

*Take Infinity*

Uthara Thelagar

Tanmay Patil

Himanshu Joshi

Anil Ramakrishna

Srikanth Madhava

*CS580*

Infinity Ray Tracer

# TABLE OF CONTENTS

Chapter 1 Introduction #

Section 1.1 Motivation #

Section 1.2 Objectives #

Section 1.3 Task Breakdown #

Chapter 2 Organization #

Section 2.1 Architecture #

Section 2.1.1 Graphics Pipeline #

Section 2.1.2 SW Architecture #

Chapter 3 Detail #

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 Conclusion #

Section 4.1 Results summary #

Section 4.2 Conclusions #

Section 4.3 Next Steps #



# 1 Introduction

In this section we review our motivations for choosing Ray Tracing and our objectives. This is wrapped up with a breakdown of tasks assigned to individual members.

## 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 from model space and render in screen space with different effects such as texturing, procedural texturing, anti-aliasing etc. Coupled with the professor’s enthusiasm and experience we were very motivated to explore various topics for the project.

For this project, we considered L trees, Geometry from Paul Burke’s website and various youtube videos on gears, character animation etc. Finally, we found inspiration in the simple elegance of the intuition behind the ray tracer. We named our team ‘*Take Infinity*’ due to the infinitely many recursive possibilities that arise out of this approach to rendering. We wanted to understand how to construct this renderer from scratch with the possibility to explore any apparent bottlenecks.

## 1.2 Objecttives

With the above motivations, we had the following objectives:

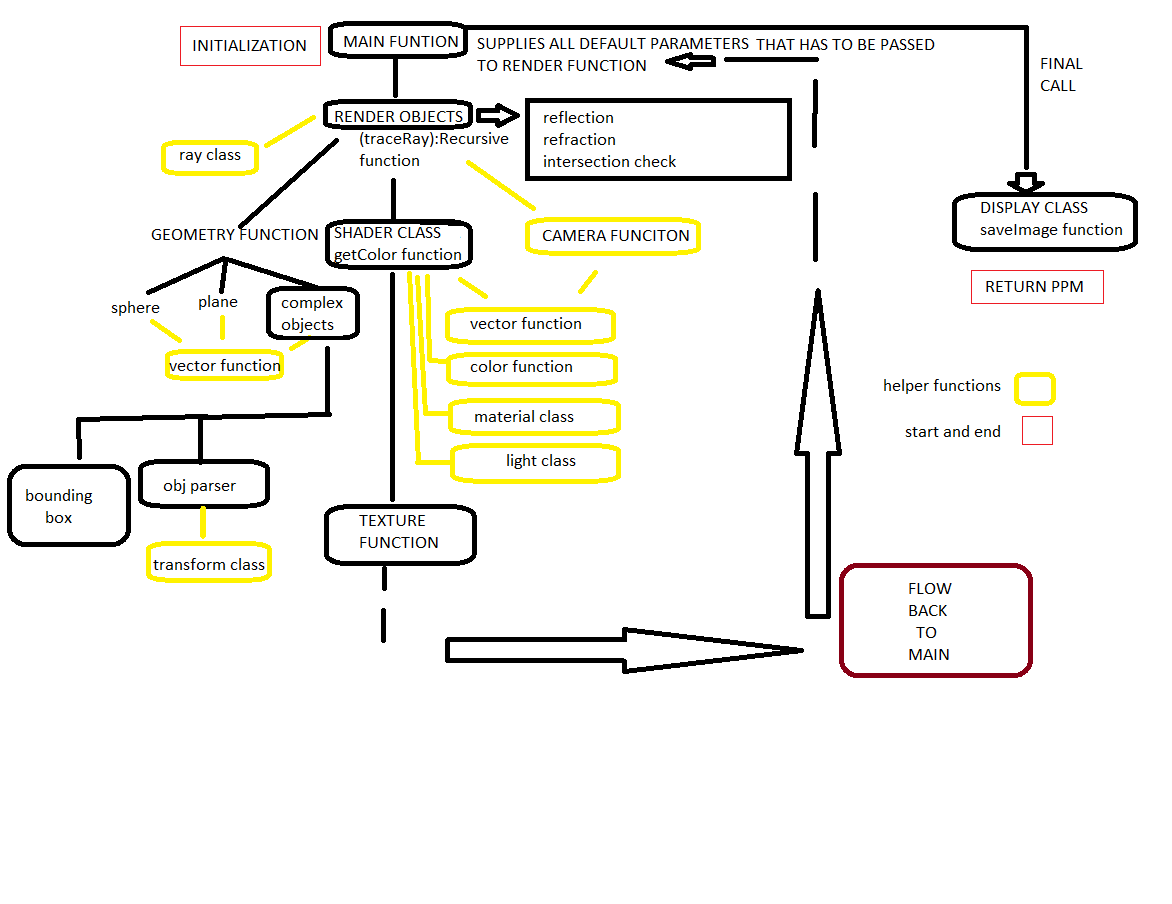
* Develop the ray tracer pipeline from scratch
  + Understand the computational aspects of Ray Tracing
    - Is there room for optimization?
* Adding features such as reflection, refraction and anti-aliasing.
* Work in a team, with a distributed asynchronous workflow.
* Support multiple platforms for development to accommodate different.
* Support multiple output formats.

## 1.3 Task BreakDown

We understand this project was unique both in its scope and its ambitions with different folks interested in different aspects of the problem. While someone was interested in understanding the graphics pipeline whereas someone else was interested in understanding how the computation is distributed to understand avenues for improvement. As requested, we are providing a tentative breakdown of tasks, listed in order of flow of modules.

|  |  |
| --- | --- |
| Name | Tasks |
| Uthara | Main concept, design, implementation, skeleton tracer, shadows, shading, reflection, look development. Domain architect. |
| Himanshu | SW Architecture, including skeleton tracer, shader, multiple object intersection, github, cross platform support, Documentation. |
| Tanmay | Object Parser, OctTree optimization, s-t and normal interpolation |
| Anil | Refraction, debugging, code refinement |
| Srikantha | Textures, anti-aliasing, website |

# 2 Architecture



## 2.1 Rationale

### 2.1.1 Graphics Pipeline

### 2.1.2 SW Architecture

# 3 Details

## 3.1 Barebones tracer

## 3.2 Object Parsing

## 3.3 Ray casting

## 3.4 Shading

## 3.5 Reflection

## 3.6 Refraction

## 3.7 Texturing

### 3.7.1 Simplex Noise

## 3.8 Anti-aliasing

## 3.9 Scene Setup

# 4 Challenges Faced

## 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 Conclusion

## 4.1 Results Summary

## 4.2 Next steps

This project has multiple possible avenues it can branch. Here we are listing the ones we feel are important

The quote is bold and distinctive. The galleries include items that are designed to coordinate with the overall look of your document.

# References

**First reference.**

Scratch Pixel tutorial on Noise http://www.scratchapixel.com/old/lessons/3d-advanced-lessons/noise-part-1/pattern-examples/

**Additional references.**

Ray Casting calculation

http://www.macwright.org/literate-raytracer/

Scratch Pixel tutorial on Ray Tracer http://www.scratchapixel.com/old/lessons/3d-basic-lessons/

Geometry calculations for Sphere and Plane Andrew Glassner Intro to ray tracing

Reflection http://www.cs.jhu.edu/~cohen/RendTech99/Lectures/Ray\_Tracing.bw.pdf

Refraction

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