Feature Complete Ray Tracer

Geoffrey Wright
Zachary Lynn
Rensselaer Polytechnic Institute

Abstract

In homework assignment 3, the class was tasked with implementing a simple ray tracer with features such as sphere intersection, reflection, soft shadows, and anti-aliasing. After researching more distributed ray tracing effects, we decided to implement a range of new features on top of the existing homework 3 infrastructure in order to produce more realistic renderings.

CR Categories: 1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: distributed ray tracing

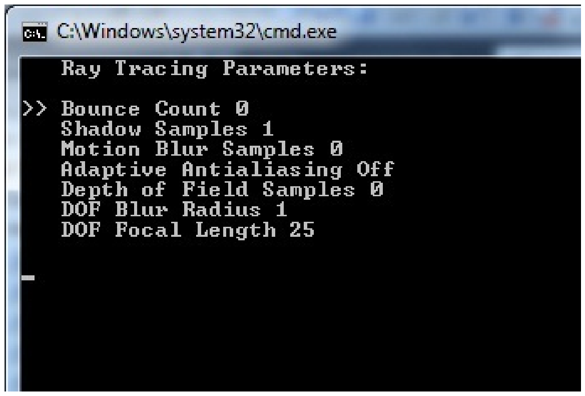
1 Introduction

Distributed ray tracing is an improvement upon ray tracing which allows for other, soft effects through the averaging of multiple rays per sample. This paper demonstrates the implementation of several distributed ray tracing effects, such as glossy reflections, motion blur, depth of field, and adaptive antialiasing. Additionally, we implemented other utility features which improved the quality of the images, such as refraction, or enhanced the usability of our ray tracing test bed, such as in-window parameter controls and various rendering modes.

2 In-Window Parameter Controls

In-window parameter controls allow the user to change parameters of the scene, such as bounce count, shadow samples, use of adaptive antialiasing, and depth of field, in order test multiple setups without restarting the application. Figure 1 shows the list of in-window parameters used in our testbed. This new feature is incredibly useful for determining proper parameters for good renderings when setting up test scenes.

In the original testbed for homework assignment 3, students were required to close the application and enter new parameters in the command line arguments in order to test new parameters. This can be very time consuming, especially in development applications like Microsoft Visual Studio, which do not use a command line interface and require users to traverse the menu system in order to enter these arguments. For example, in setting up our test scene for depth of field, we necessarily had to test several focal lengths in order to find a good depth to place the sharp plane. It became rather simple for us to test this and compile good test scenes, as we were able to simply vary the parameters quickly within the application instead of closing the application each time to change parameters.



```
   Ray Tracing Parameters:

>> Bounce Count 0
   Shadow Samples 1
   Motion Blur Samples 0
   Adaptive Antialiasing Off
   Depth of Field Samples 0
   DOF Blur Radius 1
   DOF Focal Length 25

_
```

Figure 1: In-window parameter controls used in our ray tracing testbed. The user can edit these parameters without exiting the application, making testing much quicker.

3 Render Modes

Ray tracing is an expensive process that can take a significant amount of time, depending upon the complexity of the scenes and the resolution of the image to be produced. Rendering every pixel for testing would be inefficient and wasteful. The user needs the capability to quickly render an image to determine if the parameters they have specified will yield desirable results. However, the user also needs the capability to render an image without the iterative refinement of our quick render method. Also, the user needs the ability to save their images quickly and efficiently without interruptions in the ray tracing process. As a result, we have implemented three unique rendering modes to satisfy the demands of the user.

3.1 Quick Render

The quick render mode iteratively refines the image by sampling finer regions from the entire image to the size of a single pixel. This allows the user to quickly identify if their specified parameters produce the intended image. Additionally, quick render allows the user to see artifacts in the rendered image caused by input parameters. Figure 2 shows an image during the quick render process.

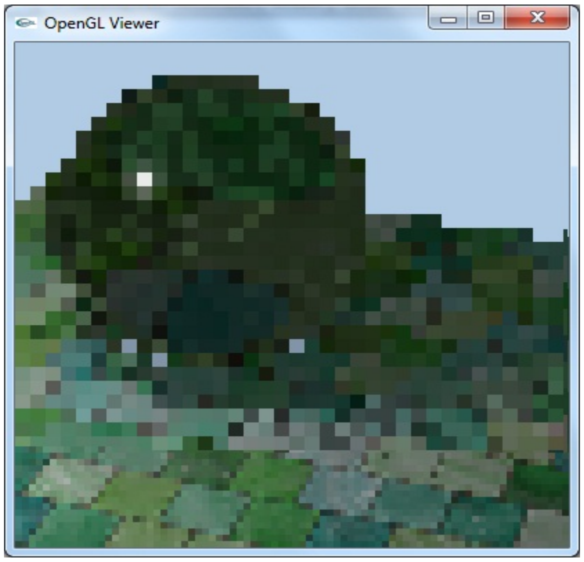


Figure 2: Image during the quick render process.

3.2 Full Render

Once the user is satisfied with the parameters of the test scene, the full render mode renders the scene pixel by pixel, forgoing the iterative process used by quick render in order to speed up the rendering time considerably.

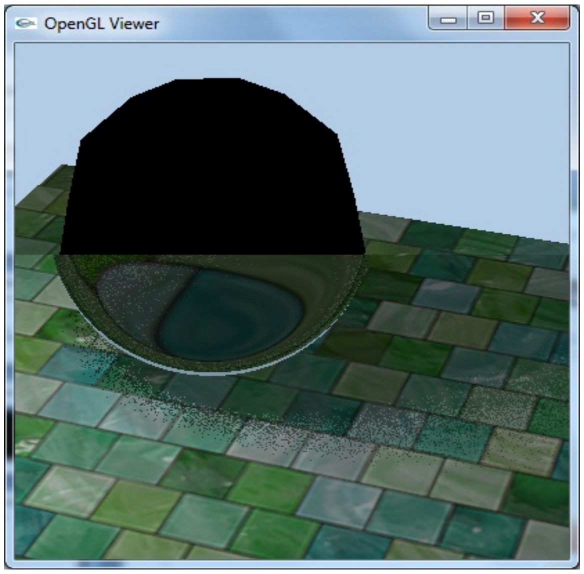
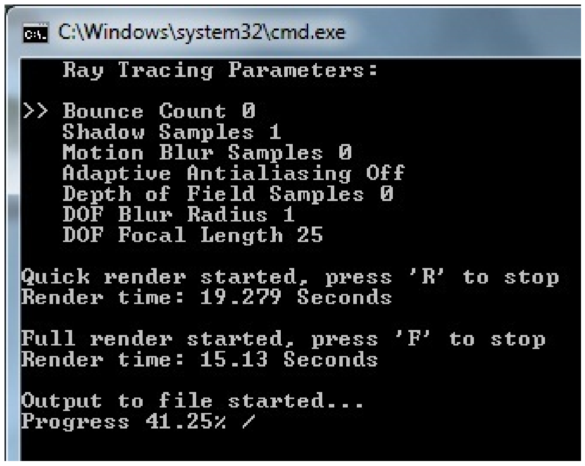


Figure 3: Image during the full render process.

3.3 Render to File

The render to file feature allows users to render the scene directly to an image file. This offers several advantages to the user. Render to file does not render anything to the screen and saves time on OpenGL calls. Also, this process cannot be interrupted and doesn't allow OpenGL to redraw the mesh if the render window goes in and out of focus. Additionally, this allows the user to save the image to a file without having to take a screenshot of the render window.



```
   Ray Tracing Parameters:

>> Bounce Count 0
   Shadow Samples 1
   Motion Blur Samples 0
   Adaptive Antialiasing Off
   Depth of Field Samples 0
   DOF Blur Radius 1
   DOF Focal Length 25

Quick render started, press 'R' to stop
Render time: 19.279 Seconds

Full render started, press 'F' to stop
Render time: 15.13 Seconds

Output to file started...
Progress 41.25% /
```

Figure 4: The render to file feature shows the progress of writing the image to a file in the console window

4 Adaptive Antialiasing

The implementation of antialiasing in homework assignment 3 has two main drawbacks. First, it casts random samples through each pixel, which can create artifacts if the random samples are all cast in the same general region of the pixel. Also, since the algorithm casts samples through every pixel, many areas are unnecessarily sampled. Second, the algorithm does not allow for further refinement in areas of the image which require it.

The algorithm we implemented breaks each pixel into four subpixels, casting a ray through each subpixel. These four samples are averaged to determine the color of the pixel. If the color of any sample is sufficiently different from the average, the algorithm is recursively called on the corresponding subpixel. We allow for up to three recursive calls within any one pixel in order to avoid unnecessary computation for marginal improvements.

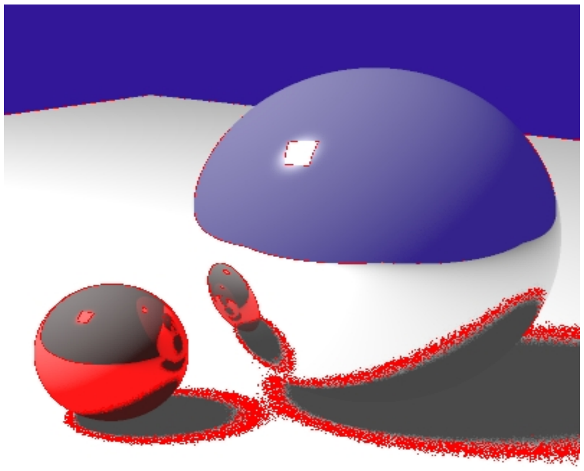


Figure 5: Image using adaptive antialiasing. The red pixels indicate areas targeted by the algorithm.

5 Refraction

Refraction is the bending of light as it passes between two different mediums. In order to implement refraction, we added an additional parameter to each material in the scene obj file called IOR (index of refraction). We used the IOR to determine the direction a ray (D) gets refracted (R) at a surface normal (S) according to the following formulas.

N_1 = IOR of the original medium
N_2 = IOR of the new medium
$N = N_1 / N_2$
$C_1 = -\mathrm{dot}(S, D)$
$C_2 = \mathrm{sqrt}(1 - N^2 * (1 - C_1^2))$
$R = (N * D) + (N * C_1 - C_2) * S$

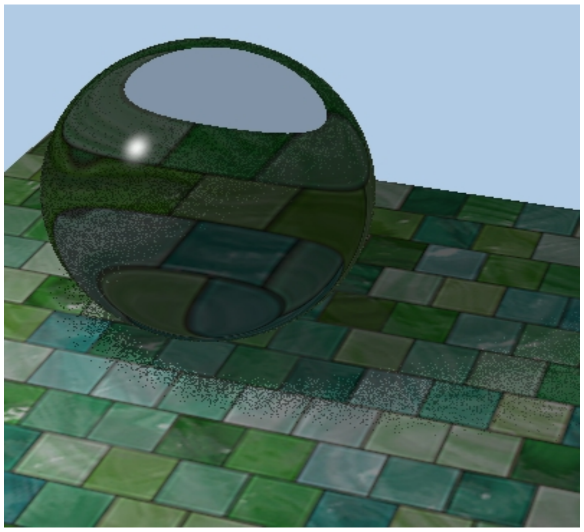


Figure 6: Refraction of light through a glass sphere with index of refraction 1.3

6 Motion Blur

Motion blur is the apparent blurring of objects that move in a still image. In order to implement motion blur we needed to add a velocity component to each object, and a time component to each ray. Using these additionally fields, we could sample an image over time. To control the quality of the motion blur we added a parameter specifying the number of temporal samples to use per pixel. Figure 7 shows the motion blur algorithm using only 5 samples. With such a low sample count, the various time samples can be easily distinguished. Figure 8 shows the same render using 50 samples, which exhibits the blurring effect in higher detail. Using more samples is unnecessary; little more detail is shown.

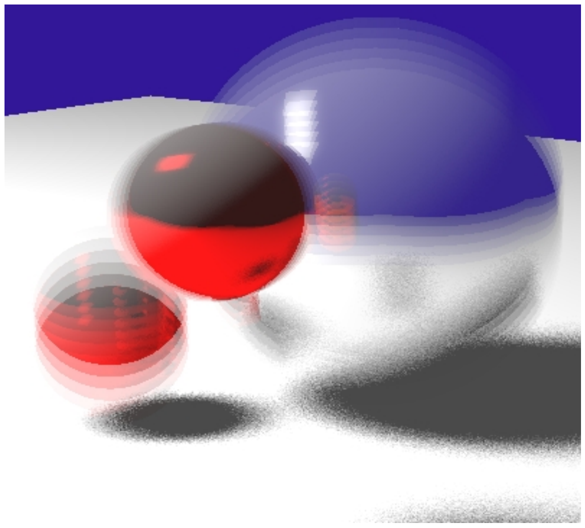


Figure 7: Render showing motion blur using 5 temporal samples.

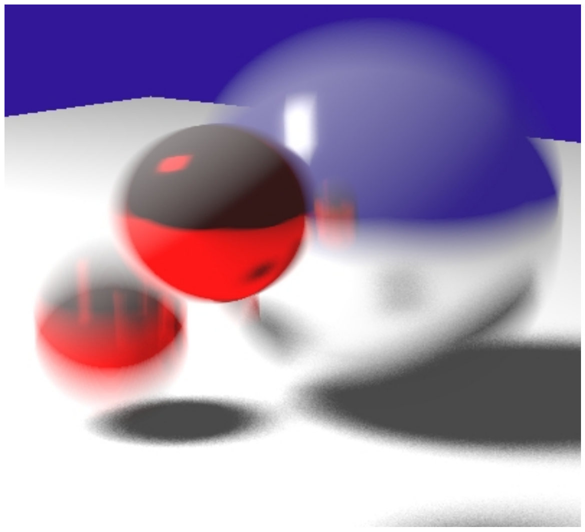


Figure 8: Render showing motion blur using 50 temporal samples.

7 Glossy Reflections

Glossiness is an optical property of a material's surface which measures the roughness of the object's surface. When rays reflect off the surface, the mirror angles are perturbed by the roughness, causing a blur in the reflection. To simulate this effect, we averaged the color of multiple reflection ray coming off the surface at relatively random angles. In order to determine the range of these angles, we first followed a ray in the perfect mirror direction from the glossy sphere. After following the ray one unit from the sphere's surface, we create a new sphere. We then cast rays from the glossy sphere to randomly chosen points on this new sphere. The radius of this sphere will determine how glossy the reflection looks because as the sphere's radius increases, the range of angles increases. Additionally, more samples should be used as the radius increases to improve the blur quality.

See figure 9 for an image representation of this algorithm.

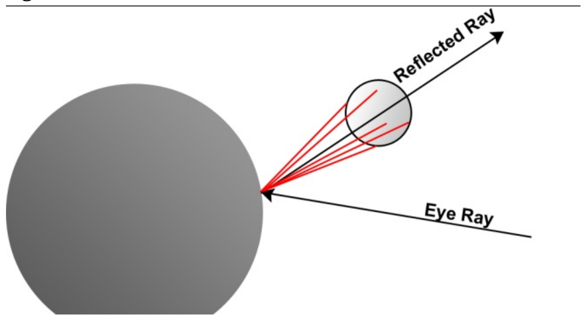


Figure 9: Illustration of the glossy reflection algorithm.

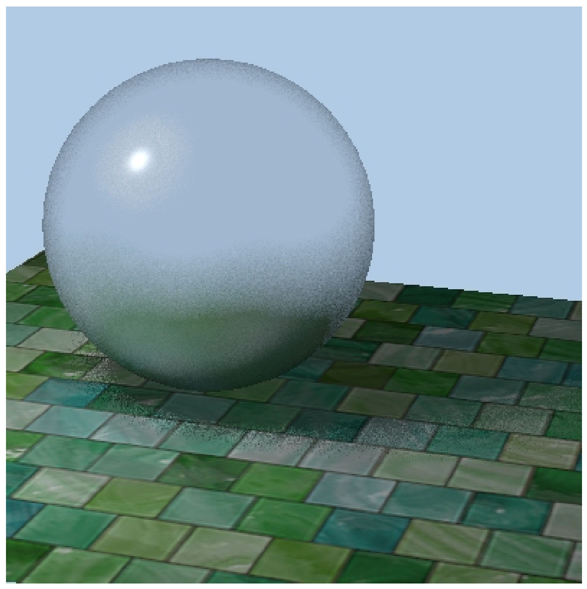


Figure 10: Render of a sphere with a 50% glossy surface.

8 Depth of Field

Depth of field is distinction in sharpness of objects which are at different distances from the camera. The camera uses a near and far plane in order to create the depth of field effect. All of the rays shot out from the camera intersect the near plane, where the severity of the effect is determined. We modeled the near plane using a similar technique as the glossy reflection algorithm. By placing an imaginary sphere centered at the intersection of the ray and the near plane, we can vary the amount of blur by casting new rays from random points on this sphere. We cast rays from these new points on the sphere in the direction of the intersection between the original ray and the sharp plane. The distance to the sharp plane (focal length) determines which portion of the scene is in focus. Figure 11 demonstrates this algorithm visually.

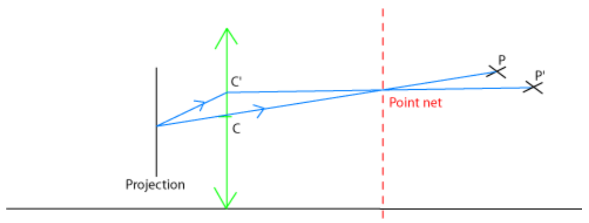


Figure 11: Visual representation of the depth of field algorithm.

In our implementation, we allow the user to vary the focal length (distance to the sharp plane) and the blur radius, which determines the severity of the blur for objects out of focus. Additionally, the user can specify the number of rays per sample for the depth of field effect. Figure 12 shows an example of depth of field in which the sphere is the focal point of the image.

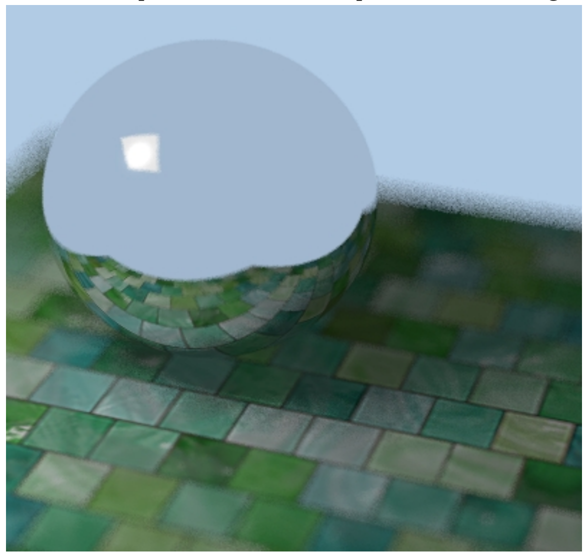


Figure 12: Example rendering of the depth of field effect using 100 samples per pixel.

8 Results

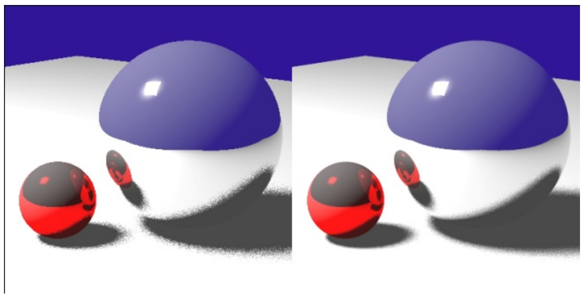


Figure 13: Adaptive antialiasing algorithm. Comparison of off (a) and on (b).

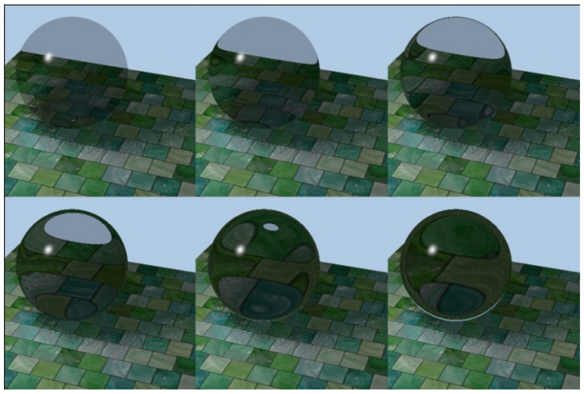


Figure 14: Refraction algorithm. Comparison of index of refraction: 1.01 (a), 1.1 (b), 1.2 (c), 1.3 (d), 1.4 (e), and 1.5 (f).

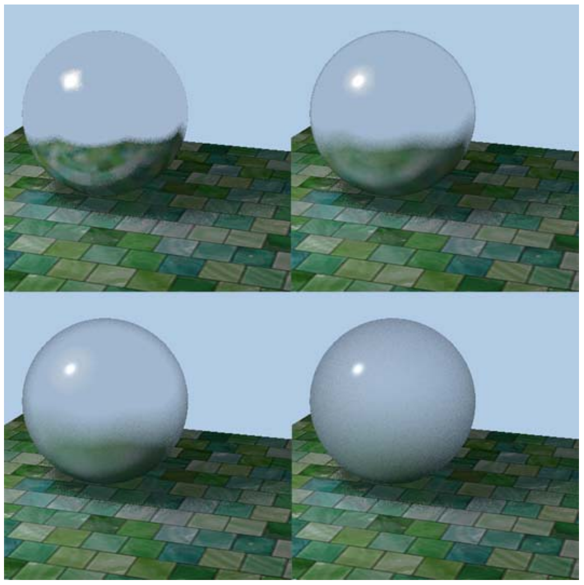


Figure 15: Glossy reflection algorithm. Comparison of glossy values: 0.1 (a), 0.25 (b), 0.5 (c), and 1.0 (d).

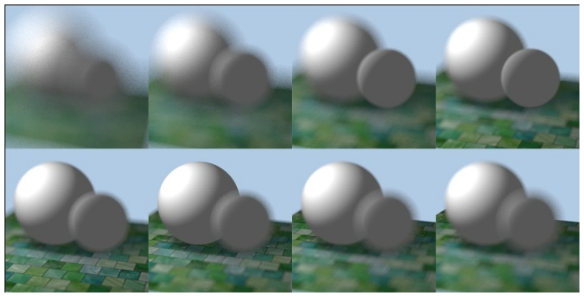


Figure 16: Depth of field algorithm. Comparison of focal lengths: 15.0 (a), 20.0 (b), 25.0 (c), 30.0 (d), 35.0 (e), 40.0 (f), 45.0 (g), and 50.0(h).

9 Conclusion

We successfully implemented all of our planned features. Some of the features, primarily depth of field and motion blur, were quite computationally expensive, but our goal was not to make these algorithms as efficient as possible; our goal was simply to make them work and render some good looking images.

10 Sources

"A Raytracer in C++ - Part IV - Depth of Field, Fresnel and Blobs." CodermindIn a Coder's Mind, 2008. Web. 5 May 2011. <http://www.codermind.com/articles/Raytracer-in-C++-Depth-of-field-Fresnel-blobs.html>.

Cook, Robert L., and Porter, Thomas, and Loren Carpenter, 1984. Modeling the interaction of light between diffuse surfaces. *Proceedings of SIGGRAPH 1984*.

Genetti, Jon, and Dan Gordon. "Ray Tracing With Adaptive Supersampling in Object Space." UAF Department of Computer Science. Web. 05 May 2011. <http://www.cs.uaf.edu/~genetti/Research/Papers/GI93/GI.html>

Mitchell, Don, 1990. The Antialiasing Problem in Ray Tracing.

Rademacher, Paul. "Ray Tracing: Graphics for the Masses." Department of Computer Science, UNC-Chapel Hill. Web. 05 May 2011. <http://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html>.

Whited, Turner, 1980. An Improved Illumination Model for Shaded Display. *Communications of the ACM, 23, 6*.