Final Project Revised Proposal: Extended Raytracer

Date: July 11, 2006
Name: Mike Jutan
Userid: mjljutan
ID: 20079294
Section: 002
Professor: Gladimir Baranoski

Contents

Final Project Proposal: Extended Raytracer

1 Proposal

1.1 Purpose

My passion in Computer Science has been centered on the fusion of Fine Arts and Computer Science for just about as long as I can remember.

As a Computer Science student at Waterloo, I have enrolled in many Fine Arts and Film classes to better my understanding of the Fine Arts world and to further develop my artistic skills. At the time of writing this proposal, I am enrolled in an upper-year Fine Arts independent study course, where I am being taught in a private one-on-one fashion. In this course, I am learning concepts of colour theory, scene composition, reflection and refraction and I am also getting a chance to do lots of modeling, texture mapping and other techniques in Maya.

This explanation I hope provides the reader with a context for my project, and why I have chosen to extend my Raytracer with some Fine Arts techniques and ideas in mind.

I plan to take some of the Fine Arts skills that I am developing, and apply them to the Final Scene for my Extended Raytracer Project. The technical requirements for my Raytracer are based quite closely on my artistic goals for this Project, and thus the technical requirements were a direct consequence of solidifying my artistic goals.

Pixar's creative genius John Lasseter is someone who I really look up to, and as he said in his famous quote: *"The art challenges technology and the technology inspires the art."*

I hope to take John Lasseter's words to heart on this Project and really make some art that doesn't just look like it's been obviously computer-generated. Also, I'd like to figure out some pretty tough technical details in the extremely limited 2.5 week time frame we have for this project.

1.2 Topics

- Raytracing speed enhancements.
- Advanced Raytracing and light interaction techniques.
- Using Fine Arts concepts to create a Final Scene.

1.3 Statement

I plan to model some beer glasses (pint glasses) and perhaps other objects in Maya, a 3D modeling package. I will then export these items as `Wavefront .OBJ` files, and import them into my Raytracer. This will require some extensions to the supplied `OBJ` importer, to allow for vertex normals and perhaps other `.OBJ` requirements (such as texture coordinates.)

I plan to have a Final Scene with multiple mesh objects on a wooden table, with a picture frame in the background, and other common household items on the table. I would like to show what appears to be a random assortment of items on a table, something that is so commonplace in everyday living, that many people would simply ignore these items completely. My focus in my Fine Arts courses it to suggest that normal, everyday items can have a lot of beauty to them, if the small details are observed. For instance, light reflection and refraction in glasses is very beautiful, and is quite often completely ignored due to it's mundane, everyday nature. I hope to convince the observer that these types of light interactions with "everyday" objects have a much deeper level of beauty than most people would initially expect.

To implement a Raytracer capable of such goals, I will need several specific features. The obvious one is of course a decent glass simulation, which I will achieve with reflection and refraction, possibly using the Fresnel method to achieve a more realistic effect. To setup the indoor table scene, I will need interesting solid (procedural) textures, such as wood grain. I plan to try several different functions, such as Perlin noise, to get a decent texture simulation. This kind of indoor scene suggests the need for a hanging picture frame, and thus will require texture mapping. In the hopes of further increasing the believability and effect of the Final image, I also feel that soft shadows, phong shading and several other graphics techniques are necessary to achieve my artistic vision.

I believe that this is a very interesting concept and I hope that I have convinced the reader that my ideas are worthwhile. In terms of the complexity of this Project, I believe that many of the goals that I have made are computationally expensive, and this implies the need for some speed-up Algorithms, which I will implement in the form of spatial subdivisions, and by running my Raytracer as a multi-process computer program.

I hope to learn many new Graphics techniques by implementing this Project, and I hope to better understand the Physics of light interaction which I believe will follow from my implementation of reflection and refraction. More importantly, I want to have fun creating this Raytracer! While the overall learning experience is perhaps the ultimate benefit of this project, I hope to appreciate the journey I take to achieve this knowledge. I hope that I have such a great time and learn so much by developing this extended Raytracer, that I want to continue working on my Raytracer after finishing CS 488.

1.4 Goals

- Raytracer speed enhancements to allow more complex scenes to be rendered.
- Effective glass simulation, using Reflection and Refraction.
- Using Fine Arts techniques to create an artistic Final Scene with good composition and object placement.

1.5 Communication

The basic flow of information is as follows.

Input :
`Wavefront .OBJ` file(s), describing polygonal mesh objects. A scene description file, written in `LUA`, which uses an `OBJ` importer to extract the necessary data from the given `OBJ` file(s).

Interaction :
The Raytracer will not take any user input after the Raytracing process has been started.

Output :
A rendered image of the specified size, meeting the scene specification as supplied by the given `LUA` file.

1.6 Modules

This project will be based on my implementation of Assignment 4, and therefore my code organization will be similar to A4, where matching `*.hpp` and `*.cpp` files will specify and implement a class.

There will, although, be some changes from the setup in A4 for efficiency and ease-of-implementation issues.

For instance, instead of passing around the STL containers throughout the A4 render method, I will likely set these objects as member variables to reduce any unnecessary passing of pointers and copying of objects. I am also planning some extensions for triangle-only mesh objects. Since I am planning to have several mesh objects in my final scene, I will create triangle-specific intersection code that is separate from the generic polygon-intersection code. This will allow me to use a faster, more efficient method for ray-triangle intersections, since I will be doing an astronomical amount of these intersections and I would like this code to be as efficient and as fast as possible as I can make it in the given time frame.

Command line options will also be added to the Raytracer to allow for the user to specify certain parameters such as Anti-aliasing amounts, recursion depths etc.

1.7 Technical Outline

While the artistic vision of my Project is well-defined, the method that I am going to use to achieve a kind of "realistic" look in my Final Scene in not very well-defined. Therefore, I had to come up with a list of objectives that I think will be achievable in the given time frame, and will maximize the effectiveness of my final scene, while still maintaining a level of programming complexity that is not overboard for the given time frame.

I will now outline the technical aspects of my proposed objective list and discuss some of the practical issues I will need to consider when implementing these objectives.

1.7.1 Uniform Spatial Subdivisions (Grid Creation, WCS Axis-Aligned Bbox Transformations and Voxel Traversal)

This is perhaps my hardest objective, but will also be the most important. I am planning to implement this objective first, so that subsequent objectives benefit from the speed enhancement from this objective.

I am planning on taking this objective in 3 sub-steps: Grid Creation, WCS Axis-Aligned Bbox Transformations and Voxel Traversal.

Firstly, I will need to create a 3D grid of uniform-sized cube objects, which will be axis-aligned and will setup the initial grid structure.

Secondly, I will need to write code to determine which primitives are contained within which Voxel (or set of Voxels), and an association must be made so that each Spatial Subdivision Box contains a list of the Primitives that are contained within it. This will be done by adding bounding boxes to each primitive object, and then converting these bounding boxes from an AABB (Axis Aligned Bounding Box) in Model Coordinate Space to an OBB (Oriented Bounding Box) in World Space, and finally to an AABB in World Space. This method will also be used to calculate the extent of a scene so as to draw a bounding box around the entire scene's contents.

Lastly, I must implement a 3D version of the DDA algorithm, as described in the paper *"A Fast Voxel Traversal Algorithm for Ray Tracing"* by Amanatides, J., and Woo, A. This will allow my Rays to take a path through the Spatial Subdivision Boxes, and will greatly increase the performance and efficiency of my Raytracer.

1.7.2 Adaptive Anti-aliasing

Rather than implementing straightforward Supersampling, I am going to use an Adaptive method which operates in the following manner:
First, the image will be fully Raytraced with only 1 ray per pixel. Following this, a method will iterate over these pixels and locate "bad pixels." Bad pixels are defined to be pixels that differ from their surrounding (neighbouring) pixels by more than a certain threshold.
After making a set of bad pixels, the Raytracer will then Raytrace these pixels again with multiple rays. This way I can concentrate the computational power only on the pixels that "need" Anti-aliasing more than other pixels, and not waste precious computational time by sending multiple rays through pixels that do not need this.
I will likely make this Anti-aliasing a multi-pass process, and it will continue until there are no more "bad pixels" or after reaching a specified number of passes.

1.7.3 Refraction

To implement Refraction, I will start with the method specified in class and in the CS 488 course notes. I will send a Transmitted vector through the surface using Snell's Law as discussed in class. As an extension to this, if I have time, I would like to combine Reflection and Refraction together by implementing the Fresnel equations to better simulate a dielectric

glass material.

As mentioned in *Physically Based Rendering, pp. 419-420*, Pharr explains that if we make the assumption that light is unpolarized, the Fresnel Reflectance equations are simplified to the average of the squares of the parallel and perpendicular polarization terms.

A close approximation to the Fresnel reflectance formula for Fresnel dielectric materials is:

$$r_{\parallel} = \frac{\eta_t \cos\theta_i - \eta_i \cos\theta_t}{\eta_t \cos\theta_i + \eta_i \cos\theta_t}$$
$$r_{\perp} = \frac{\eta_i \cos\theta_i - \eta_t \cos\theta_t}{\eta_i \cos\theta_i + \eta_t \cos\theta_t}$$

where $r_{\parallel}$ is the Fresnel reflectance for parallel polarized light and $r_{\perp}$ is the reflectance for perpendicular polarized light. The Fresnel reflectance for unpolarized light is described in detail in the Pharr book.

This will require a new command (or a new parameter) added to the material equations to specify the refraction properties of the surface (including refractive coefficient.)

1.7.4 Phong Model using Vertex Normal Interpolation on Triangles

To create smoother and more effective mesh rendering, I will require the use of vertex normal interpolation. I will be using Barycentric Coordinates to calculate the interpolated vertex normal N, and using this as the surface normal for the shading calculations as opposed to the straight cross-product of the triangle sides which I will use for the naive approach in A4.

This will require some changes to the commands to allow the user to specify a model with Vertex Normals. I will also need to write or find an extension to the supplied `OBJ` importing code that reads Vertex Normals from the given `OBJ` file and outputs these to the Mesh objects in my Raytracer.

1.7.5 Texture Mapping

I will be creating a straightforward texture mapping system as described in class and with help from the Pharr book and other references.

I will be creating normalized UV coordinates for a flat, planar, rectangular surface which will be a picture frame in my Final Scene. These calculated (u,v) coordinates will be used as an index into a `.png` image file, which I will then map directly onto the flat, planar surface.

1.7.6 Multi-processing and Analysis

Another speed-up I am planning is multi-processing. Unfortunately the machines in the lab are not dual-core, or even multi-processor machines. I did look into the hardware specifications of the glXX machines although, and it does appear than many (or all) of these machines are Hyper-threaded.

I thought that therefore it would be an interesting idea to try a multi-process approach, and attempt to let the Operating System handle this as a dual (or multi) process, hopefully harnessing some of the power of Hyper-treading on the processor chips. While the code for this will be relatively simple compared to my other objectives, I plan to do a thorough analysis of the results and create several graphs plotting the Raytracing runtime vs. the number of created processes.

This will allow me to find (and use) a specific number of processes which maximizes the computational output per unit time of a specified computer.

1.7.7 Solid Texturing and Procedural Texturing using Noise

Like most people my age, I was really excited when the T-1000 melted up and out of the checkerboard-textured floor in the film Terminator 2.

As an homage to this amazing historical Special Effects sequence, I thought it would be necessary to implement the Checkerboard texture. As described by Pharr in *Physically Based Rendering, pp. 542-543*, Solid Checkerboard is a very simple texture.

Since this alone does not make up much of an objective, I have decided to also implement Procedural Texturing, using a form of noise function. I plan to use Perlin's Noise Function, perhaps using `http://mrl.nyu.edu/~perlin/noise`, a reference implementation of Perlin's paper, if it seems appropriate. The following website should also be useful in determining some functions for wood texture and/or any other procedural textures that I feel are necessary for my final image: `http://freespace.virgin.net/hugo.elias/models/m_perlin.htm`

1.7.8 Area Light Support and Soft Shadows

I will need to add support for Area Lights to get a Soft Shadow effect. I will cast multiple shadow vectors to various points on the light to achieve a percentage of shadow rays that hit the light, which will determine the relative amount of shadow. If jaggies are created by this approach, they will be smoothed out by my Anti-aliasing function.

1.7.9 Glossy Reflections

To make more interesting reflections, I will add capability to specify a material as having "Glossy", or "Diffuse" Reflection. This means that these surfaces will perturb their standard reflective ray in a cosine distribution to give a more interesting glossy reflective effect.

1.7.10 Final Scene

Finally, I will convert all the models from Maya to `OBJ` format, then import them into my Raytracer using my extended `OBJ` importer.

Following this, I will apply Fine Arts concepts to arrange the objects and create the scene in a compositionally well-designed manner. I will use the features created in my Raytracer to create an image which I hope to be quite realistic, and which will display the features that I have created.

1.8 Milestones

The following approximate order will be taken to achieve my objectives.
I will start with the speed optimizations first, to speed up render times for subsequent objectives.

- Uniform Spatial Subdivisions (Grid Creation, WCS Axis-Aligned Bbox Transformations and Voxel Traversal)
- Multi-processing and Analysis
- Adaptive Anti-aliasing
- Refraction
- Glossy Reflections
- Solid Texturing and Procedural Texturing using Noise
- Texture Mapping
- Phong Model using Vertex Normal Interpolation on Triangles
- Area Light Support and Soft Shadows
- Final Scene

1.9 Bibliography

I will likely use the following papers, textbooks and websites in detail, along with several more that I have yet to locate and read.

Amanatides, J., and Woo, A. "A Fast Voxel Traversal Algorithm for Ray Tracing." In Proceedings of Eurographics '87, G. Marechal, Ed. Elsevier North-Holland, New York, 1987, 3-10.

Department of Computer Graphics, "CS488/688 Course Notes", Spring 2006.

Donald and Baker, "Computer Graphics with OpenGL, Third Edition", Prentice Hall, 2003.

Pharr, M., and Humphreys, G. "Physically Based Rendering", Elsevier/Morgan Kaufmann, 2004.

Perlin, K. "Improving Noise." In Transactions on Computer Graphics (Proc. of ACM SIGGRAPH '02), 2002.

Perlin, K. "Improved Noise Reference Implementation", 2002. URL:
`http://mrl.nyu.edu/~perlin/noise`

Williams, A., Barrus, S., Morley, R.K., and Shirley, P. "An Efficient and Robust Ray-Box Intersection Algorithm." In Journal of Graphics Tools, Vol. 10, No. 1:55-60, 2005.

1.10 Organization

The files and organization will be similar to the setup of A4, with extra classes as necessary.

1.11 Documentation

The file README will be a brief guide, describing how to run the Raytracer.
The final report will be of a similar form to the following guide and will be created with Latex and exported to a PDF. This final report will show the algorithms and data structures used in my Raytracer in a detailed fashion.

1.12 Sources

Source files will be of the following format: `src/*.cpp` and `src/*.hpp`.
make will compile the Raytracer.

1.13 Executables

./rt [command-line options]
Command line options will be defined at a later date once I have developed the Raytracer.

1.14 Data Files

The /data/ directory will contain the required `LUA` and `OBJ` files.

Objectives

Due: Thursday, July 20, 2006.

Name: _____

User ID: _____

Student ID: _____

A4 Extra Objective: Reflection

___ 1: **Uniform Spatial Subdivisions (Grid Creation, WCS Axis-Aligned Bbox Transformations and Voxel Traversal):** The Raytracer will use Uniform Spatial Subdivisions as a method to increase efficiency. This requires 3D grid creation, bounding box code for for all implemented primitives and Voxel Traversal using a 3D version of DDA.

___ 2: **Multi-processing and Analysis:** The Raytracer will run a number of processes, greater or equal to 1, which maximizes computational speed. This number will be determined by a set of Raytracing experiments, and will be documented in the analysis.

___ 3: **Adaptive Anti-aliasing:** Anti-aliasing is carried out on the scene to remove jaggies.

___ 4: **Refraction:** Snell's law is used to compute the angle of transmission and secondary rays are cast on intersection with translucent objects to produce refraction effects.

___ 5: **Glossy Reflections:** Objects may have a glossy reflection material applied.

___ 6: **Solid Texturing and Procedural Texturing using Noise:** Textures can be applied to primitives, including checkerboard and procedurals using Perlin noise.

___ 7: **Texture Mapping:** Texture mapping has been implemented.

___ 8: **Phong Model using Vertex Normal Interpolation on Triangles:** Vertex Normal Interpolation is used to create smoother surfaces.

___ 9: **Area Light Support and Soft Shadows:** Planar light sources are implemented and extra shadow rays are used to produce soft shadow effects.

___ 10: **Final Scene:** A final, unique scene is created with techniques from Fine Arts. This scene will demonstrate the features of my Raytracer.

Declaration:

I have read the statements regarding cheating in the CS488/688 course handouts. I affirm with my signature that I have worked out my own solution to this assignment, and the code I am handing in is my own.

Signature: