Extended Ray Tracer

Ken Bellows
Rory Murphy

Abstract:

We present an extension of the homework 3 code, implementing algorithms and designs proposed by Cook et. al. in the seminal 1998 paper “Distributed Ray Tracing”, as well as completing features proposed in "An improved illumination model for shaded display" such as calculation of transmitted light through translucent mediums. Combining these features, along with tidying up the original Homework 3 code, allows us to render computationally complex, but visually impressive scenes via raytracing.

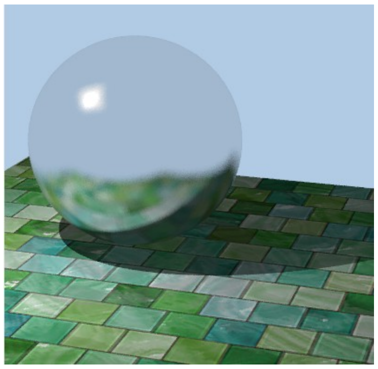
1. Introduction

RayTracing has long been considered one of the most accurate rendering methods, but it also suffers from being very inefficient. In fact, raytracing is so slow it is virtually unusable in any sort of real-time rendering situation. Also, due to the number of calculations that are required in the bounces of just one ray, even small increases in quality severely increase the rendering time on a given scene. The tradeoff for this is an almost unerring accuracy in the final rendered images, as well as the ability to implement several different effects quite easily. For this project, we decided upon 4 major features we wished to add to the RayTracer. Glossy or blurred reflections, transparency, Depth of Field, and motion blur.

2. Process

In an attempt to balance the workload between both group members, we divided the 4 features into 2 groups, and tasked each member with completing two features. Rory completed the Glossy Reflections and Depth of Field implementation, while Ken worked on transparency and motion blur. The team met weekly to ensure progress was continuing apace of the expected schedule, and we reviewed our current tasks after every class.

3. Glossy Reflections

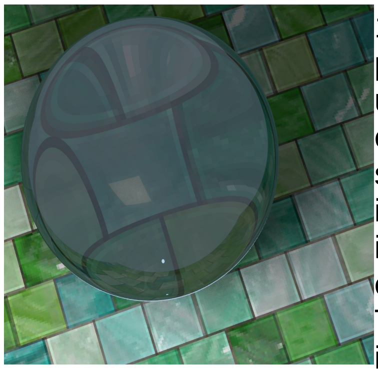


In order to implement glossy reflections, we needed a “glossiness” setting for a given material, as well as the number of Glossy rays that should be shot upon each glossy reflection. We co-opted the “roughness” parameter already in place for materials to act as an indicator of glossiness. The roughness of a sphere can range between 0 and 1, where 0 is a perfectly reflective surface and 1 is a perfectly diffuse surface. In order to properly simulate a glossy surface reflection, upon a hit we send multiple rays out from the point of reflection. However, instead of sending a ray in a perfect reflected direction from the point of contact, we vary the destination of the ray. The range of possible variation is dependent upon the roughness of the reflective material. Ideally, the distribution function is weighted similarly to the phong shading model. Unfortunately, we did not have time to implement this distribution, so we instead have a distribution of uniform radius about the reflected direction.

4. Transparency

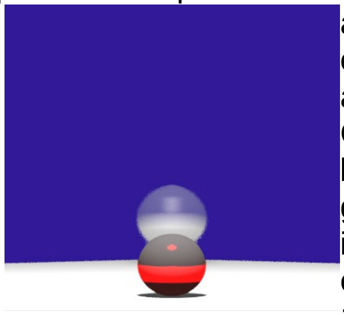
Ray tracing through transparent materials is based largely on refraction. Thus, the math is based on Snell's Law: Ni * sin(Ai) = Nr * sin(Ar), where Ni = refractive index of incident material, Nr = refractive index of refractive material, Ai = angle of incidence, and Ar = angle of refraction. This results in Ar = arcsin(Ni/Nr * sin(Ai)).
Essentially, once the ray orthogonal to the normal in the plane (assumed to lie parallel to the Y-axis) is found, the projections of the new vector onto the Normal and Orthogonal vectors is easily found and summed to find the final direction vector.

It is quite interesting to see just how much a small change in the refractive index can change the picture. A sphere with an index of



1.15 will render a picture right-side-up, with distorted edges. However, a sphere with an index of 1.33 will invert the picture entirely.
The other interesting thing is combining reflection with transparency. It is important to remember to properly average the values so that the colors are not simply summed, which would flush out the colors and create an overly bright image.

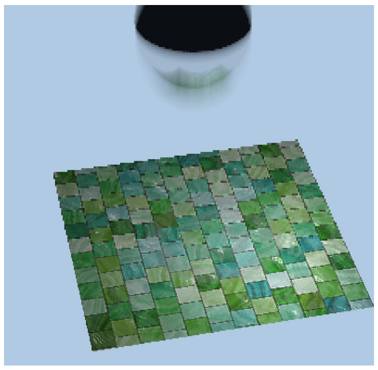
5. Depth of Field
The Depth of Field implementation was similar to the process which was used for glossy reflections. However, we require 3 inputs instead of two. the three needed inputs are the simulated aperture, the focal distance, and the number of depth of field rays the user wishes to shoot into the scene. Then, the calculation proceeds like so: first, starting with a normal ray in the center of a given pixel, find the point *p*, exactly the focal distance along the center ray. Then, vary the origin of the ray about the center of the
current pixel. The size of the domain which the origin is allowed to vary within is determined by the user given aperture. This process is repeated for multiple random rays



around the initial origin, as many time as the user specifies. Cook recommends at least 100 rays for good results, but there is a noticeable effect even when as few as 10 rays are sent. An effect is created where only objects that are the focal distance from the camera appear to be in focus, and everything closer and further is blurred.

6. Motion Blur
Motion blur is a matter of averaging a certain pixel's color over a specified period of time. The idea is based on a long exposure on a film. Of course, it is not a good idea to simply add up all of the colors, because the image will very quickly become flushed out. So, the most intuitive option, and the one we implemented, is essentially to sum the colors observed at each timestep and divide by the number of timesteps.

In order to implement this at a very basic level, the object file is given six additional values, comprising two additional 3 vectors: initial velocity = V(x,y,z) and acceleration = A(x,y,z).



Additionally, two new command line arguments are introduced: length of the timestep to be used and number of timesteps to traverse.
This works well enough, but the images seem intuitively too foggy and transparent. It is very difficult to see the object being blurred. However, without a real scene to compare to results, it is difficult to tell just how accurate results are. Of course, the scene looks much better with a smaller timestep, as each pixel is averaging in more frames of the object and fewer of the background.

7. Results
We were able to get all 4 features working properly. The included images show the various features being tested with varying input parameters. Some features, such as depth of field, are much more costly than others, and the combination of different effects can compound this problem further. In addition, several of the different effects do not interact properly. The most glaring one is the interaction of gloss and transparency. Ideally, a transparent, glossed object should look to be made of frosted glass. However, the refracted surfaces appear too dark when they are glossed over. The refracted images do blur

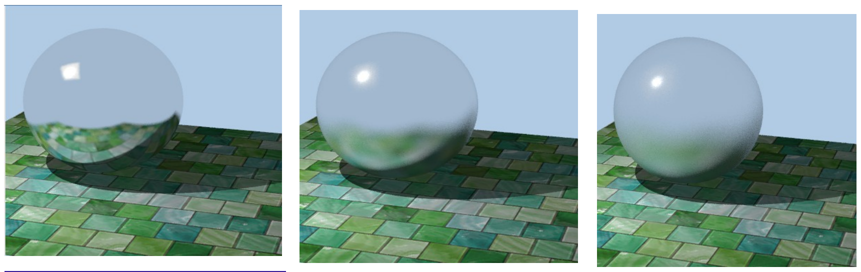
properly though. Small problems like these can negatively affect the quality of the image, and were our main source of trouble as we prepared the final code.

8. Closing

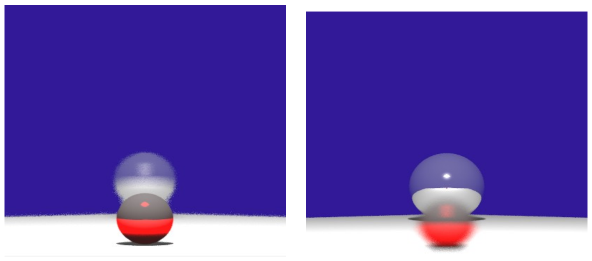
We fully completed all features that we outlined for our project, but we did not get to any of the extended features we wished to implement. This was due to our desire to polish the required features and ensure they worked well over simply implementing as many features as possible. Going forward, we would like to correct this code further in the hopes of making this raytracer faster, more accurate, and more reliable.

9. Sources

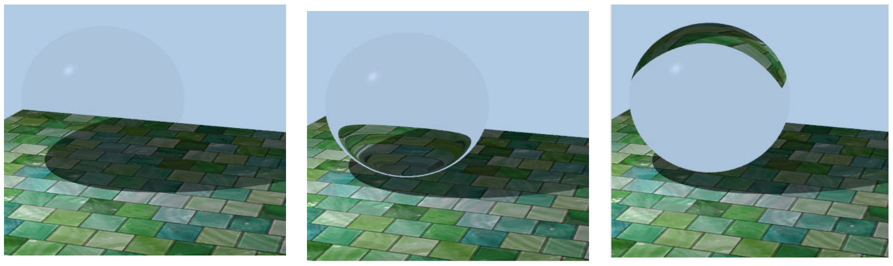
- "An improved illumination model for shaded display" Turner Whitted, 1980.
- “Distributed Ray Tracing” Robert L. Cook, Thomas Porter, Loren Carpenter, 1984



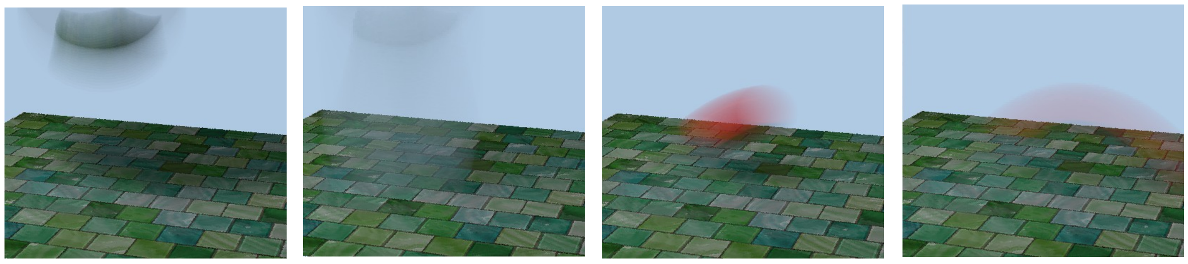
Examples of the same scene with Glossiness of 10%, 50%, and 100%



The same scene is rendered with the focal depth set to 10 and 20, respectively



A translucent Sphere, rendered with refractive indexes of 1, 1.05, and 1.33



The two images on the right are the same scene with a timestep of 0.05 seconds and 25 and 100 iterations, respectively. A similar effect can be seen in the right two images, which have 70 and 200 iterations for their timestep of 0.25 seconds

10. Bloopers

Here are some of the major issues we ran into along the way, with corresponding screenshots for some below.

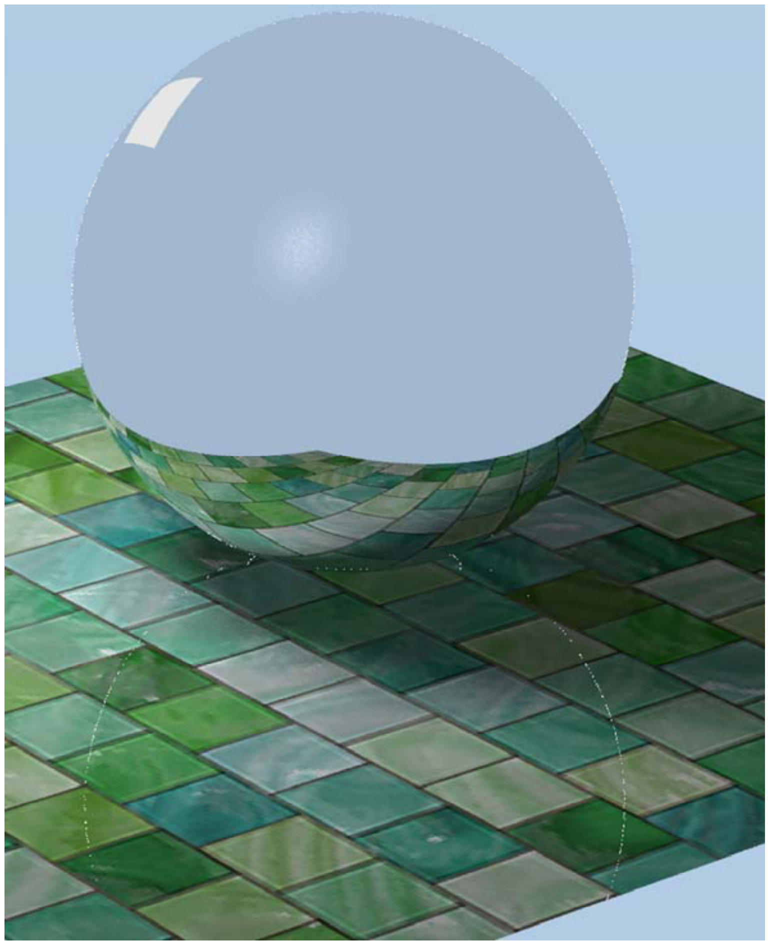
a) There were some very strange issues when we first began the project regarding shading and anti-aliasing. They mostly turned out to be issues with the sphere object's intersection method.
b) The math for refraction was very wrong for a long time. The refraction angle vectors were all wrong.
c) After the math issues were resolved, there was an additional sphere intersection issue that prevented the ray from noticing the intersection with a sphere as it left it, after the refraction. Thus, it would hit the ground and refract off into space or it would simply hit nothing.

11. Time Table and Work Division

4/24:
Rory - Glossy reflection
Ken - Transparency

5/1:
Ken - Motion Blur
Rory - Depth of Field

a) Sphere intersection bug. Note the white, spotty anomalies around the upper edge of the sphere and on the ground, forming a circular shape.



c) Another sphere intersection bug. The sky is being shown from all angles. Note the faint concentric rings of proper color among the blue.