CS 384G Final Project: Rendering Caustics

Jared Croy
jac5659

1 Introduction

Standard ray tracing is able to render some rather incredible scenes. The key technique for ray tracers is to follow a ray from the camera to the scene and calculate the amount of light that is located at that point. This works very well for most cases. However one phenomenon that is not possible is the rendering of caustics. Caustics are caused by light being reflected or refracted such that the light gets focused in an area, causing that surface to look brighter. Some common examples of caustics are the patterns caused at the bottom of a swimming pool and light being focused by a magnifying glass. In order to get closer to realistic rendered images, a ray tracer should implement at least this effect.

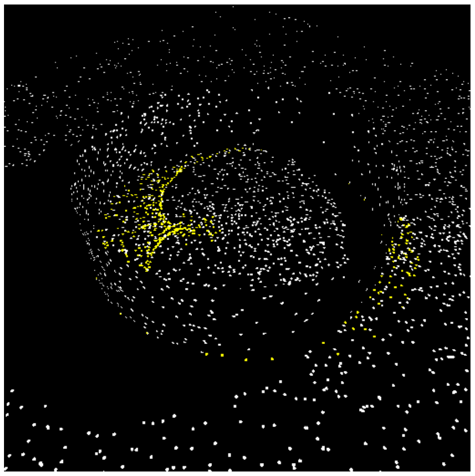
2 Implementation

Because caustics are caused by light being bounced around a scene, vanilla ray tracing techniques will not be enough. In order to render these artifacts we will need to implement simple photon mapping. Photon mapping can be thought of as the opposite process of standard ray tracing. A light ray is emitted from a source and its path is traced following reflection and refraction. When the ray hits a diffuse surface its location and intensity is stored in a photon map. The photon map can then be used by a standard ray tracer to determine where photons have been focused and how the illuminate the area. In the real world, there are an infinite number of photons emitted by every light source. Since this is impossible to achieve using a computer, photons are emitted in a random direction and their intensities are interpolated by taking the sum of the intensities of all the photons found in an area and dividing by the area. In this implementation a simple nearest neighbor algorithm was used to find all the photons that fell within an area around a point. This interpolation method was taken directly from Henrik Jensen's 1996 paper, "Rendering Caustics on Non-Lambertian Surfaces." The issue with emitting photons in random directions is that it is highly probable that the photon will never intersect with the scene, resulting in a large amount of wasted cycles emitting useless photons. Since objects within our ray tracer have bounding box capabilities, we can use a clever heuristic to "point" the random photons in the right direction. Using the minimum and maximum values obtained from the scene's bounding box generate a random point within the box. Then simply normalize the difference of the random point and the light's position to find a "random" direction that has a much higher probability of intersecting with our scene.

The images in Figure 1 are a couple examples of how photons are stored in a photon map. For display purposes, reflected and refracted photons are shown in yellow and blue respectively. Figure 1a shows the map created by photons hitting a reflective ring. Notice the nephroid curve generated by the reflected photons. Figure 1b is the result of photons refracting through a transparent glass lens. Both of these scenes were made emitting 50,000 photons. Attempting to compute the interpolated intensity of each of these photons would be too computationally intensive to realistically use. Therefore we will only be storing the photons that occurred because a photon was reflected or refracted off of a surface, i.e. the yellow and blue caustic photons. Furthermore we will be only storing those photons that intersect a diffuse surface after being deemed a caustic photon. This has the added benefit of the map containing more caustic photons. Instead of the full photon map containing 50,000 photons (caustic and luminance), the caustic map will contain 50,000 caustic photons. When the ray tracer begins rendering the scene, this will result in a smoother caustic. In general, increasing the number of emitted photons will result in a smoother caustic.

3 Results

Using the implementation I described above I was able to generate caustics for both reflective and refractive surfaces. Figure 2 shows some examples that were rendered using my algorithm. All scenes were rendered using 10,000 photons and were smoothed using 9x anti-aliasing


(a) Ring

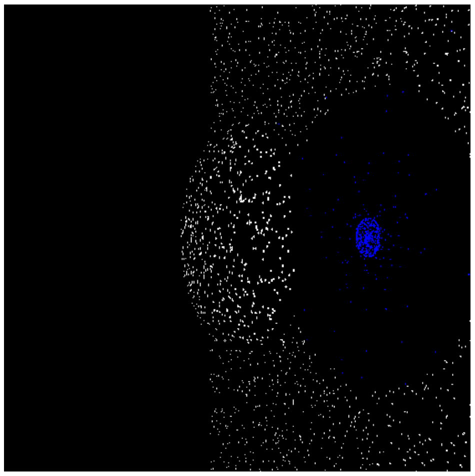
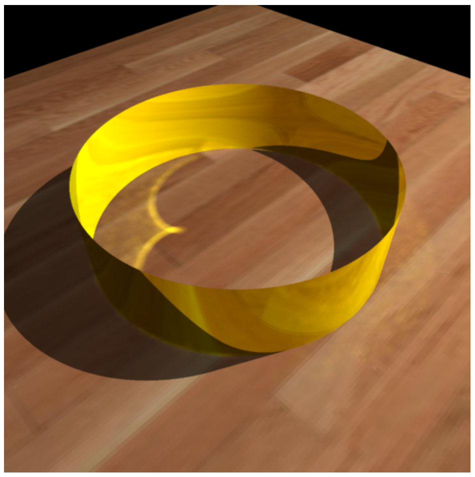

(b) Lens

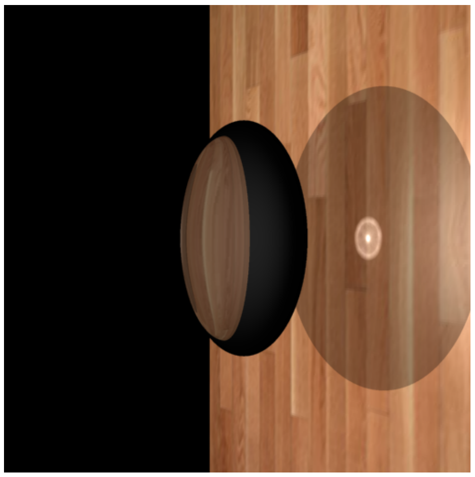
Figure 1: Example Photon Maps

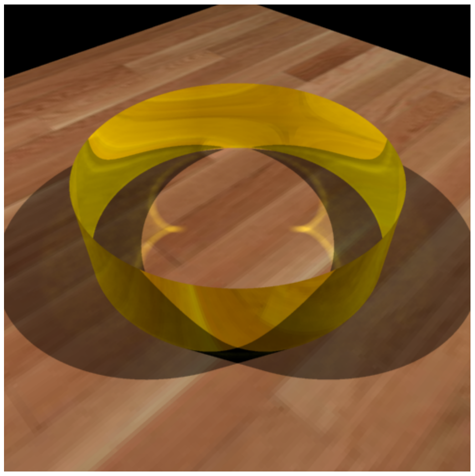
Figure 2a and Figure 2b are the full render of the photon maps from the previous section. By comparing the maps to the rendered image, you can see how well the interpolation performs. For the ring's map, you can see a particularly high concentration of photons along the edge of the nephroid. As expected this results in the edge of the caustic being slightly brighter than the rest. This effect has similar results in the lens scene.

Figure 2c was achieved using two light sources on opposite sides of the ring. For each light there were an equal number of stored photons. Resulting in a photon map containing twice as many photons as figure 2a. Each light source therefore caused a nephroid to appear on the opposite internal face of the ring.

Figure 2d is a simple scene but it was generated using a trimesh object instead of a standard box object. This illustrates that the caustics can be generated for all the implemented object types within our tracer. A interesting aspect of this scene however is the blue caustic. This illustrates that the reflected photons have the ability to pick up the color of the object which they were reflected off.


(a) Ring


(b) Lens


(c) Two Light Sources

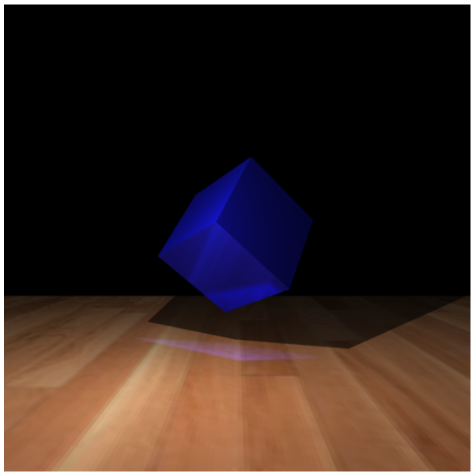

(d) Trimesh Cube

Figure 2: Rendered Caustics

4 Future Work & Conclusion

I'm quite happy with the results that I've been able to achieve using this relatively simple algorithm. However there are are number of points that could be improved with continued development. Firstly there's the issue that is slightly noticable in figure 2a. If you look in the shadow along the bottom edge of the ring, you can see a a small amount of light bleeding in from the other side of the ring. This is caused by our nearest neighbor algorithm finding photons within the area without taking into account that there is a wall between them. This could possibly be fixed by taking into account by including a shadow factor in the radiance calculation. Another issue that I could not solve was extending the photons to be emitted from directional lights. Currently caustics are only generated when point lights are used within the scene. I'm still not quite sure of why my code would not allow the use of directional light as is. The largest point of improvement is without a doubt the performance. I attempted to render figure 2a using 100,000 photons using 9x anti-aliasing and the computation took nearly 10 hours to complete. Jensen's paper suggests using a kdtree to store and retrieve values from the photon map. This would be a good starting point to improve the performance. Another would be to find a interpolation method that produces similar results using fewer photons.

The next logical step for caustic generation would be to extend the photon map to achieve global illumination. Global illumination essentially extends the interpolation to include the photons that intersected with all the objects in the scene not just the caustic photons. This would require even more photons and therefore would be best attempted after the performance issues are resolve.

Photon mapping is a very important technique in graphics rendering. Especially if the goal is to achieve photo realistic renderings. Generating caustics are just a single step toward this goal. It will probably be several year until we are able to render realistic scenes using all these methods, but researchers are moving forward everyday. And in my own opinion, ray tracing with global illumination will most likely be the final solution to this goal.

5 References

Henrik Wann Jensen: ”Rendering Caustics on Non-Lambertian Surfaces”. In Proceedings of Graphics Interface ’96, pp. 116-121, Toronto 1996.