A Better Ray Tracer

David Cohen

Advanced Computer Graphics
Rensselaer Polytechnic Institute

1. Introduction

Ray tracing is a rendering method that produces very realistic results. It works by tracing rays from the camera through the scene in a manner that simulates light. Since the rays start at the camera instead of the light this is sometimes called backwards ray tracing. In its most basic form, ray tracing produces very believable results but there are some relatively simple additions that can drastically increase how realistic the results are.

2. Refraction

The addition of refraction is a good way to allow a ray tracer to produce more realistic and believable results. Refraction occurs when an incident ray intersects with an object that is transparent. This interaction is described by Snell's law:

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

The η terms are the index of refraction of the materials, η_1 is the material that the ray is coming from, η_2 is the material that the ray is refracting into. The θ terms are the angles between the incident and refracted rays and the normal at the point of intersection.

Adding refraction to a ray tracer is relatively simple, although care must be taken to ensure that your math is correct. The equation for the direction of a refracted ray is shown below [1]:

$$t = \eta i + \left(\eta \cos\theta_i - \sqrt{1 - \sin^2\theta_t}\right) n$$

In this equation, t is the direction of the refracted ray, i is the direction of the incident ray, n is the normal at the point of intersection, and $\eta = \eta_1/\eta_2$. Since we are solving for t and do not know θ_t, we can use Snell's law to solve for the sine term and substitute the solution into the previous equation resulting in the following:

$$t = \eta i + \left(\eta \cos\theta_i - \sqrt{1 - \eta^2(1 - \cos^2\theta_i)}\right) n$$

It is important to note that in the above equations the term under the radical may be negative, resulting in an imaginary refracted direction. When this occurs it is called total internal reflection. This can only occur when $\eta_1 > \eta_2$. As the name suggests, this effect causes the light to reflect around the interior of the object and no refraction occurs.

The final addition to improve realism was the inclusion of the Fresnel equations. When a ray of light is incident to a specular surface, the amount of light that is reflected (R) and the amount that is refracted (T) are dependent on the angle between the incident ray and the normal. These amounts are given by the following equations [1]:

$$R(\theta_i) = \begin{cases} \left(R_\perp(\theta_i) + R_\parallel(\theta_i)\right)/2\,, & \neg TIR \\ 1, & TIR \end{cases}$$

$$T(\theta_i) = 1 - R(\theta_i)$$

The amount of reflected light usually depends on the polarization of the incoming light; however, most ray tracers do not polarize light so the two terms may simply be averaged. The two polarized terms are show below:

$$R_\perp(\theta_i) = \left(\frac{\eta_1 \cos\theta_i - \eta_2 \cos\theta_t}{\eta_1 \cos\theta_i + \eta_2 \cos\theta_t}\right)^2$$

$$R_\parallel(\theta_i) = \left(\frac{\eta_2 \cos\theta_i - \eta_1 \cos\theta_t}{\eta_2 \cos\theta_i + \eta_1 \cos\theta_t}\right)^2$$

2.1 Results

Figure 1 shows the results of adding refraction. The bending of light is visible at the edges of the glass bar.

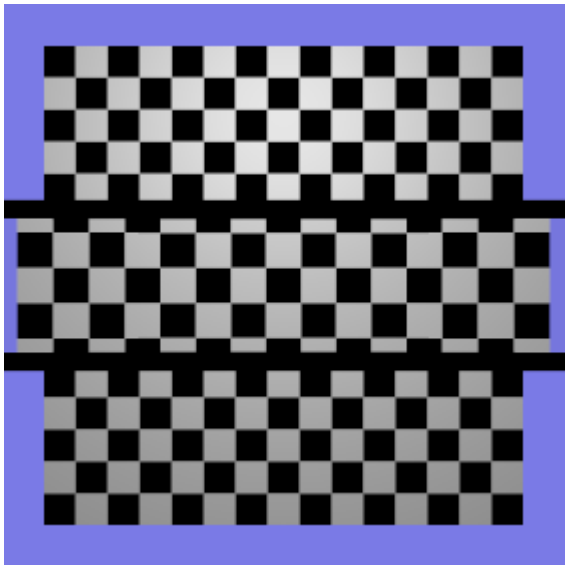

Figure 1. Refraction test image.

3. Spatial Data Structures

A good way to improve the realism of a scene is to add more detailed geometry. This is also a good way to ensure that your render takes ages to complete. A naive ray tracer must check every piece of geometry whenever a ray is cast through the scene and as such the rendering time is directly proportional to the amount of geometry. Spatial data structures offer an elegant method to solve this problem. They partition space and the geometry in that space in such a way that a ray only has to check against the geometry in the partitions that it crosses. Most spatial data structures have a pair of parameters to tune the performance, a limit to how many partitions there are, and a limit to how much geometry may be in a partition.

A slight downside to using a spatial data structure is that it introduces some overhead since the ray must check if it intersects a given partition. This overhead is rather minimal, assuming the partition intersection check is efficient, and may be overlooked in very large scenes. The same can't be said for small scenes or improperly tuned structures, the partition's bound may end up being just one more thing to check for intersection.

Kay and Kajiya introduce a fast method to check for ray-volume intersection in [2]. By representing the volume as a set of slab pairs the test for intersection becomes a couple of dot products, some multiplications and some divisions. If the normals for the slabs are constant then the dot products may be computed once for a ray and saved for following tests.

3.1 Octree

One common type of spatial data structure is the octree. The octree is a tree based structure, the volume that a node represents decreases as you move down the tree. The name comes from the fact that when a leaf is split due to having too much geometry, the volume it represents is evenly split along every dimension to produce 8 child leaves. Octrees are often used due to their simplicity while still producing good results.

A downside to octrees is that a node may encapsulate no geometry. Since a node is only split if it has geometry in it, a parent node can have up to 7 empty children. This causes some amount of inefficiency in terms of memory use. It also causes extra overhead when traversing the structure if you are not careful since the ray will check if it intersects this node even though doing so will not change the result.

When implementing an octree it is important to consider geometry that spans across multiple nodes in the tree. If geometry is only stored at the leaf level then one object may be stored in many nodes, which is a waste of memory. To alleviate this, geometry can be stored at every level of the tree. This way, if an object is inside more than one node it may be stored a single time inside the parent node.

3.2 KD-Tree

Another common, tree based spatial data structure is the kd-tree. The name stands for k-

dimensional tree. It is very similar to the octree except for how it splits. When a node in a kd-tree is split it is split along one axis, producing two children. The axis can be chosen in many ways but it is usually chosen to be the longest axis represented by the node. The point along the chosen axis at which it splits if often chosen to be the midpoint.

Like the octree, the kd-tree can also suffer from storing empty space. Unlike the octree, a kd-tree will store at most one empty child per node since each node has only two children so the problem is not as severe. The same solution applies in regards to objects that span multiple nodes.

3.3 Results

Figure 2 shows how the addition of a spatial data structure can greatly reduce the time it takes to render a complex scene. The speedup compared to the naive ray tracer was about 3.3 for the octree and about 3.8 for the kd-tree.

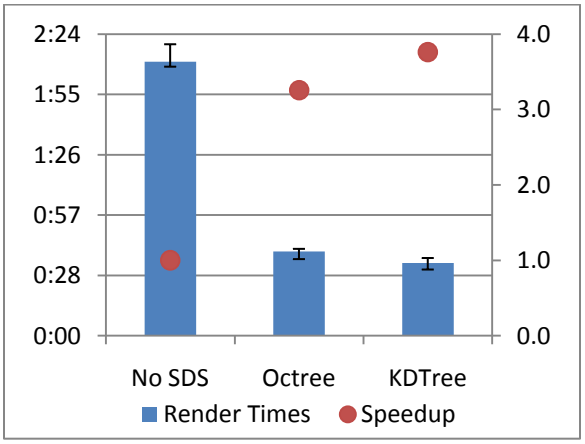

Figure 2. Rendering time and speedup with a spatial data structure on a large scene. Left axis is time in hh:mm. Right axis is speedup.

Figure 3 indicates that using a spatial data structure on a scene with a low amount of geometry can increase render time due to the extra overhead.

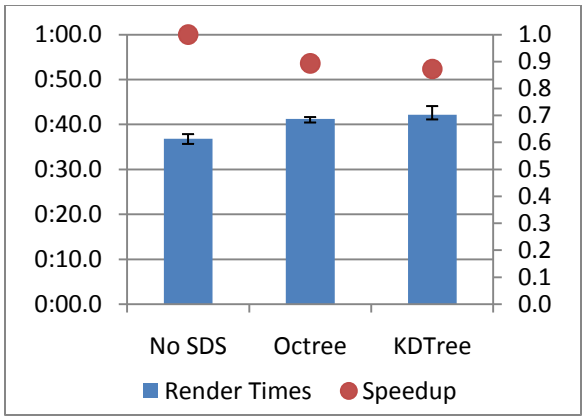

Figure 3. Rendering time and speedup with a spatial data structure on a small scene. Left axis is time in m:ss. Right axis is speedup.

4. Photon Mapping

Basic ray tracing leaves out many complex lighting effects that can be seen in the real world. One such effect is the caustic. A caustic is produced when a specular object focuses light onto another surface. Another effect is global illumination where a surface may be receiving light indirectly from the light source. Photon mapping aims to simulate both of these effects. It works by tracing photons from light sources as they bounce around the scene. Whenever a photon strikes a surface it is recorded into the photon map. The map is then used when rendering a scene to look up how much indirect light a given point is receiving.

It is important that the photons traced through the scene have an appropriate intensity. If the photons are too strong then the effects photon mapping aims to simulate will be overpowering, but if they are too weak then there may be no visible result. The power of a photon should depend on the power of the light emitting it, and how many photons that light is emitting [3]. If there are multiple lights in the scene then each light should emit a number of photons proportional to the strength of the light compared to the combined strength of all the lights.

When gathering the indirect lighting it is necessary to find the nearest photons to the

given point. Therefore it makes sense to store the photons in a structure like the kd-tree that allows for fast nearest neighbor lookup. Sometimes when gathering there will not be enough photons in the area which can produce fuzzy and incorrect caustics. It is suggested that a filter is used on all gathered photons to help smooth out the noise [3]. One such filter is the cone filter which assigns a weight to a photon based on its distance to the point that indirect light is being gathered for. The cone filter is defined by a filter constant, $k \geq 1$.The weight, w, for a photon with a distance d from the gather point, with a gather radius r is given by the following equation:

$$w = 1 - d/kr$$

In order to normalize the result from the gathered photons, the result is divided by $1 - 2/3k$.

4.1 Simple Optimizations

The addition of photon mapping can cause rendering times to skyrocket. Tracing the photons through the scene takes time proportional to the number of photons and how much geometry is in the scene and gathering indirect lighting can be very slow if you are trying to gather a large number of photons. Luckily, there are some simple optimizations that can provide significant increases in performance.

The first optimization relates to the tree used to store the photons. Even though kd-trees are known for their very good nearest neighbor lookup, the actual performance depends heavily on how well balanced the tree is. If care is taken to balance the tree then performance gains as large as 50% may be seen [4]. To balance the tree, one could add all of the photons at once and when splitting a node choose the median along the split axis. This ensures that half of the photons will be on each side of the split, thus creating a balanced tree.

The second optimization is to the gathering of indirect lighting. Since you want to find the n nearest neighbors to a point, some sorting will eventually need to be done. The naive approach is to sort everything you find but this is a poor choice since you may be sorting large numbers of photons. While gathering photons you are only interested in ones that meet certain criteria, such as the direction of that photon being within the hemisphere around the incident ray. Using these criteria you can cull the set of photons and greatly reduce the number of photons that need to be sorted.

The third optimization is also the simplest optimization. Gathering photons involves sorting them by their distance to a point. Calculating this distance can be expensive due to the square root. Instead of calculating the distance to the point every time you compare two photons, it would be beneficial to compute the distance once and store it for future comparisons.

4.2 Results

Figure 4 shows the caustic formed by a reflective ring.

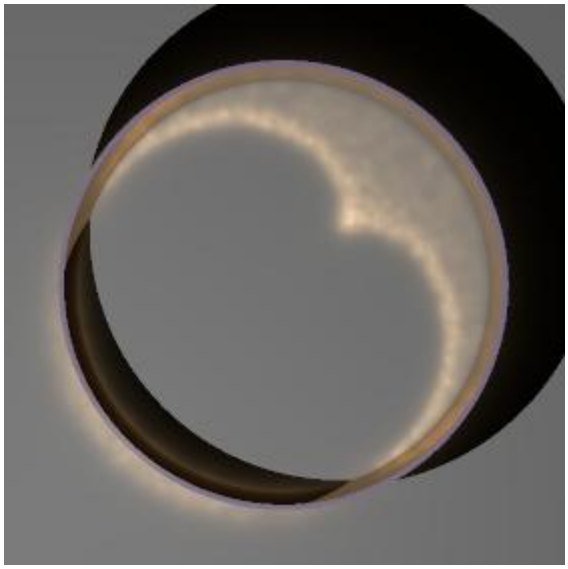

Figure 4. Caustic caused by specular reflection.

Figure 5 shows the caustic formed by a glass sphere.

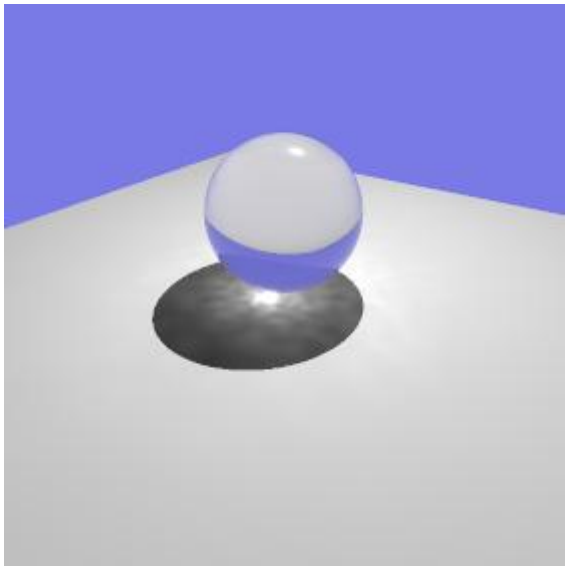

Figure 5. Caustic caused by specular transmission.

Figure 6 shows how changing the value of k in the cone filter affects the resulting caustic.

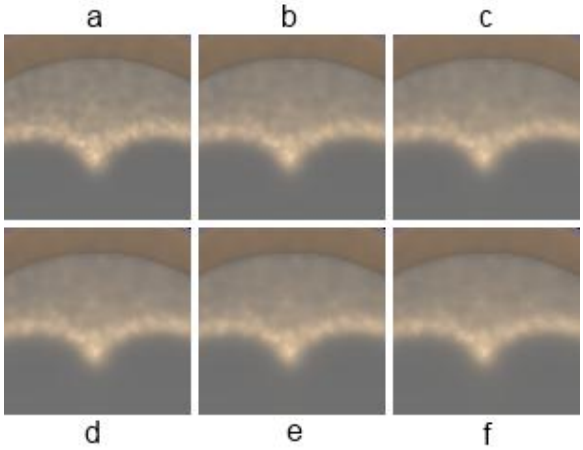

Figure 6. The effect of varying the k in the cone filter. a = 1.00, b = 1.25, c = 1.50, d=1.75, e=2.00, f=10.00.

6. References

[1] Bram de Greve. "Reflections and Refractions in Ray Tracing." November 13, 2006.

[2] Timothy L. Kay, James T. Kajiya. "Ray Tracing Complex Scenes." ACM Siggraph 1986.

[3] Wojciech Jarosz, Henrik Wann Jensen, Craig Donner. "Advanced Global Illumination Using Photon Mapping." ACM Siggraph 2008.

[4] Henrik Wann Jensen. "Rendering Caustics on Non-Lambertian Surfaces." Proceedings of Graphics Interface 1996.

7. Other Images

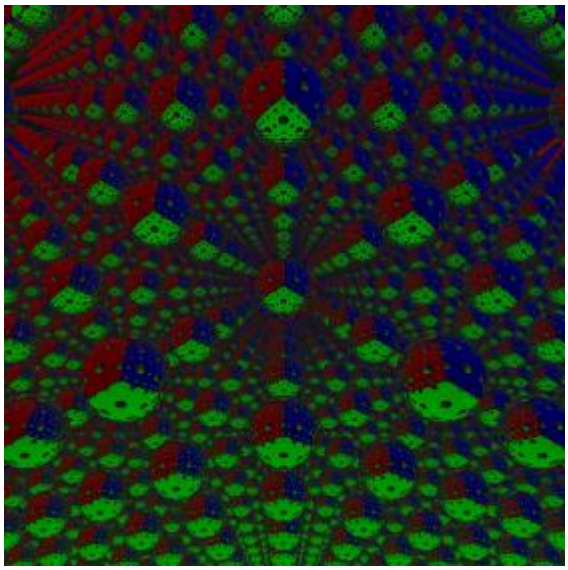

Figure 7. The scene used to create Figure 2. There are 8000 spheres in the scene.

Figure 8. The scene used to create Figure 3. There are 5 spheres in the scene.