Adaptive semi-transparent ray tracing with depth of field

Kevin O'Connor
Rensselaer Polytechnic Institute
oconnk6@rpi.edu

Dimitar Dimitrov
Rensselaer Polytechnic Institute
newbrict@gmail.com

ABSTRACT

We present a combination of techniques to implement a feature-rich ray tracer that applies adaptive sampling to optimize performance. We build these features upon existing methods in which we experiment with techniques to optimally combine them in a ray tracer.

Keywords

Computer Graphics, ray tracing, adaptive supersampling, refraction, depth of field, soft shadows

1. RELATED WORK

Our methods are strongly built upon existing work from several different authors using various techniques.

Cook proposes a solution to the artifacts created with uniform sampling [1]. The method to fix this is through nonuniform sampling.

The basis of our ray tracer is based upon the work of Cook et al. [2] on their distributed ray tracer. Their work focuses on distributed rays in the direction of the analytic function they sample in order to reduce the "fuzziness" in existing ray tracing. They further provide methods for calculating depth of field, penumbras, translucency.

For adaptive supersampling, Whitted [3] presented an approach for doing such by subdividing the pixel and computing average color. An ϵ range was used to compare whether or not a color was significantly different and requiring further subdivision.

Yauney [4] did a similar approach for his project in the Spring of 2012 and sampled a few different techniques to Whitted's [3] work in order to implement a fast ray tracer.

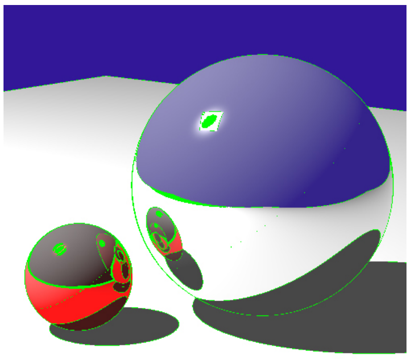


Figure 1: Adaptive supersampling for AA visualized using green pixels where additional rays were cast. Noise leads to regions with aliasing not being perfectly outlined in green

2. ADAPTIVE SUPERSAMPLING

One of the main focuses of our project was to optimize our ray tracer in order to reduce the amount of rays that must be casted into the scene. This was an important feature due to our addition of depth of field which adds considerable more ray casts. Therefore we implemented methods published by Whitted [3] in order to reduce wasteful ray casts.

2.1 Antialiasing

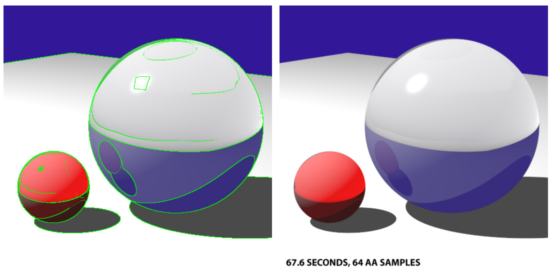


Figure 2: Using the corner-point method for AA sampling we were able to perfectly target aliased regions using an ϵ of 0.01 leading to a 500x500 render with a max depth of 64 AA samples being rendered in 67.6 seconds

Building upon Whitted [3], we first shoot a ray directly into the center of a pixel and record the color there. We then shoot four rays in the corner boundaries of the pixel and average their color. Given a defined ϵ (determined by the user), we see if the difference between the color at the center pixel differs by more than ϵ to that color calculated from the corners. If it does not then we immediately return the average color of the corners. If it does differ by more than ϵ we then iterate where $i = 2 \ldots n$. At each iteration we shoot 2^i rays stratified randomly across the pixel and average their color. We then recompute the difference against the last iteration (or the four corners for the first iteration) and compare it against ϵ. The user defines n to be the depth at which they desire to stop if the ϵ check has not yet passed.

Our first iteration of this algorithm omitted the corner check and only went to the first iteration in which the points are chosen stratified randomly in the pixel. What we found was that there was noise in the resulting image where AA occurred as seen in Figure 1.

We then switched the the corner-point method to help better identify points where a majority of the pixel might be one color with a slight variation in one of the corners being missed when randomly sampled. Our result from this method can be seen in Figure 2 in which we were able to perfectly identified aliased regions and only shot more than 5 rays in those regions. All other regions of the image received exactly 5 rays.

2.2 Soft Shadows

Our initial soft shadow implementation rendered the penumbras using the methods developed by Cook et al. [2]. The unoptimized version develops the penumbras by tracing a ray from a point in object space to the light source many times and averaging the light contribution based on where or not the ray hit the light or an blocking piece of geometry.

To optimize this we applied the basic methodology from Whitted [3] for antialiasing to this situation. Given a point in object space we shoot four rays into the corners of the light source. If there is a consensus in the rays of either all hitting the light or all not hitting the light source then we know the point is not the penumbra and can be either entirely in shade or entirely unshaded. We demonstrate this targeting in Figure 3 where we target the penumbra regions of the scene.

Once we have targeted the regions in the penumbras we shoot n rays, determined by user preference, to the light source and average their shaded values. This means that we only cast four rays for all pixels not in penumbras and exactly n for pixels in the penumbras.

In our scene in Figure 3 we were able to render the scene with 128 shadow samples in approximately 129.38 seconds. For comparison, the render without any shadow samples took 36.1 seconds. The unoptimized render using 128 shadow samples took 309.68 seconds.

We attempted to use a similar method as we did with antialiasing in order to implement early stopping to reduce the amount of rays we shot. This method, however, generates a lot of noise in the resulting output. We attempted to balance an ϵ value and the numbers of rays being casted, but couldn't come to a balance that we found satisfactory.

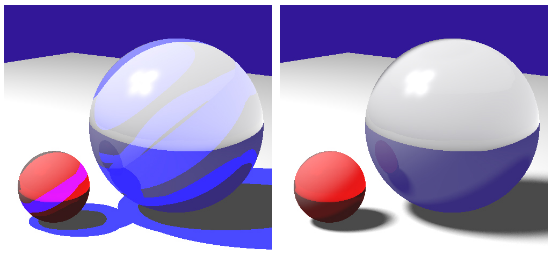
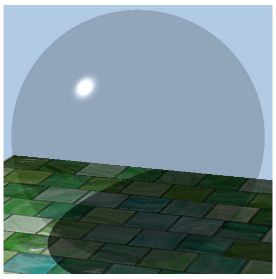
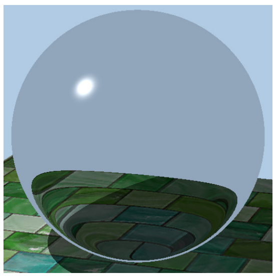


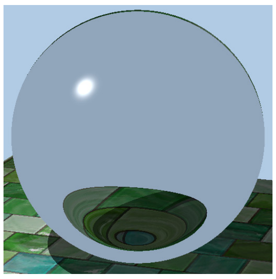
Figure 3: Our implementation targets the penumbra regions, colored in blue, to determine where additional rays must be traced. On the right we after the render of a ray tracing with 128 shadow samples rendered in 129.38 seconds.



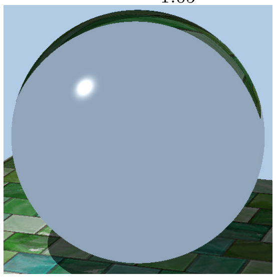
(a) $\eta = 1$



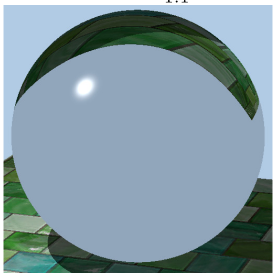
(b) $\eta = \frac{1.000293}{1.05}$



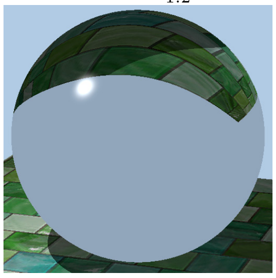
(c) $\eta = \frac{1.000293}{1.1}$



(d) $\eta = \frac{1.000293}{1.2}$



(e) $\eta = \frac{1.000293}{1.33}$



(f) $\eta = \frac{1.000293}{1.5}$

Figure 4: Various renderings of a refractive sphere with different values for η.

3. REFRACTION

Refraction is the phenomenon that occurs when a light wave passes between two objects with differing indices of refrac-

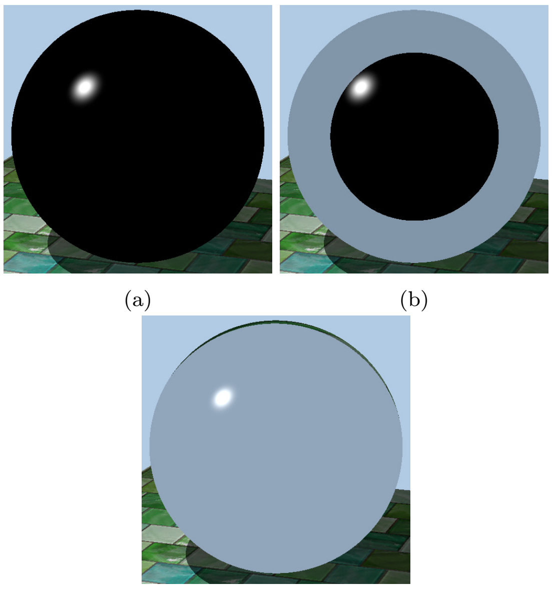


Figure 5: 5a has incorrect values for both N and η. 5b has an incorrect value for N. 5c has an incorrect value for η.

tion. Figure 4 shows our implementation of refraction with various indices of refraction.

Implementing refraction is deceptively complex. Given the incident ray I, surface normal N, and the ratio of the indicies of refraction $\eta = \frac{n_i}{n_r}$, We use GLMâĂŹs implementation of snell's law to compute the direction of the refracted ray R. The order of refractive media is important for both η and N. In our implementation we check if we are within an object and invert both η and N if that is the case. Figure 5 shows the results of incorrect incorrectly setting the values for η and N. The results of having both N and η incorrect are very similar to those with only an incorrect value for N. The black areas show where total internal reflection would occur erroneously since the variables were incorrect. The trickiest case by far is having a correct value for N but an incorrect value for η which produces convincing results with a low tolerance for the camera angle.

For semi-reflective refractive objects we simply use a contribution ratio C. We compute all reflection and refraction at the same time and weight their contributions to the final sample color by C and $(1 - C)$ respectively. Figure 6 shows two renderings using this method.

4. DEPTH OF FIELD

Depth of field is an effect commonly seen in photography and cinematography which produces blurry images with objects at the focal length being in focus. In the physical world depth of field is caused by the diameter of the aperture, the focal length of the lens, and the camera's distance to each object in the scene. To simulate depth of field we assume our camera is the lens, and our focal length r is the distance between the camera and the point of interest. We use a Monte-Carlo approach by uniformly sampling a spherical surface patch of points at the blur radius, which is analogous to the circle of confusion in physical models. We rotate the camera around the point of interest at a radius r for each sample, and average the color per pixel.

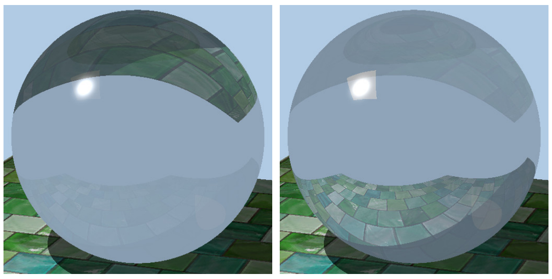


Figure 6: The left image was rendered with $C = 0.1$, the right with $C = 0.5$. Both were rendered using a reflective depth of 3. You can that the light source was internally reflected from the bottom right of each image.

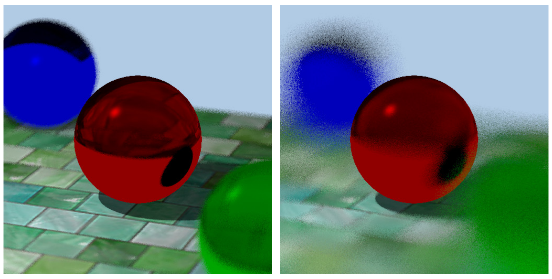


Figure 7: The image on the left was sampled over a blur radius of 2, the image on the right was sampled over a blur radius of 5.

Producing smooth blurry depth requires a very high number of samples, and in turn takes a very long amount of time to render. Also if you increase the blur radius you have to increase the number of samples to produce similar quality images Figure 7 shows just how dramatically a change in the blur radius can affect the smoothness of an image.

All of our features work together nicely, Figure 8 is a rendering using adaptive sampling for both antialiasing and shadows, with refraction, and depth of field.

It's very important that you return the camera to its original position after each sample, this is a mistake we made early on which produced some very interesting results shown in Figure 9.

5. LIMITATIONS

Our method of adaptive supersampling is robust enough to handle most renders that would be thrown at it. However, there are conditions in which there may be a tiny object that might be missed by ray tracing through the corners. We were not able to create a realistic scene where this error would occur, but it is still possible. However, this is a compromise we were willing to settle for in exchange for the performance increase.

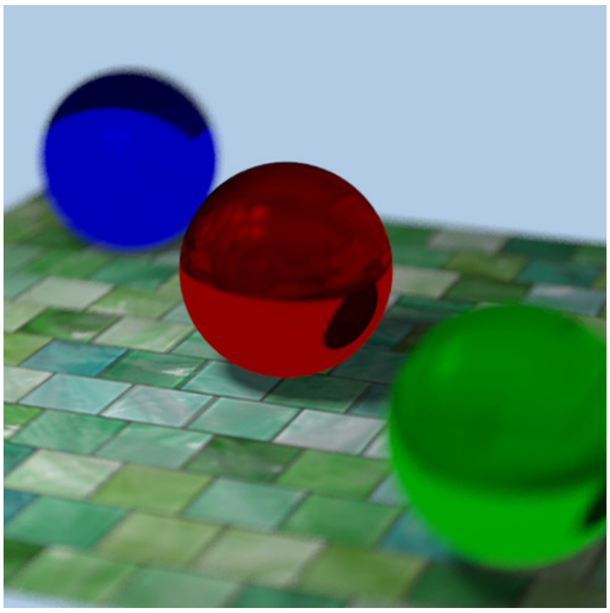


Figure 8: An rendering combining all of our techniques. 4 shadow, and antialias samples, reflective depth of 3, 70 depth of field samples, a blur radius of 5, and $\eta = \frac{1.000293}{1.5}$. The rendering took 6 hours to complete.

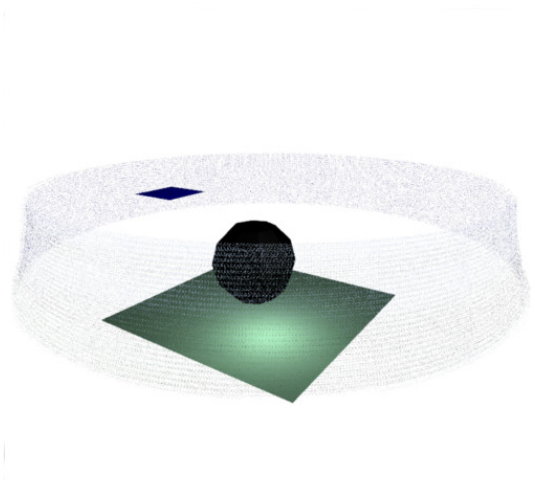


Figure 9: An incorrect repositioning of the camera caused the image plane to be spread along the cameras path.

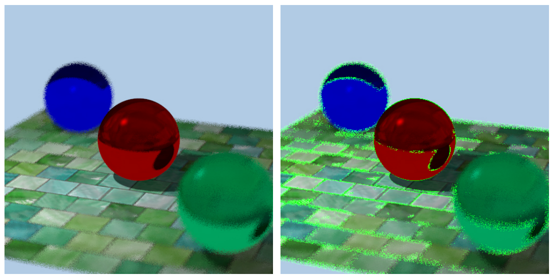


Figure 10: Both images were rendered with 64 antialias samples, 1 shadow sample, reflective depth of 3, 5 depth of field samples, a blur radius of 2, and $\eta = \frac{1.000293}{1.5}$. The right image is the same as the right but with debugging pixels turned on to show where antialiasing occurred.

Since our method for adaptive soft shadows uses a similar method as antialiasing, it is possible that for an occluder in the center of the pixel, but not the corners, to be missed. Given a big enough light source, this could very much be a concern and it's something that we would have liked to expand on. For our current scenes we were not able to see any artifacts from this compromise. Given more time we would have liked to spend more time finding a way to implement early stopping to reduce the number of rays being casted for soft shadows. We also started work on glossy reflections and refractions, but they were not finished in time for the paper.

Adding depth of field samples on top of our adaptive super sampling is very time consuming. We contemplated the idea of mapping the adaptive sampling ϵ value to a blur radius based raised cosine probability density function around the point of interest. This would minimize wasted sampling done on inherently blurry parts of the image. Figure 10 shows how even blurry parts of the image are sampled for antialiasing with our current implementation.

6. CONCLUSION

Overall we were able to complete all the parts of the project that we had planned on finishing. We spent about a week and a half on the actual implementation of our methods and building scenes to test with. Dimitar mostly focused on the depth of field and refraction implementation while Kevin worked on the adaptive antialiasing and soft shadows. We feel that we were successful in our implementation and were ultimately able to render very photorealistic renders in a reasonable amount of time.

7. REFERENCES

[1] R. L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, Jan. 1986.

[2] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, Jan. 1984.

[3] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.

[4] G. Yauney. Kind of quick ray tracing. 2012.