of a sphere as shown below (1.1).

(1.1)

As for 3D graphics, those need to be converted into vectors so that x, y, z axis is under the hood in the Vec3 class(Used to represent vectors in 3d space). And we will utilize above equation for determination of ray-object intersection(see figure 5).

Vector : It is as is the vector .

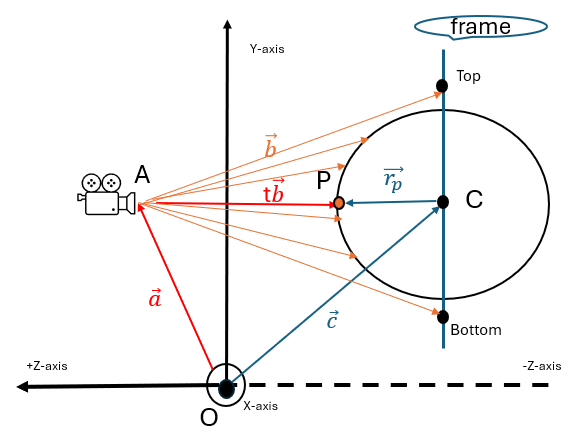


Figure 5

Vector : Vector camera to the viewport.

Vector : Sphere centre to a potential intersection point P.

Vector t : If there were an intersection along the vector would be t (t is real number).

A math equations with numbers and symbols

Description automatically generated

The left solving process for the math equation looks hard to understand, but the vectors and radius in that equation are all constant and known. The only unknown is t, and we have a , which means that this equation is quadratic. It is possible to solve for this equation form by using quadratic formula as shown below (figure 6).

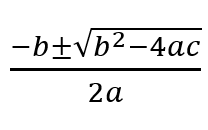
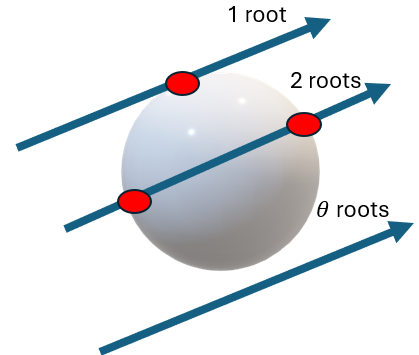
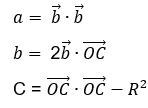


Figure 6

From the quadratic formula, we can induce the discriminant equation using the given values a, b, c. means two real solutions, means one real solution, and means no real solution. With this discriminant analysis, the identification of ray-intersection with object will be feasible. Even though ray-sphere intersection is not directly relevant to curve primitive, this process could be included in the part of ray-curve intersection later.

## Bezier Curve

Unlike simple basic primitives, the curve called ‘Bezier curve’ is flexible to handle diverse shape of objects if multiple vector points are given. It was devised by French engineer Pierre Bezier (1910–1999) who used it for designing the bodywork of cars back in 1960s. Coincidently, his curve has been playing a pivotal role in computer graphics such as user Interface design or animation. In terms of smoothness, this curve can be scaled up or down without any loss of quality.

### 1.2.1 Mathematical Concept

Bezier curves are a mathematical concept used to model smooth curves that can be scaled indefinitely. They are defined by a set of control points, within his curve's shape being determined by the relative positions of these points. The fundamental principle of Bezier curves is based on the linear interpolation of points using Bernstein polynomials, which blend the control points at varying proportions depending on a parameter ' **t** ' that ranges from 0 to 1. As 't' changes, the interpolated points trace out the curve. This makes Bezier curves highly versatile and useful in computer graphics for creating intricate shapes and paths with a high degree of control and precision.

#### 1.2.1.1 Control points

When creating graphical representations of objects within a scene, it is essential to define their shapes and positions within an N-dimensional space. The characteristics of these objects—such as orientation, location, and the connections between points—are important. Take, for example, a sphere: its geometry is defined by its radius and the coordinates of its centre point in space. Similarly, in the construction of Bezier curves, we utilize a set of points known as control points.

#### 1.2.1.2 Parametric curves

In a way of description for Bezier curves, it is also called ‘Parametric curve’. Parametric curves are defined by an equation where a variable, known as a parameter, is represented by the letter 't'. This parameter 't' can range from negative to positive infinity. However, for the purposes of plotting a Bezier curve, which is only interested in the values of 't' from 0 to 1.

#### 1.2.1.3 Bernstein Polynomials

When describing parametric curves, Bernstein’s theorem of polynomials is a key component. Sergei Natanovich Bernstein devised the formula to provide a practical approach to demonstrate the Weierstrass approximation theorem(R.T Farouki, 2012). Bezier curve of degree n is represented by

(1.2.1.1)

The coefficient, , stands for the control points and the Bernstein basis polynomial determine the shape of the curve.

(1.2.1.2)

Where the terms are called binomial coefficients. They can be obtained using factorials (“ ! ”) with the following equation:

= (1.2.1.3)

For a given degree n, there are n+1 Bernstein polynomials. it is also possible to change the value of n which is the largest degree of any one term in that. Figure 7 illustrates Bezier curves of different degrees through a process of linear interpolations.

A diagram of a mathematical equation

Description automatically generated

Figure 7

Once looking at the cubic curve, it consists of connections with multiple linear interpolations(see figure 8).

A black text with red line

Description automatically generated

Figure 8

Namely, the shape of the curve is drawn by the combining the control points weighted by scaler coefficients.

#### 1.2.1.4 The De Casteljau's Algorithm

Through the 1.2.1 parts, we know each point on a Bezier curve can be expressed as the weighted sum of the control points, using Bernstein basis polynomials. The De Casteljau’s algorithm provides the practical method for these calculations unlike the mathematical description Bernstein polynomials represented. The below example will be helpful to understand so that Its operation is rather simple (see figure 9).

1. Begins with 4 control points P1~P4 (set initial control points).

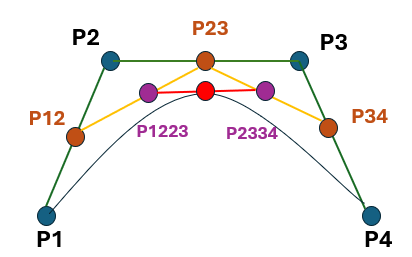


Figure 9

1. Find points on the line segments between control points (p12, P23, and P34) corresponding to a **t** value.
2. Use these new points to form new line segments (yellow lines).
3. Find the points (P1223, P2334) corresponding to **t** on these.
4. Repeat this process until only one point remains (the last point = the point on the curve at t).

In fact, The De casteljau’s work is numerically more stable than Bernstein’s theorem because of the property with recursive execution reduces the input errors to be negligible in the final output (Farouki and Rajan 1987, cited in M. Hanik 2024).

### 1.2.2 Bezier Surface (Mesh modelling)

For rendering dynamic shapes unlike ray-tracing basic primitives, the bezier surface patch formed by bezier curve(1.2.1) is one of them. It can be formed by moving curves through space and is finally defined by a grid of control points. In the mathematical representation of bezier surface, the 1.2.1.3 formula is used (see equation 1.2.2).

, 0, v1 (1.2.2)

As for finding a point on the surface, this equation 1.2.2 shows a double sum of control points and coefficients which forms a sum of bezier curves.

As the m, n values go by, it is possible to create bi-cubic, bi-quadratic, and bi-n surface can be calculated and created based on the two parameters, named **u** and **v** (see figure 10). However, the amount of required computation should be considered. In this project, the bicubic bezier surface will be focused on. Figure 11 illustrates the bicubic bezier surface for which total 4x4 control points(i.e. n=3 and m=3 in 1.2.2 equation).

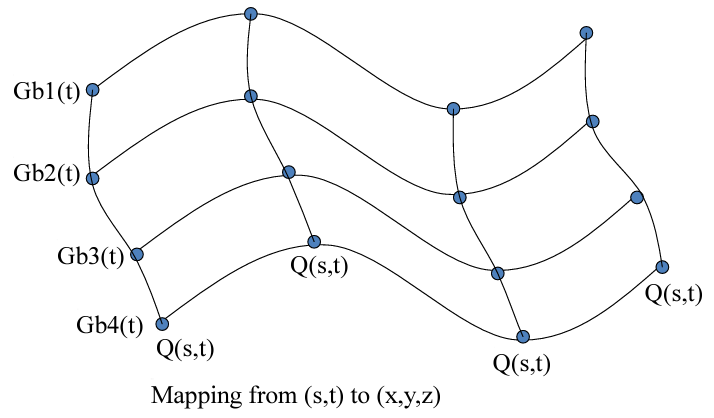


Figure 10 courtesy of Colorado state uni

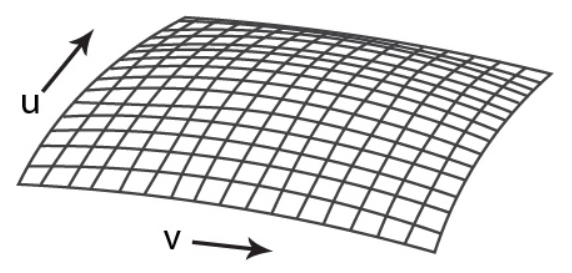


Figure 11

When it comes to the Bernstein polynomials representation (see 1.2.1.2 equation) on bicubic surface, it can be written like below.

,

# Chapter 2

### Methods

## 2.1 Development tools

To develop a Ray-tracer in which Bezier curve & surface can be rendered, the foundation of this project bases on Peter Shirley’s week series because his book is well defined to implement the ray tracing technique from scratch[ P.Shirley ].

The IDE(or text editor) used was Visual studio because it is suitable for debugging and testing C++ application in windows platform. The reason why C++ was selected among programming languages is to provide high performance on graphics with diverse graphics libraries like OpenGL, Vulkan and so on and to give me familiarity as computer graphic module last semester used it. To build and manage Visual Studio solution file(.sln), the open-source build system ‘Cmake’ was employed. As for version control, GitHub was used to track changes to code through Git and to host code.

## 2.2 Ray-tracer construction

Many ray tracing resources are designed to use several graphic API like OpenGL or Vulkan for high performance but, this project aims to research how the ray-tracer operates and renders shape based on curve. So, generating decent image with no API would be academically good approach in this project. The architecture of the project is basically designed to be modular and extensible, namely it bases on object-oriented programming. Let it be divided into key components: ray, camera, intersection, shading system and lighting. These will be connected to make a single ray tracer through 2.2.1.

### 2.2.1 Rendering pipeline

Prior to examining the technical methods, this section describes the sequential stages of the rendering process, which is pivotal transforming a 3D scene into a visually accurate 2D image. To facilitate efficient project execution, the flow chart below provides immediate visual identification of the ray-tracer’s components (see figure 12).

A diagram of a flowchart

Description automatically generated

Figure 12

### 2.2.2 Rendering an image

The core of the ray-tracing algorithm involves casting rays through pixels and determining the colour observed in the direction of those rays.

Scenario



1. Calculate the primary ray which goes from camera to pixel.
2. Select where that ray intersects in the way of nearest to the camera.
3. Compute a color for that intersection point.

**Ray generation**

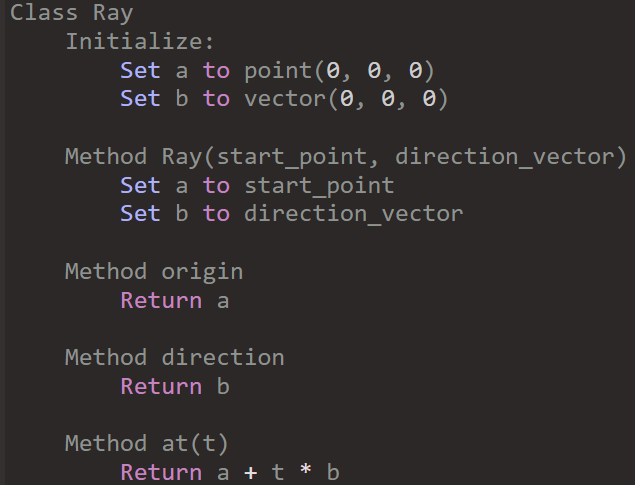


Figure 13

To represent a ray as well as the primary ray generated from camera, it is necessary to define ray class. Its way of implementation likewise follows the initial description in 1.1.1.1 Ray. Figure 13 pseudo code will be used in any circumstance relate to ray.

**Record for ray-intersection history**

Moreover, once a ray generated from camera reached any surface, ray-tracer requires the records of the ray’s information which contains intersection point, parameter value t at that point, and normal vector. Figure 14 shows the ‘Hit\_record’ class records what ray information when the ray hits a surface in the 3d space.

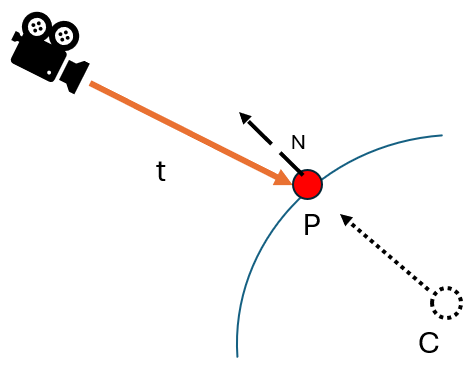
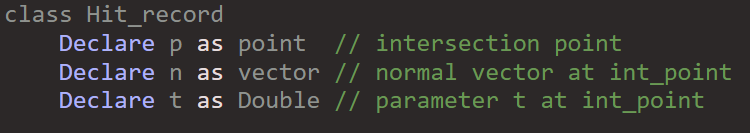


Figure 14

**Test for intersection**

When it comes to intersection ray with any object, there is a wide variety of methods to choose from because each shape has their own geometric properties. The simplest method involves working with sphere geometry, as its foundational principles have already been outlined in section 1.1.1.3 on intersections. However, this approach can also be adapted for rendering curves later.

To determine if a ray meets with sphere or not, an algorithm for testing any intersection is needed. Figure 15 pseudocode outlines the algorithm for determining whether a camera ray intersects with a sphere in a 3D space. Firstly, it begins with defining variables which are coefficients ‘a’, ‘b’, ‘c’ and ‘oc’ derived from the mathematical sphere equation and the ray’s properties in section 1.1.1.3. And then it uses the quadratic formula to solve for ‘t1’ and ‘t2’ which stand for the scalar values along the ray’s direction vector at which the ray intersects the sphere’s surface (see figure 16). Lastly, the result should be updated in ‘hit\_record‘ (derived from figure 15) when hitting any surfaces. Additionally, the scalar values t1 and t2 are vary based on the camera's position, which will also be addressed in camera model part later.

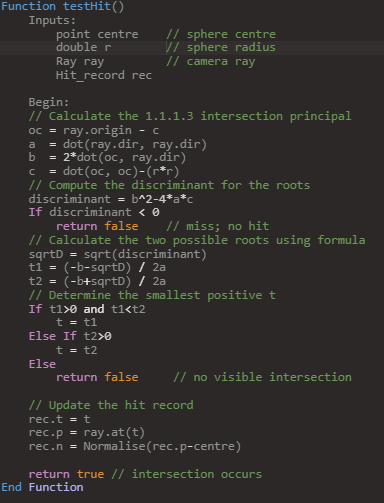


Figure 16

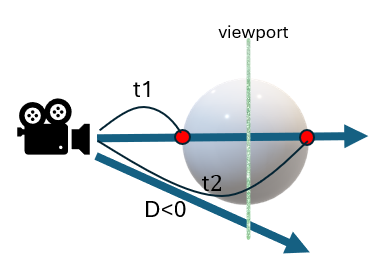


Figure 16

**Template for geometry**

The principal benefit of Object-Oriented Programming(OOP) design is the simplification of code structure. Thanks to the inheritance mechanism of C++, a plate class embodies this concept as a template for various geometric forms. All shapes in 3d space are under the range of ray hit. Therefore, ray-intersection method must be implemented in each geometry class. Figure 17 shows How an abstract function which is vacant is implemented differently by the derived class. By utilizing this mechanism, it also provides convenience in rendering multiple geometries within the camera's scene. The ‘Hittable\_list’ class, as illustrated in figure 18, acts as a container that simplifies the process of managing as well as rendering the geometries. Particularly, ‘testHit’ function iterates through each object in the list and checks for intersections to detect an intersection. According to P. Shirely’s book[?], this structure is not only efficient but also scalable, making it easy to add or remove objects dynamically without the need for individual intersection logic for each object.

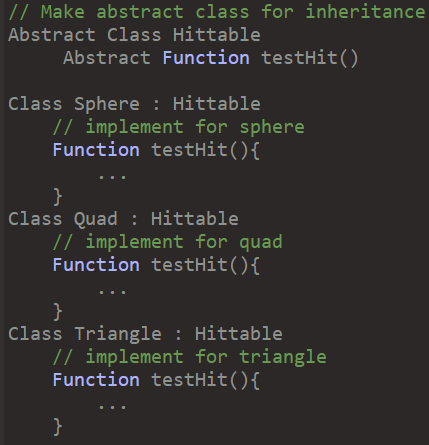


Figure 17

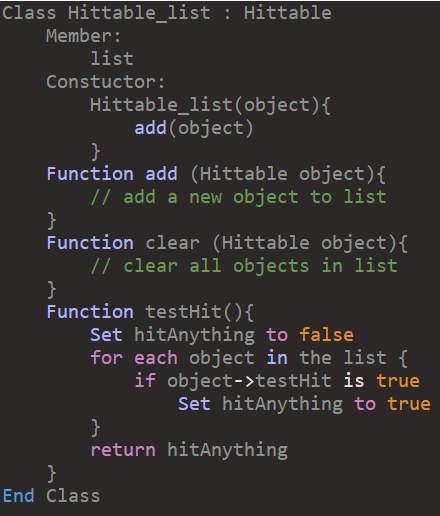


Figure 18

**Image format PPM**

Essentially, the image data is written to ‘std::cout’, which is C++’s standard output stream, and formatted specifically for the PPM image file format. Put simply, the format begins with a header that specifies the image encoding, the dimensions of the image in pixels, and the maximum colour value for each pixel. In the case of this ray-tracer, the identifier at first line is ‘P3’, which represents the PPM plain text format, followed by the image’s size, and ‘255’ to indicate the colour value for each colour channel(red, green, and blue) [Clemson]. Figure 19 shows the PPM format’s simplicity. On the left, there is an example of one snippet of a PPM file’s contents. On the right side, a graphical depiction of how pixels are structured in the image grid.

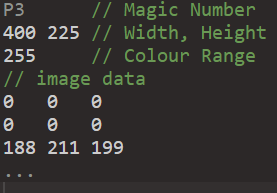
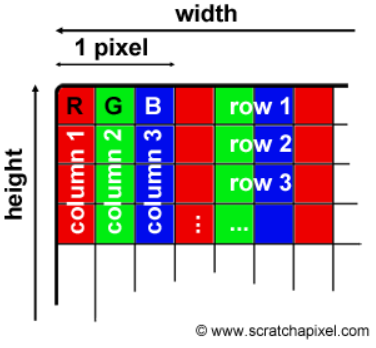


Figure 19



**Colour to pixel**

### Under this section, the ‘write\_color’ function in figure 20 translates the computed colour values into pixel data for the final image. Once each component colour extracted from the pixel

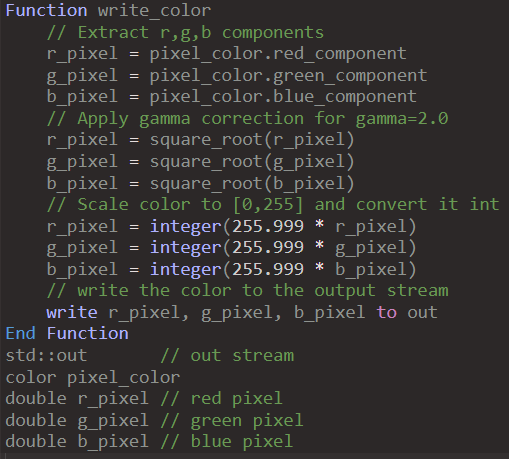


Figure 20

**Rendering**

### A computer screen shot of a program Description automatically generatedThe core of the ray tracing process is to render the scene it intends.

### 2.2.3 Camera model

By the figures presented so far, these are all projected by the camera at only a fixed -Z axis, so that the output is consistently a front-facing view without variation in angle or perspective. To overcome this limitation, controlling over camera is needed. So, positionable camera system is introduced

### 2.2.4 Lighting and shading

## 2.3 Bezier processing integration

### 2.3.1 Translating Bezier mathematics into code

### 2.3.2 Integration ray-tracer with Bezier curve

### 2.3.3 Fetch Implementation using curves

### 2.2.2 Rendering an image

### 2.2.3 Geometric intersection simulation

### 2.2.4 Lighting and shading

## 2.3 Ray-intersection with Bezier Geometry

### 2.3.1 The De Casteljau's Algorithm

### 2.3.2 Geometric intersection simulation

Without putting basic primitives

-> none of ray intersection -> noting is displayed

When sphere chain

When triangle chain

# Chapter 3 Improvement / Optimisation

## 3.1 Anti-aliasing

When an image rendered using pixels, a phenomenon looks like staircase or irregular colour pattern commonly occurs on the screen. Such effects we call “Aliasing”.

If the sampling rate(the number of rays cast) during ray tracing is not high enough, details in the scene may not be properly represented, resulting in aliasing. For reducing these, Anti-aliasing(AA) algorithms have been used in computer graphics.

### 3.1.1 Method



This technique is divided into four major categories.

High Resolution Display

* Pixel Phrasing
* Pre-Filtering
* Post Filtering

Supersample Antialiasing(SSAA)

Full-Scene Antialiasing (FSAA)

As the simplest Anti-Aliasing technique, this method

### 3.1.2 Result

### <Basic RT algorithm>

|  |  |  |
| --- | --- | --- |
| **Tracing a ray per each pixel** | **Staircasing** | **Noise** |
|  |  |  |

**<Anti-aliasing RT algorithm>**

|  |  |  |
| --- | --- | --- |
| **Multiple rays within a pixel** | **Smoothed** | **Low Noise** |
|  |  |  |

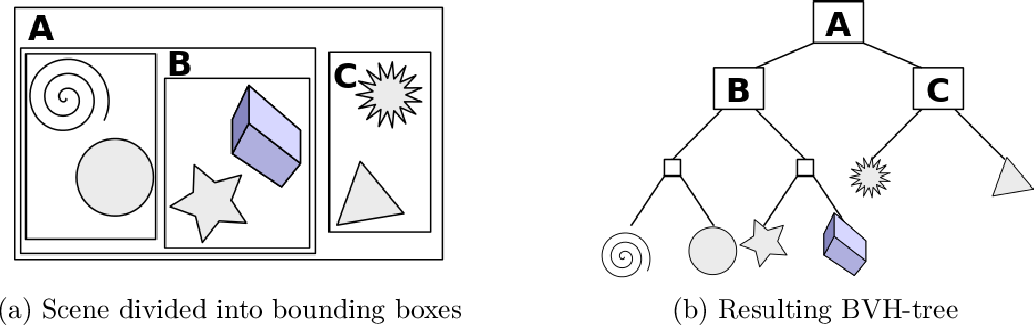
## 3.2 BVH

It's essential to trace a large of rays otherwise, the images may end up looking unrealistic with visible noise and artifacts, like a grainy or pixelated appearance(D. Meister et al., 2021). In a nutshell, the more rays traced, the better outcome we get. But the main constraint in a ray-tracer is the quantity of ray-object intersection, and the time is proportional to the number of objects(P. Shirley, 2016). This is where BVH **(Bounding Volume Hierarchy)**, comes into play, optimizing the ray-tracing process. The technique, as known for collision detection algorithms, is to produce clearer, more realistic images without the need for excessive computational resources.

The term “Bounding Volume(BV)” refers to physically any shape that can encapsulate any object. The idea behind this concept is to simplify object interactions by using BV or spatial decompositions(Dividing the space occupied by the object), thereby reducing the frequency of contact checks between objects (J.T. Klosowski et al. 1998). So, BVH is a tree(hierarchy) based structure that contains multiple hierarchically arranged bounding volumes.

### 3.2.1 Method

The key point is that BVH transforms the time-consuming linear search into a much faster logarithmic search, like the efficiency gains seen in binary search. Numerous rays are cast onto the same model in ray-intersection, sorting process is initially in demand for sublinear search. By dividing the objects, the process can be developed faster than dividing the space.





< 출처:https://www.semanticscholar.org/paper/Parallelization-of-BVH-and-BSP-on-the-GPU-Imre/961ca2300140ec357b8f878a534858b09bb84abc/figure/6>

A screen shot of a computer code

Description automatically generated

Once the program passes through this code block, it is time to enter the process of primitive division, namely, to find a volume that fully encloses all the objects.

#### A diagram of a rocket Description automatically generated3.2.1 .1 AABB

A diagram of a graph

Description automatically generatedFor dividing objects effectively, various shapes of bounding volume have been used in BVH algorithm. The image displays various types of bounding volumes (see Figure ?). In this context, the AABB (Axis-Aligned Bounding Box) will be employed because it is particularly advantageous for its computational efficiency and minimal memory usage. The method uses a concept known as 'slabs,' with an n-dimensional AABB being the intersection of n such axis-aligned intervals(see Figure ? ).

**Figure ?**

Figure ? indicates an 2D interval at (x0,x1)(y0y1)

#### 3.2.1 .2 A bounding box for an object

### 3.2.2 Hierarchy of boxes

### 3.2.4 Result

# Chapter 4 Results

<Results, evaluation (including user evaluation) *etc*. should be described in one or more chapters. See the `Results and Discussion' criterion in the mark scheme for the sorts of material that may be included here.>

# Chapter 4 Discussion

<Everything that comes under the `Results and Discussion' criterion in the mark scheme that has not been addressed in an earlier chapter should be included in this final chapter. The following section headings are suggestions only.>

## 4.1 Conclusions

<Text in 11-point size and 1.5 line spacing.>

## 4.2 Ideas for future work

<Text in 11-point size and 1.5 line spacing.>

# List of References

*<It is expected that the list would reflect the breadth and depth of scholarly research undertaken by the student during the course of the project.>*

P. Shirley, 2016, Ray Tracing in One Weekend,

P. Shirley, 2016, Ray Tracing: the Next Week,

K. Nakamaru and Y. Ohno, 2002, Ray Tracing for Curves Primitive

NVIDA Ray-Tracing <https://developer.nvidia.com/discover/ray-tracing>

J. T. Whitted of NVIDA, 2018, Why Ray Tracing Is a Natural Choice for Global Illumination, A Ray-Tracing Pioneer Explains How He Stumbled into Global Illumination.

N. DAMERA-VENKATA and N. L. CHANG, 2009, Display Super-sampling, DOI : <http://doi.acm.org/10.1145/1477926.1477935>

D. Meister et al., 2021, A Survey on Bounding Volume Hierarchies for Ray Tracing, Computer graphic forum, Vol: 40, Issue: 2, DOI: <https://doi.org/10.1111/cgf.142662>

J.T. Klosowski et al., 1998, Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs, *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, Vol: 4, Issue 1, DOI: [10.1109/2945.675649](https://doi.org/10.1109/2945.675649)

R. T. Farouki, 2012, The Bernstein polynomial basis: a centennial retrospective,

M. Hanik and E. Nava-Yazdani, 2024, De Casteljau’s Algorithm in Geometric Data Analysis: Theory and Application, <https://arxiv.org/html/2402.07550v1>

Clemson University, chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://people.computing.clemson.edu/~dhouse/courses/405/notes/ppm-files.pdf

**image reference**

Pixel - <https://www.scratchapixel.com/lessons/digital-imaging/simple-image-manipulations/reading-writing-images.html>

# Appendix A Self-appraisal

<This appendix must contain everything covered under the ’self-appraisal’ criterion in the mark scheme. Although there is no length limit for this section, 2-4 pages will normally be suﬃcient. The format of this section is not prescribed, but you may like to consider the following sections and subsections.>

## A.1 Critical self-evaluation

## A.2 Personal reﬂection and lessons learned

## A.3 Legal, social, ethical and professional issues

<Refer to each of these issues in turn. If one or more is not relevant to your project, you should still explain *why* you think it was not relevant.>

### A.3.1 Legal issues

<Discussion of legal issues>

### A.3.2 Social issues

### <Discussion of social issues>

### A.3.3 Ethical issues

### <Discussion of ethical issues>

### A.3.4 Professional issues

<Discussion of professional Issues>

# Appendix B External Materials

<This appendix should provide a brief record of materials used in the solution that are not the student's own work. Such materials might be pieces of codes made available from a research group/company or from the internet, datasets prepared by external users or any preliminary materials/drafts/notes provided by a supervisor. It should be clear what was used as ready-made components and what was developed as part of the project. This appendix should be included even if no external materials were used, in which case a statement to that effect is all that is required.>