

Uthara Thelagar

Tanmay Patil

Himanshu Joshi

Anil Ramakrishna

Srikanth Madhava

Infinity Ray Tracer

# TABLE OF CONTENTS

Chapter 1 Introduction #

Section 1.1 Motivation #

Section 1.2 Objectives #

Section 1.3 Task Breakdown #

Chapter 2 ARCHITECTURE #

Section 2.1Architecture #

Section 2.1.1Graphics Pipeline #

Section 2.1.2 SW Architecture #

Chapter 3 IMPLEMENTATION #

Section 3.1 BareBones Tracer #

Section 3.2 Object Parsing #

Section 3.3 Ray Casting #

Section 3.4 Lighting #

Section 3.5 Reflection #

Section 3.6 Refraction #

Section 3.7 Procedural texturing #

Chapter 4 CHALLENGES #

Section 4.1Results summary #

Section 4.2Conclusions #

Section 4.3 Next Steps #

Chapter 5 Conclusion #

Section 4.1Results summary #

Section 4.2Conclusions #

Section 4.3 Next Steps #

BIBLIOGRAPHY #



# 1. Introduction

In this section we review our motivations for choosing Ray Tracing and our project objectives.

## 1.1 *Motivation*

The CS580 course has been about growth – both as software engineers and graphics programmers, with the distinction between the two constantly blurring with each project. In a few weeks we were able build a renderer that took input coordinates in the model space and rendered them in screen space with different effects such as texturing, procedural texturing, anti-aliasing etc. Inspired by the Professor’s enthusiasm and experience we were very motivated to explore several topics for the project.

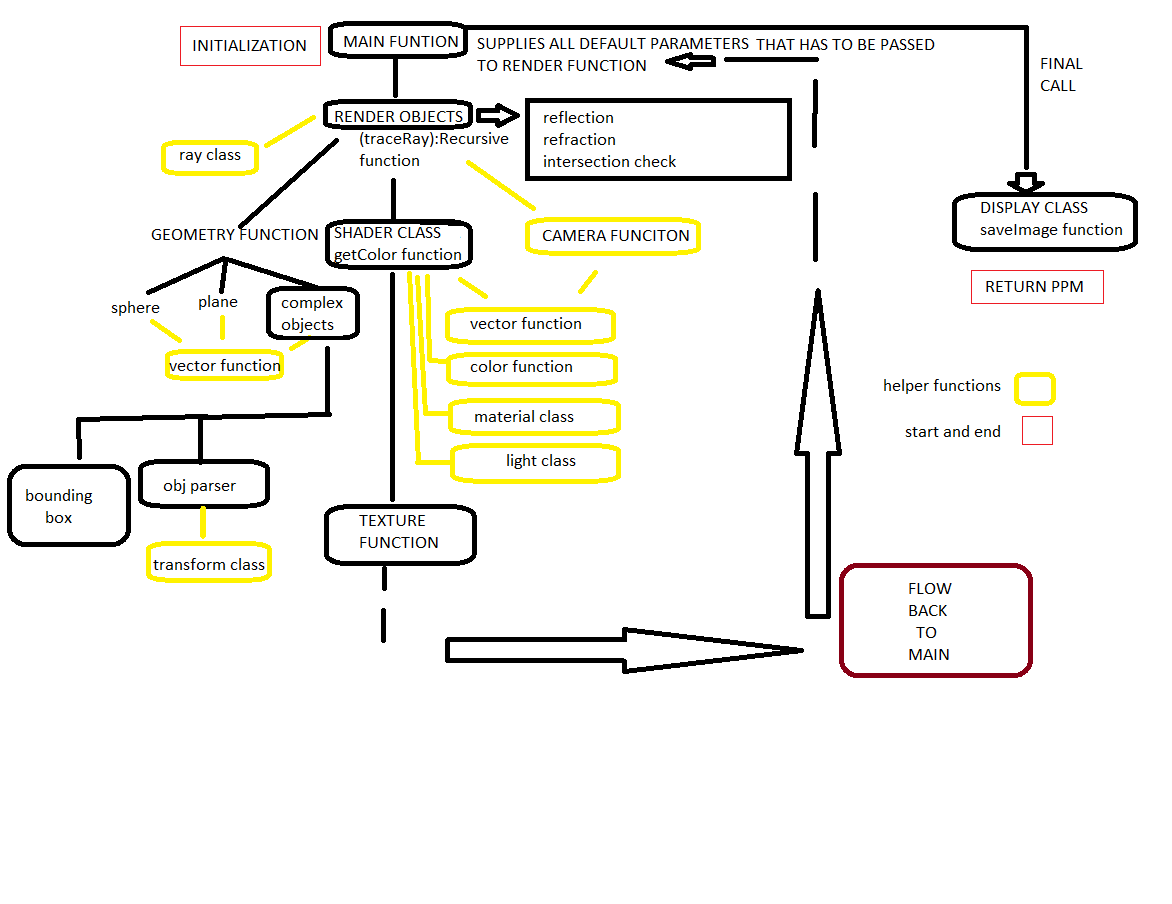
We considered Interactive animation, Cloth simulation, Procedural Animation and several other ideas for the project. Finally, we decided to go with building a ray tracer since we were inspired by its simple yet powerful concept. We wanted to understand how to construct this renderer from scratch with the possibility to explore any apparent bottlenecks. We were also keen on adding another fundamental algorithm to our arsenal of 3d computer rendering techniques.

## 1.2 *Objectives*

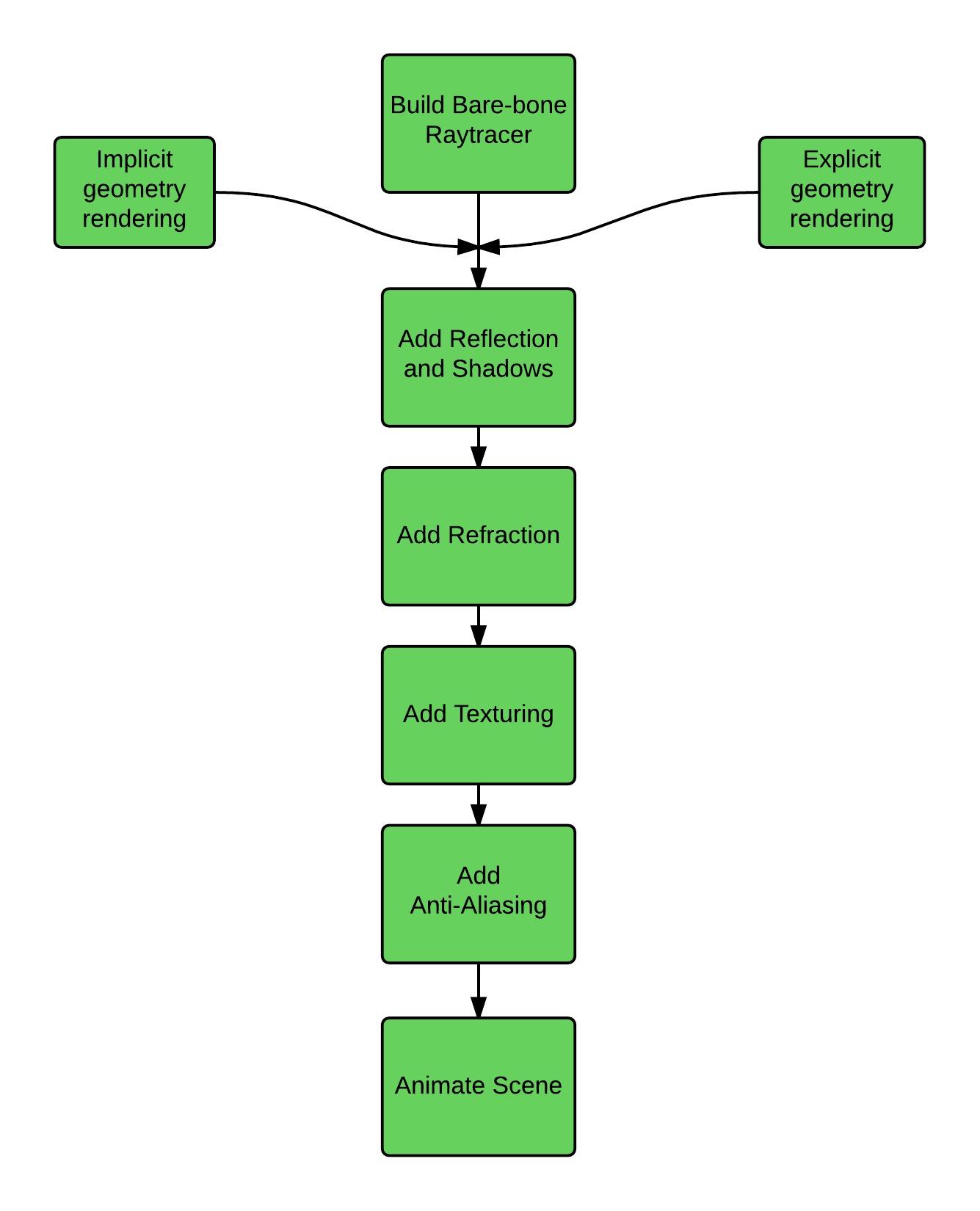
* Develop a ray tracing library from scratch
  + Understand the computational aspects of Ray Tracing
    - Is there room for optimization?
* Add Shading, Reflection, Refraction, Anti-Aliasing and Texturing.
* Create a brief animation of our scene.
* Work in a team, with a distributed asynchronous workflow.
* Support multiple platforms for development to accommodate different members' needs.

# 2 ORGANIZATION

## 2.1 *SOFTWARE ARCHITECTURE*



## 2.2 *DEVELOPMENT CYCLE*



## 2.3 *Task BreakDown*

This project is unique both in its scope and its ambitions. Some of us were interested in understanding the graphics pipeline while others in working with the computational challenges that it entailed. As requested, we are providing a tentative breakdown of tasks below.

|  |  |
| --- | --- |
| Name | Tasks |
| Uthara | Main concept, design, implementation, skeleton tracer, shadows, shading, reflection, look development. Domain architect. |
| Himanshu | SW Architecture, Skeleton Tracer, Shader, Multiple Object Intersection, Github, Cross Platform Support, Documentation. |
| Tanmay | Explicitly Modeled Object Parser, OctTree Optimization for complex object rendering, s-t and Normal Interpolation. |
| Anil | Animation, Refraction, Reflection, Shadow calculation, Debugging, Code Refinement, Documentation, Report generation. |
| Srikanth | Texture Mapping (Explicit and Procedural), Anti-Aliasing, Website, Animation, Report generation. |

# 3 Implementation Details

## 3.1 *Barebone tracer*

We built a basic ray tracing renderer by following the tutorials from . This renderer could handle basic implicitly modeled objects such as a sphere, plane, etc. The points of ray-intersection with these objects were calculated mathematically using standard theorems in 3d geometry and vector algebra . We used Phong shading for our color calculations. Our shadow calculations also follow directly from .

## 3.2 *Object Parsing*

Our next step was to add an object parsing library that could read any obj file containing model space vertices of polygons that make up the complex 3d object. We used the object parser developed by Andrew Goodney and slightly extended it to transform all the input polygons to camera space, using the transformation matrices we learnt from the class home works.

The rendering time with the basic parser however was very limiting; it took more than an hour to render the Utah teapot on our laptops. To avoid this, we added an very efficient optimization algorithm which creates a bounding box for each complex object and also recursively creates eight smaller boxes to occupy the eight 3d quadrants within the bounding box. In the innermost level of recursion, we index all the triangles that lie within each small bounding box. This is similar to creating a large number of positional hash for each triangle in the object; instead of testing for ray intersection with each triangles in the polygon, we only test the triangles that belong to the innermost bounding box that contains the ray. Using this optimization, the rendering time reduced from 1+ hour to ~ 5 minutes.

## 3.3 *Reflection*

We added reflection following the equations from and . For each ray that is incident on an object, we compute the reflected ray's direction and recursively track this ray to get the color for the original ray. To keep the computational overhead to a minimum, we limit the depth of ray interaction through reflection or refraction to 3.

## 3.4 *Refraction*

We currently perform refraction only on "glass" objects (refractive index ~1.5). For refracted ray calculation, we followed equations from and . For each object on which a light ray is incident, we computed the color due to reflection as well as refraction. These are then combined using Schlick's approximation to the Fresnel equations . We limit refraction only to simple implicitly modeled objects. Since time was limited, we could not get refraction working with implicitly modeled complex objects but we do note that the underlying mechanism is very similar to implicit objects such as spheres: Calculate two points of intersection for a ray with the object, perform refraction at each of these and combine results to get the final color.

## 3.5 *Texturing*

UV mapping for sphere was found using , using this various texture was mapped on to the sphere object in the scene. For any point P on the sphere, calculate d, that being the unit vector from P to the sphere's origin. Assuming that the sphere's poles are aligned with the Y axis, UV coordinates in the range [0, 1] can then be calculated as follows:  
  
u = 0.5 + arctan2(d\_z,d\_x)/2\*pi  
v = 0.5 - arcsin(d\_y)/pi  
  
U and V calculated is mapped to texture image to calculate the pixel.

### 3.5.1 Procedural Texturing

We added procedural texturing to the sphere by adding normally distributed and skewed 2d noise .

## 3.6 *Anti-aliasing*

We planned to do anti-aliasing using Super-sampling, since it is one of the simplest discrete methods. It involves casting more than one regularly spaced sample per pixel and using the average of the results for the pixel intensity. To render the pixel, primary rays are cast through each of the indicated sample points. The intensity value of the pixel is the average of all the samples. Because of insufficient time we were not able to implement them.

## 3.7 *Scene Setup*

In our final scene, we show three complex objects read using the object parser: A pawn and two pillars as shown in the figure below. We also have multiple spheres with textures (regular and procedural) and two transparent spheres to demonstrate refraction.

## 3.8 *ANIMATION*

Finally, we animated our scene by generating several output images while gradually moving the camera the transparent spheres. The output images were combined using ffmpeg to create an mp4 video file that we report.

# 4 Challenges

## 4.1 Choosing the right Problem

As outlined in Section 1, we looked at quite a few problems, including noise generation, generating interesting geometry, as demonstrated in <http://paulbourke.net>, or syntactic L-trees as shown on <http://en.wikipedia.org/wiki/L-system>. Although these problems seemed beautiful, in the end what drove our decision to choose Ray Tracing was the simple elegance of the intuition behind Ray tracing. This, taken with the fact that ray tracing is typically slow, we wanted to investigate potential bottlenecks to exploit.

## 4.2 Ray Tracing Objects from OBJ files

Most of the ray tracing algorithms we found dealt with sending rays through geometry – such as spheres or planes – we wanted to efficiently render a scene using triangles obtained from an OBJ parser.<<@todo add more details about optimizations>>

## 4.2 Scheduling with multiple classes

Coordination between multiple members with varying schedules was a problem which we anticipated. We dealt with this by using a distributed development model enabled via the asynchronous git workflow. Additionally, the core ray tracer was developed with the SW architecture principles of mechanism & component reuse to reduce coupling. Thus, OBJ parser, Reflection and Refraction could be developed concurrently, once the skeleton application and relevant API were ready.

## 4.3 Development Environment

Due to the various platforms people were using, and the specificities of the way the C++ compiler behaves in various environments, we faced an unanticipated challenge. The issue was several standard C defines from the math library were not usable in VC++!

INFINITY, M\_PI were unavailable on VC++ while abs was unavailable on the OS X LLVM compiler. Finally, the board support package (bsp) which contained default initializations for the tracer would were not compilable in VC++ but worked as expected in both the OS X XCode’s LLVM compiler and the gcc compiler under Cygwin and OS X. This set us back by a day or so.

## 4.4 Visual Studio Development Environment

Perhaps the most troubling and time consuming bug was an issue with the file writing routine on Windows. When saving the output in .tga file, the output was coming out distorted (<<todo>> paste examples here).

Initially, we debugged this from the angle of an issue with the renderer main loop itself. But this was not showing up on the OS X and Cygwin environments. After some debugging, we realized the file had more bytes than expected. We could not debug why this was happening, we were able to verify the input to fprintf was same in both the working and non-working cases. We added the functionality to generate .ppm files and this issue went away on windows.

# 5 NEXT STEPS

Apart from solving the challenges we mentioned above, there are several important additions we can make to this system

* Use GPU/CUDA for faster calculations and rendering.
* Add refraction to complex objects.

# 6 Conclusion

We built a fully functional 3d rendering framework using ray tracing from scratch with several important features such as shadows, reflection, refraction, textures, complex object rendering, etc. This framework can be used a useful starting point for anyone who wishes to build more advanced features such as quick, interactive animation, etc. This was also an excellent experience for all of us since we learnt several core algorithms in ray-tracing and also gave us an exposure to several software engineering processes.

# Bibliography

1. http://www.scratchapixel.com/old/lessons/3d-basic-lessons/lesson-1-writing-a-simple-raytracer/implementing-the-raytracing-algorithm/. *scratchapixel.com.* [Online]

2. http://www.scratchapixel.com/old/lessons/3d-basic-lessons/lesson-7-intersecting-simple-shapes/. [Online]

3. http://en.wikipedia.org/wiki/Phong\_reflection\_model. [Online]

4. http://www-bcf.usc.edu/~saty/edu/courses/CS580/f14/hw/FinalProject/misc/OBJ\_Parsers/parser\_Andrew/index.html. [Online]

5. http://tavianator.com/2011/05/fast-branchless-raybounding-box-intersections/. [Online]

6. http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter3.htm. [Online]

7. http://www.cs.jhu.edu/~cohen/RendTech99/Lectures/Ray\_Tracing.bw.pdf. [Online]

8. http://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection\_refraction.pdf. [Online]

9. http://www.flipcode.com/archives/reflection\_transmission.pdf. [Online]

10. http://en.wikipedia.org/wiki/Schlick%27s\_approximation. [Online]

11. http://en.wikipedia.org/wiki/Fresnel\_equations. [Online]