Enhanced Ray Tracing

Participating Media by Brian Stauffer

Rainbows by Max Curran

Gems by Gary Lu

Motivation

Ray tracing and photon mapping can render realistic results. We want to implement accurate wavelength dependent effects to properly render materials like crystal and rainbows, and extend photon mapping for refractive caustics and participating media.

Rendering the interaction between light and microscopic particles in a scene, such as smoke or floating dust, has traditionally been a computationally intensive and difficult process. To produce an accurate rendering requires simulation, or reasonably accurate approximation, of how the light interacts with these particles on a fine level of detail. Specifically, when light enters an area of some participating media, it will travel until it hits a particle, then it is scattered in another direction and travels until it hits another particle, where it is scattered again, until the light escapes the media.

Rainbows are one of the more visually impressive phenomena in nature. They are caused by refraction and dispersion in small water particles in the atmosphere. This simple explanation masks all of the complexity which makes them an active area of research. Many of the characteristics of rainbows are not fully captured by refraction and dispersion. For example, the supernumerary bows, those which occur beneath the primary bow, is caused by interference of light.

Related Work

Participating Media

Rendering light's interaction with participating media is similar to modeling subsurface scattering in partially translucent materials such as marble or milk, which is often rendered by approximating the solution to the volume rendering equation [Jensen 2001]. However, this approximation is quite computationally expensive, and does not easily work with photon mapping techniques that are often used for many other lighting conditions. Fortunately, Jensen et al. have proposed a method for using volumetric photon mapping to render light’s interaction with participating media [Jensen 1998].

Jensen et al. proposed a method for extending the photon mapping algorithm to properly handle photon interactions with participating media through use of volumetric photon mapping [Jensen 2001]. The key difference between the old method of photon mapping and this new method is the addition of a second map for photons to be stored in: the volumetric photon map. In typical photon mapping simulations, photons are stored at the surfaces that they interact with, but when involving participating media, photons must also be stored in the empty space between surfaces wherever an interaction between a photon and a particular particle took place[Jensen 2004]. Simply put, these photon interactions that do not occur at a surface are stored in the volumetric photon map.

Rainbows

Sadeghi, Munoz, Laven, Jarosz, Seron, Gutierrez, and Jensen [2012] published a paper in which they tabulated data by running a simulation of light interacting with a single water droplet. Their simulation took into account dispersion, interfere, diffraction, polarization, and differing shapes of water droplets into account. This lead to extremely accurate renders which mimicked many unusual phenomena which could not be reproduced by earlier algorithms.

Their simulation consisted of shooting rays at a water droplet then collecting them on a sphere around the water droplet. This is done for thirty three distinct wavelengths. Each ray also keeps track of its parallel and perpendicular polarization. After all the rays for one wavelength are cast, the points on the collecting sphere where the rays ended on are made into quad patches. After this is done for every wavelength, rays are then cast from the center of the sphere to the surface. Data is then interpolated using all patches the ray intersects. The value of each direction is then saved for use during a render.

The goal was to recreate this algorithm; however, this was not achieved. Instead approximations were made to render rainbows. All of the approximation come from physics and optics, whereas the above algorithm is a simulation. This means the approximations make assumptions which limit what phenomena they can render.

The method the paper was directly trying to improve on was the Lorenz-Mie theory. This provides an analytical solution to light scattering through a sphere. This solution can be used to render rainbows only with spherical water droplets. However, due to gravity and water tension, the shape of water droplets become less spherical as they increase in size.

Algorithms

Fresnel Reflection

When light from one medium to another, both reflection and refraction might occur. Viewing a surface from a grazing angle gives off more specular reflection compared to viewing the surface directly.

The Fresnel equations describes what amount of light is reflected and what amount is refracted. The equations are based off of the law of reflection and Snell's law, and depends on the polarization of the incident ray.

For our purposes, we used Schlick's approximation to approximate the contribution of the Fresnel term in the specular reflection of light from a non-conducting surface.

Schlick's approximation for calculating R, the reflectance coefficient:

$$R(\theta) = R_0 + (1 - R_0) * (1 - \cos(\theta))^5$$

Where R_0 is the reflection coefficient for light parallel to the surface normal, given by:

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

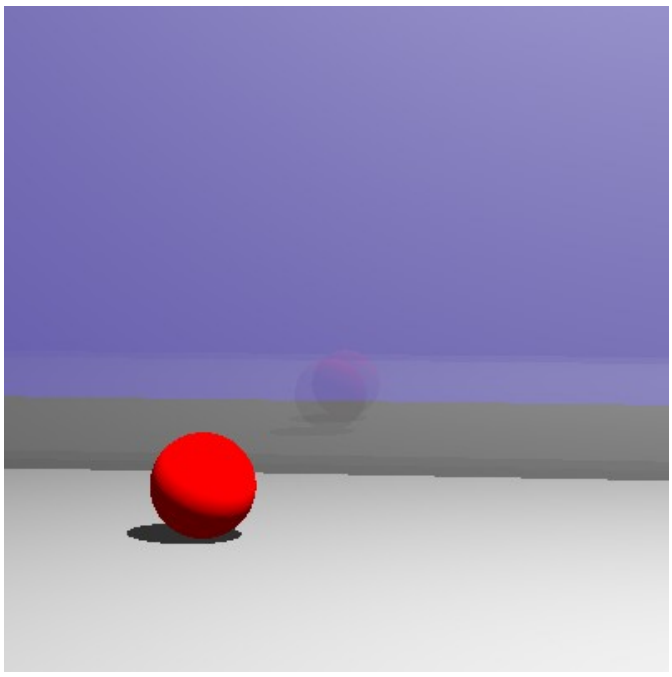
θ is the angle between the incoming ray and the normal to the surface it collides with. n1 and n2 are the refractive index of the incoming and outgoing materials.

When viewing the surface parallel to the surface normal, the reflection coefficient is the small R_0, and as the viewing angle approaches 0, the coefficient approaches 1 rapidly.

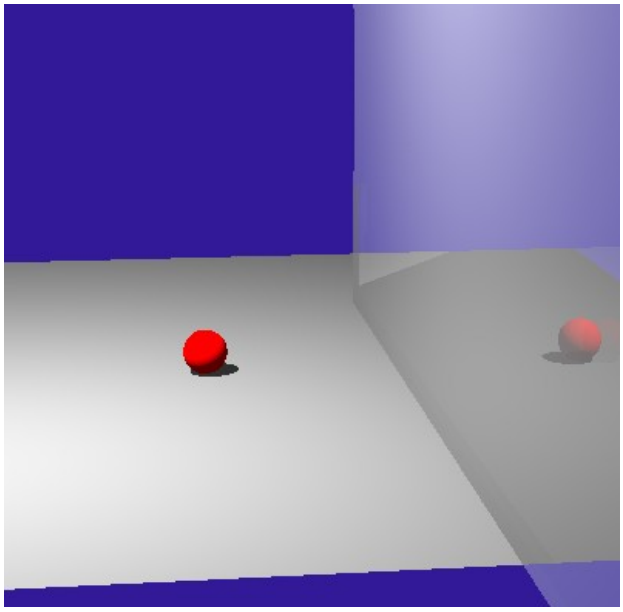
After receiving the color contribution from ray tracing, we multiply the color by R. Due to conservation of energy, the transmission coefficient T is given by:

$$T = 1 - R$$

Likewise, we multiply the refraction contribution by T.



Viewing the reflection directly results in a faint reflection. Note that the reflecting object has transparency and depth; the second reflection is due to rays bouncing off the back wall.



Viewing the wall at an angle results in a clearer reflection.

Note that while a material must have a refractive index for Fresnel reflection to work, it does not actually need to be transparent. If the material has no refractive properties defined, we default back to the original reflection by setting R to 1. The effects of Fresnel reflection are most noticeable with materials that have a slightly higher index of refraction than air. Materials with higher refractive indices tend to overwhelm the reflection, and reflections are then only visible at extreme angles.

Refraction

When light hits a material with a different refractive index, its phase velocity changes, giving off the appearance of bending. Snell's law describes the phenomenon as the relationship between angle of incidence and refractive index.

$$\frac{sin\theta_1}{sin\theta_2} = \frac{n_2}{n_1}$$

Where θ_1 is the angle of incidence, and θ_2 is the outgoing angle of incidence. n_1 is the refractive index of the material the ray is leaving, and n_2 is the index of the material the ray is entering.

The directional vector of the refracting ray can be calculated with:

$$\boldsymbol{v}_{refract} = \frac{n_1}{n_2}\boldsymbol{l} + \left(\frac{n_1}{n_2}cos\theta_1 - cos\theta_2\right)\boldsymbol{n}$$

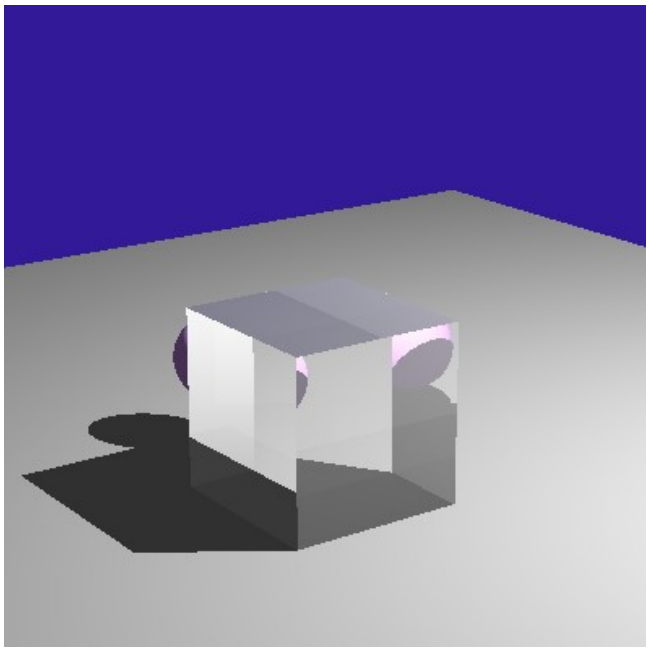
Where **l** is the normalized directional vector of the incoming ray, and **n** is the normal vector of the surface the ray hits. $\cos\theta_1$ and $\cos\theta_2$ can be found by:

$$cos\theta_1 = -\boldsymbol{n}\cdot\boldsymbol{l}$$

$$cos\theta_2 = \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 \left(1 - \left(cos\theta_1\right)^2\right)}$$

Note that $\cos\theta_1$ must always be positive, multiplying by -1 if necessary. If the radicand under the equation for $\cos\theta_2$ is negative, it indicates total internal refraction. In this case, the ray reflects off the object completely, which can be calculated with:

$$\boldsymbol{v}_{reflect} = \boldsymbol{l} + \left(2\cos\theta_2\right)\boldsymbol{n}$$



A transparent cube refracts light with a pink sphere behind the cube.

Dispersion

Refraction is wavelength dependent, so the angle of refraction depends on the wavelength. Since white light is composed of multiple frequencies of light, it appears to split into different colors as each wavelength of light refracts in their own directions.

In our implementation, we use multiple dispersion samples to calculate dispersion colors.

When a not-yet-dispersed ray strikes a transparent object, we split the ray into multiple

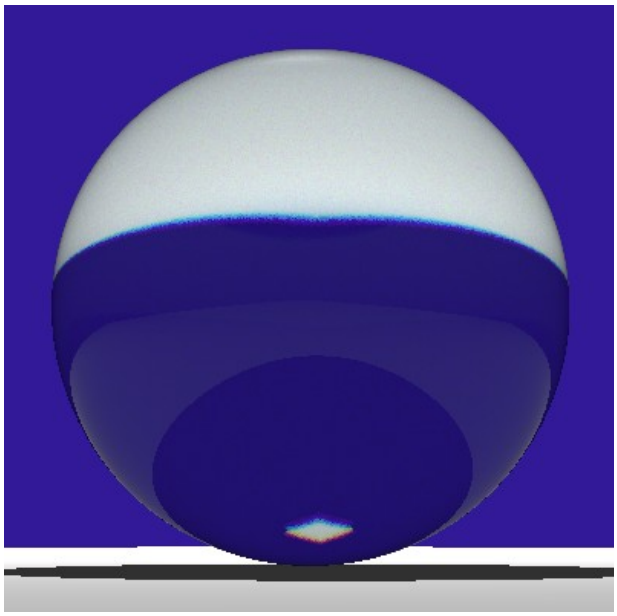
monochromatic rays. We partition the range of wavelengths into sections depending on the number of dispersion samples we use, and choose a random wavelength from each section.

We used Cauchy's equation to approximate the index of refraction as a function of wavelength.

$$n(\lambda) = B + \frac{C}{\lambda^2}$$

Where n is the index, λ is the wavelength, and B and C are coefficients determined by the type of material. Cauchy's equation just serves as an approximation, and is only accurate in the visible spectrum of light. B while technically incorrect, Cauchy's equation is a good enough model for computer graphics dealing with visible light.

After shooting out multiple dispersion rays, we convert wavelength into an RGB color scheme, and average the colors together.

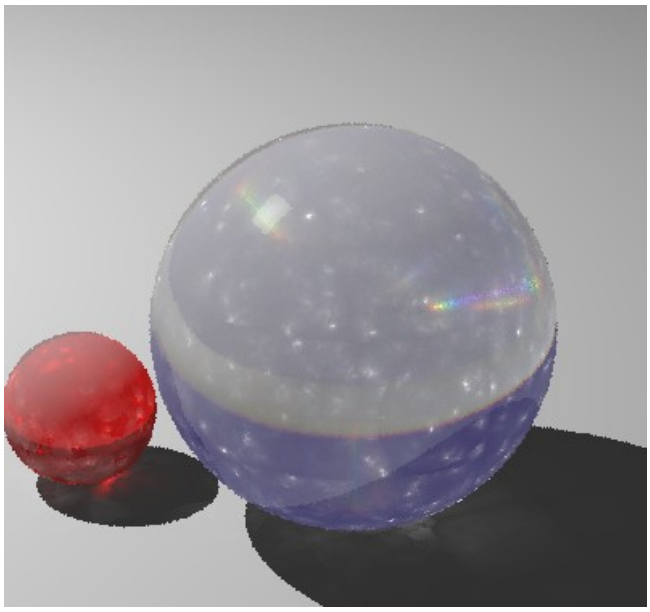


A light source far above the sphere is visible through the sphere, which has an extremely high refractive index.

Photon Mapping

For the most part, refraction and dispersion works the same for photon mapping as it did for ray tracing.

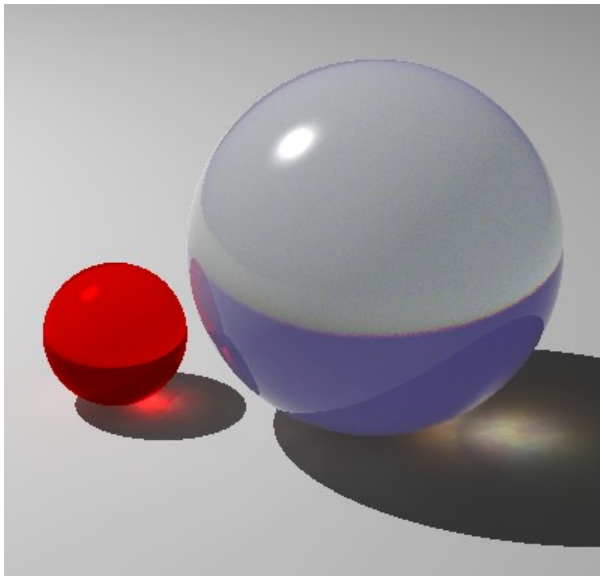
Remember that total internal reflection occurring inside in object should not be visible, so do not add a photon to the kd-tree when a photons reflects off the inside of a refractive object.



A sphere with visible internal reflections

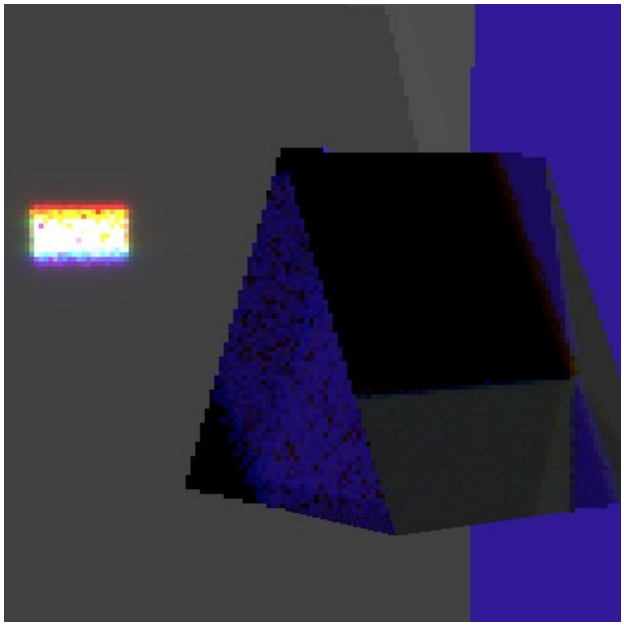
For ray tracing dispersion effects, we had to jitter the wavelengths and send out multiple dispersion rays of different wavelengths. For photon mapping, just sending a single photon of a random wavelength should be fine. This is because when gathering the indirect illumination when ray tracing with photons, the function gathers the nearest photons. As long as a large enough number of photons are collected, the color returned should be accurate.

For example, if a red, green, and blue photons are shot towards the same point, gathering the indirect illumination of that point should return white. The way we chose RGB values from wavelengths is an approximation, so sometimes when an area should be white, the color is slightly pinkish. But for the most part, the colors are accurate enough.

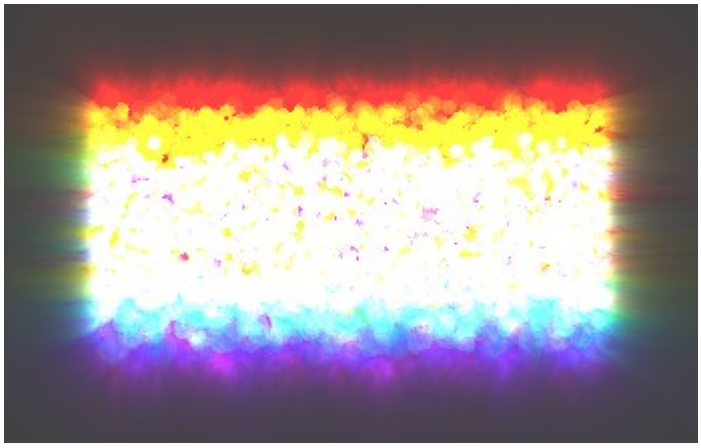


Transparent spheres with caustics

When not enough photons are collected when calculating indirect illumination, the colors may appear splotchy. This can be fixed by increasing the number of photons collected, or dramatically increasing the number of photons fired. The accuracy of photon mapping depends on the parameters set, and the constants chosen for material properties.



Incomplete render of prism caustics



Dispersion effects appear splotchy due to low number of photons collected

Participating Media

The problem that arises with volumetric photon mapping is how to determine if a photon interacts with a participating media. Of particular importance is the density of the media, as a more tightly packed cluster should lead to a higher amount of interaction. Given a photon traveling in direction *r* from origin point *o*, we want to be able to find a plausible value *d* such that the photon interacts with the participating media at *o* + *dr*. This can be accomplished by using the extinction coefficient of the media, which is essentially a parameter that describes how much light a substance absorbs. With the extinction coefficient σ_t, and a random variable χ between zero and one, such a value *d* can be calculated as:

$$d = \frac{-\log(\chi)}{\sigma_t}$$

This works perfectly for homogenates media, and can be extended to also support heterogeneous media by varying the extinction coefficient per location in the media, so that light is absorbed more in certain places than others [Wojciech 2008]. This kind of approximation is less accurate, but often produces visually acceptable renderings.

Next, once a point of interaction for the photon is determined, the photon is stored in the volumetric photon map, and then either absorbed or scattered from the point. As is usual for typical photon mapping, this interaction is chosen by various probabilities of the media. In this case the probability of the photon being absorbed is the ration of σ_a to σ_t where σ_a is the absorption coefficient of the substance. This makes the

probability of the photon being scattered *1-pr*$_a$, and the scattering coefficient $\sigma_s=\sigma_t-\sigma_a$. Given these probabilities, the type of interaction for a photon is chosen by Russian roulette, and either tracing photon either of the photon terminates, or a new scattering direction is computed depending on the media, and the photon continues to be traced through the scene.

For a photon traveling through a medium, it will travel in it's direction until one of two things happens: it interacts with the participating media, or it hits a surface. In order to accommodate for both of these options, a ray is cast from the photon at position *o* in the direction it is traveling, *r*, to determine what surface that it would hit if it didn't interact with the participating media. This produces a value *t*, such that the surface the ray hit is at $x = o + tr$. Then, a value for *d* of where the photon would interact with the media is calculated from the above formula. If $d < t$, then the photon interacts with the media at position $x = o + dr$, otherwise, it interacts with the surface at $x = o + tr$.

If a photon interacts with a surface, it is stored in the surface photon map as normal, but if it interacts with the participating media, it is stored in the volumetric photon map. There are a few reasons why these maps are kept separate. First, keeping them separate means that they can be calculated independently at different densities. If more detail is needed for the volumetric map than the surface map or vice versa, photons can be traced through the scene multiple times with different amounts in order to populate the maps with the desired density. Typically, however, the volume and surface photon maps are constructed at the same time. Second, when rendering the scene, it is incorrect to use the volume photons to light on a surface, or to use surface photons to render light in the volume. Finally, it leaves the surface photon map less cluttered, so those lookups are much more efficient [Jensen 1998].

Estimating radiance at a particular point in the volume using these photon maps can be viewed as a simple extension to the surface radiance estimation [Jensen 2004]]. The estimate at a point *x* in the direction *w* is the integral around all incoming directions of the radiance incoming radiance in that direction multiplied by the bi-directional reflectance distribution function at that point between those directions:

$$L_i(x,\vec{w})=\int_{\Omega} f(x,\vec{w}',\vec{w})L(x,\vec{w}')dw$$

This particular estimate is infeasible to use to precisely calculate the radiance, but can be estimated using the constructed photon map. By using the derivation of Jensen et al., this exact calculation can be reduced to the approximation:

$$L_i(x,\vec{w})=\frac{1}{\sigma_s(x)}\sum_{1}^{n} f(x,\vec{w}_p)\frac{\Delta\Phi_p(x,\vec{w'}_p)}{\frac{4}{3}\pi r^3}$$

Here, $\sigma_s(x)$ is the scattering coefficient of the medium at *x*, or $\$\sigma_s(x) =\sigma_t(x) - \sigma_a(x)$, Φ_s is the in scattered radiance of the photon *p* stored in the photon map, and *r* is the radius of the smallest sphere centered at *x* that surrounds all *n* closest photons [Wojciech 2008]. The primary difference between the surface radiance estimate and this is that the volume estimate uses photons in three dimensions, so the radiance estimate is divided by the volume of the enclosing sphere, rather than the area of the enclosing disc.

In order to render the scene, this radiance estimate must be applied along the eye direction to accumulate the surface radiance estimation as well as the contribution of radiance from the media [Jensen 2004]. This can be done using a simple ray marching algorithm along the ray from the eye point to the point where the ray intersects a surface. The idea behind this ray marching algorithm is to evaluate the radiance estimation of the media along evenly spaced points along the ray, as in figure X. At each discrete point along the ray, the volume estimated radiance is calculated using the above formula. Each of these estimates is combined and then added to the surface radiance estimate and color by:

$$L(x,\vec{w})\approx L(x_s,\vec{w})+\sum_{t=0}^{S-1}\sigma_s(x_t)$$

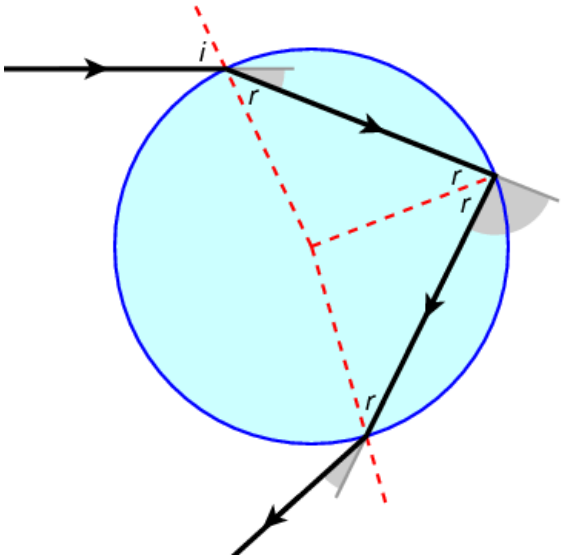
where $L(x_s,\vec{w})$ is the radiance approximation at the surface point, and each radiance term in the sum are the volumetric radiance estimates taken along the ray [Jensen 1998].

Rainbows

Descartes

Four hundred years ago, Rene Descartes produced the first detailed study of rainbows [Casleman]. Descartes figured out rainbows are caused by the bunching of rays after they enter a water droplet reflect some number of times then exit the sphere. The primary bow is caused by one reflection within the sphere. For any ray that

enters the water droplet with an angle of *i* as shown:



The total deflection is therefore:

$$D = 4\mathrm{r} - 2\mathrm{i}$$

Then by using Snell's law:

$$\sin(i) = n\sin(r)$$

$$r = \arcsin\left(\frac{\sin(i)}{n}\right)$$

$$D = 4\,arcin\left(\frac{\sin(i)}{n}\right) - 2i$$

The rainbow is caused by the bunching up of rays to form a caustic ray. This can be found by finding the minima of *D* as a function of *i*. Therefore the angle *I* of the caustic ray is:

$$i = \arccos\left(\sqrt{\frac{n^2-1}{3}}\right)$$

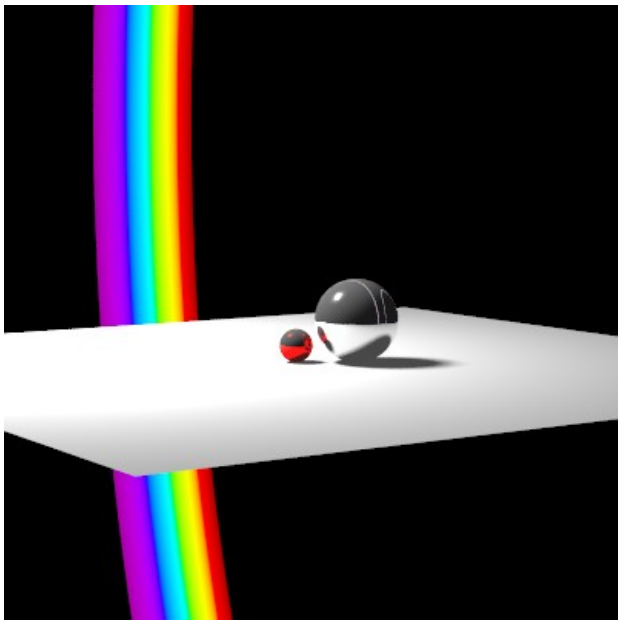
The angle of deflection can then be found by plugging this value back into the equation for *D*.

This results in the finding of the angle of the caustic ray, but does not directly explain why a rainbow shows the spectrum of color. This is due to the fact that different wavelengths have slightly different indexes of refraction. This causes the angle of the caustic ray to be slightly different causing the rainbow to show the entire spectrum.

This also only accounts for the primary bow, because the angle was only calculated for one reflection. The formulas only need to be slightly altered to find the caustic ray after two reflection, resulting in the secondary bow. This process can be continued for higher order bows; however, they are almost never visible, because they are near the sun, and have very little energy.

In order to use this to render an actual rainbow using Descartes equations requires a little more work. The equations work with light coming from the sun, whereas ray tracing works with rays coming from the camera. In order to reverse this process there are two options. The first is to compute the wavelength based on the angle between the incoming direction and the direction to the sun.

T



Secondary bow, made by solving for wavelength

The image above shows a rendering using this method. By using Descartes equations, it is possible to solve for *n* given *D*, by using the following equation:

$$n = \frac{1}{2}\sqrt{(2(k+1)^2-2)\cos(D)+2(k+1)^2+2}$$

,where *k* is indicates how many reflections.

Then by using the approximation:

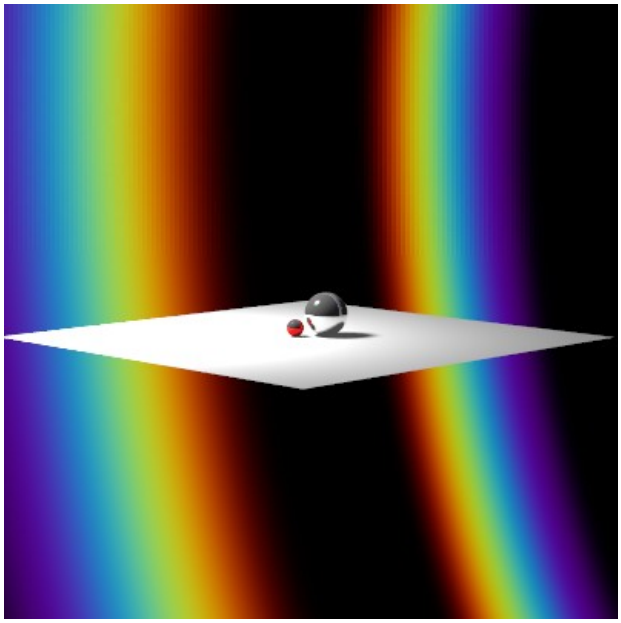
$$n = B + \frac{C}{\lambda^2}$$

it is possible to find the wavelength of the caustic ray, where *B* and *C* are constants based on the material.

The other method that makes use of Descartes equations is to samples across the spectrum, and for each of these wavelengths compare the actual angle of deflection to the caustic angle of deflection. Then take a contribution of the color of the difference is

within some bound. The color should also be weighted that a difference of zero corresponds to one, and as the difference increases in magnitude the weight goes to zero.

This method introduces noise and blending, which causes the rainbow to appear more accurate. In an actual rainbow, these appearances are caused by diffraction and polarization, but this added noise recreates similar visual results as shown below:



Sampling with Descartes

Airy

Descartes work was before light was well understood. After Fresnel came up with his wave theory, George Airy applied it to spheres. His integral explained that interference caused the supernumerary bands. When the wave of light leaves the sphere the light is in an *S* shaped front, which intersects itself, which causes interference. Airy solved this problem with the integral of

$$A(m)=\int_{-\infty}^{\infty}\cos\left(\frac{\pi}{2}(s^3-ms)\right)ds$$

where m is proportional to the angle from the caustic ray.

This integral has no analytical solution, but can be approximated by a Taylor series. This was first performed by De Morgan, who came up with the following approximation:

$$\overline{A}(x)=A(0)\left(1-\frac{1}{3}\frac{x^3}{3!}+\frac{1\cdot4}{3^2}\frac{x^6}{6!}-\ldots\right)$$
$$+A'(0)\left(x-\frac{2}{3}\frac{x^4}{4!}+\frac{2\cdot5}{3^2}\frac{x^7}{7!}-\ldots\right)$$

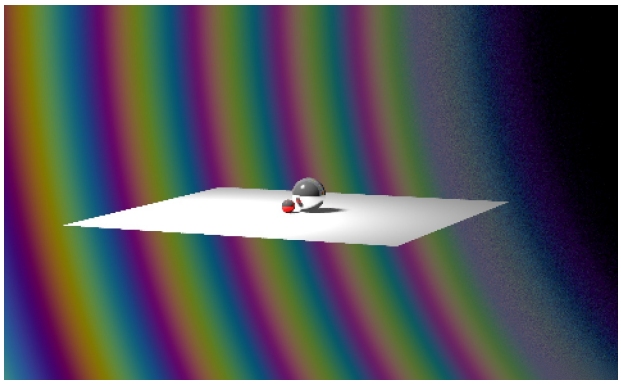
where ,

$$A(0)=\frac{1}{3^{\frac{1}{3}}*\Gamma\left(\frac{1}{3}\right)}$$

$$A'(0)=\frac{1}{3^{\frac{2}{3}}*\Gamma\left(\frac{2}{3}\right)}$$

This approximation is only good for values between -4 and 5. Anything outside these bounds will have great error from the real value of the integral.

To render a rainbow using they Airy integral, a truncated version of De Morgan's approximation must be used. Then Descartes equations must be used to find the deflection angle, and the difference between this and the viewing angle must be used in De Morgan's approximation to find the weight. The proportion to make a realistic rainbow as not found, but a proportion which rendered the desired affect, supernumerary bows, was used. As shown below, this was achieved:



Supernumerary Bows using the De Morgan Approximation

The frequency and amplitude of these bows is too high. Also the artifact on the right is caused by error in the De Morgan approximation. The render does show that as the bows go away form the primary bow, they do start to blend together and become less intense. To render the bows more accurately, the proportion from the difference to the value passed to the De Morgan approximation must be tinkered with.

Lorenz-Mie

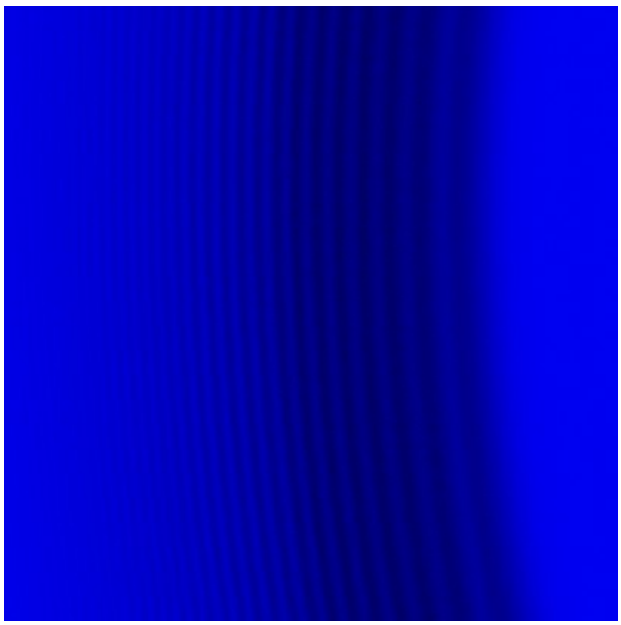
Airy's integral did not solve the problem to light scattering through a sphere, it was an approximation itself. The solution to light scattering in a sphere was first published by Mie in 1908. His solution took into account the full behavior of light.

To render a rainbow using this solution, I made use of existing code which solved the equations, because the Physics and Mathematics behind the solution were too in depth for my knowledge. The implementation I used was based on papers by Hong Xue[2004] and Wiscombe[1980].

In order to render anything using these calculations in any reasonable amount of time, the data must be precomputed on lattice points. Then during rendering the data can be interpolated from these lattice points to find the values at intermediate values.

The calculations are still intensive and create large data files. These files can then be read into the renderer and then lookups can be performed quickly. Unfortunately, I was unable to make a good render of a rainbow from the data, because the Mie solution deals with light intensity. Any rainbow I was able to make was too spotty and dark, or had too much background noise.

Much of the interesting data, such as supernumerary bows, were hard to differentiate from most of the data. The render below mapped values from the max value to the average value from red to black, and the average value to the minimum value from black to blue for one wavelength on a logarithmic scale:



Render of range of values using Mie solution

The small variations have a clear pattern, but I was not able to separate them to render a rainbow. Also, these values oscillate between the average and below average. The maximum value occurs at theta equal to zero, not in the rainbow.

Results

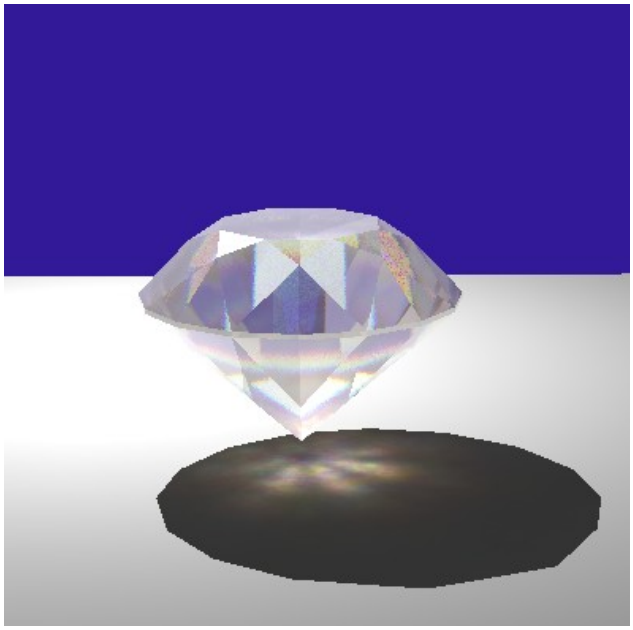
Gems

Fresnel reflections and refractions look accurate for both ray tracing and photon mapping.

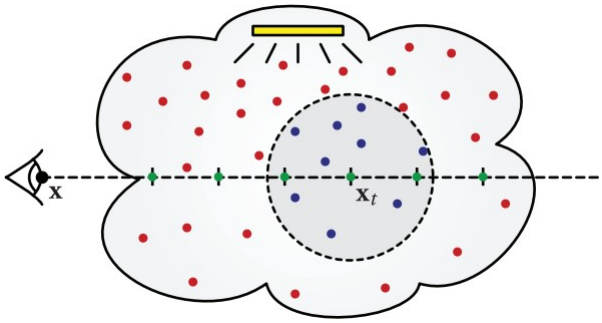
With dispersion, the images contained lots of noise during ray tracing, which could be remedied by either increasing sample sizes or using anti-aliasing, both of which would dramatically decrease performance.

With dispersion, colors might seem off on occasion due to the random nature of both the ray trace and the photon mapping. This could be fixed with a more accurate conversion of wavelength to RGB value.

Renderings using photon mapping cause a huge hit in performance, and may take hours to render depending on the scene, without anti-aliasing or soft shadows.



Final render of a diamond



Ray marching through participating media. Volume radiance estimates are computed at discrete evenly spaced points along the ray to approximate the radiance along that ray. (Figure taken from [4]).

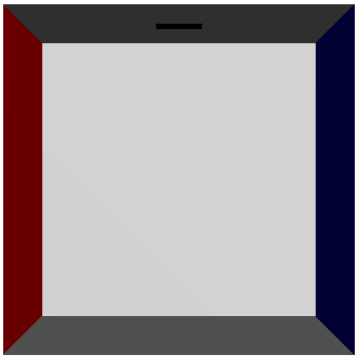
Participating Media

Running this algorithm on a number of test scenes show promising results with a few obvious flaws. Figures X-Y show scenes that render a) only primitives, b) only the radiance contribution from participating media, and c) the final image. The radiance contribution from participating media shows almost exactly what would be expected: a distribution of light throughout the scene as if there is some dust-like substance in the room clustered closer to light sources. In particular, in the cornell box scenes, even the diffuse light reflected off of the walls is visible in the media, though it gets lost in the final render.

While the results look promising, there are noticeable problems with them, namely the noise in the render. Using this algorithm, isolated bright spots are common, especially when a ray shot through a pixel comes in close contact to a surface at a grazing angle. This could have resulted for a few reasons. Using ray marching chooses evenly spaced points to gather light from, but these points aren't necessarily analogous between adjacent pixel rays. This can lead to one or two points along a ray that "get lucky" and hit a dense pocket of photons, where neighboring pixels might be too far from these (in the ray direction) to be quite as bright. Alternatively, any number of bugs could have been introduced, including incorrect calculation of photon energy at an intersection or incorrect gathering of photons.

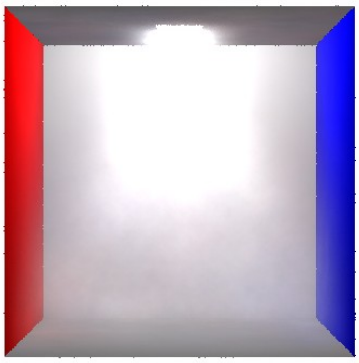
In addition to these bright spots, what appear to be dark "artifacts" also appear in the images. These may be caused by similar problems to the bright spots, but appear to be less isolated. Instead, these dark spots may in fact be caused by the randomness of photon mapping, where an area might be darker simply because not enough of the photons happened to get stored there, and shooting more photons in the scene may help to address this problem.

In terms of run time, the results are much less exciting. Even with only 10,000 photons shot into the scene, 100 photons gathered at each radiance estimate, and only 5 samples taken in the ray marching algorithm, even the simple scene in figure X took 64 minutes and 36 seconds to render. This is due to the nature of ray tracing being a very computationally intensive algorithm in addition to ray marching adding an additional level of complexity. Gathering the n nearest photons to a point is at least $O(n)$ for a very good gathering algorithm. Taking m samples in the ray marching algorithm means that for each ray, it takes $O((m+1)n)$ time to approximate the radiance (m for the points along the ray, plus the point on the surface). Combining this with the fact that there is one ray per pixel, with p pixels it takes $O(cp(m+1)n)$ time to compute, where c is an additional coefficient based on how much other work is done per ray. Unsurprisingly, this complex of an algorithm takes enormous amounts of time to compute.



Simple scene with Cornell Box and directed light overhead. Rendered tracing 10,000 photons, using 5 sample points for the ray marching algorithm, and an extinction coefficient of 0.5. The base geometry to be rendered.

The contributing radiance from the participating media.



The final render.

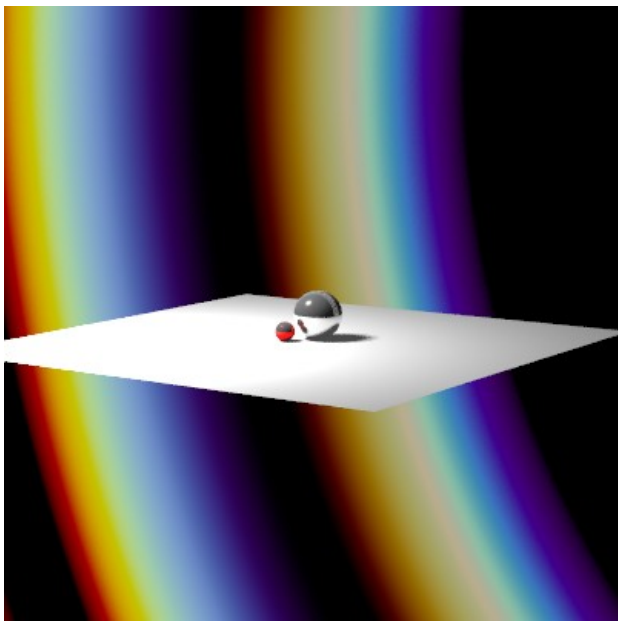
Rainbows

Rainbows are a visually stunning phenomena which are still not entirely understood. The various theories about rainbows approximate their behavior and are thus unable to render accurate rainbows. Each successive theory also requires more computation in order to render a quality rainbow as well.

The first method of that used Descartes's theory, was by far the quickest, because it did not require sampling. However, the results were not good. This is partially due to the complex nature, but also due to the method of converting between wavelength and RGB I implemented. My implementation broke the wavelengths into linear gradients which when used with jittered sampling was alright, but the first method of using the Descartes equations accentuated this problem.

The second method gave quite good results, and was decently fast. Due to the sampling the actually rainbow was smooth, and faded from the background. The only flaw was that it was missing some of the smaller, yet more interesting phenomena. This is because Descartes did not have knowledge of all the characteristics of light at the time when he formulated these equations. For a small budget this method can produce decent results.

By using De Morgan's approximation to the Airy Integral, it is possible to render much more accurate rainbows. However, it does require some tinkering with constants and a considerable amount of time. The method also has some error which could cause problems as well. De Morgan's approximation can only approximation the integral between -4 and 5. Also the infinite series in De Morgan's approximation must be truncated. By using too few terms, the number of supernumerary bows decreases, and each bow begins to blend too much creating a white band in the center, as shown below:



Too few terms used in De Morgan Approximation

Lorenz-Mie theory does provide a very accurate way to render rainbows, but it does take some work to take the initial data from the solution and make it usable for Graphics. However, it is extremely to calculate, making it impossible to compute while rendering. For this reason it is necessary to precompute the values then read them in during rendering. This means that for a good result, a large amount of time must be spent precomputing the results for one set of parameters. It took me on my Lenovo 45 minutes to compute 100 wavelengths for angles between 0 and π using a delta .0001. It later took me 5 hours to compute 1000 wavelengths for the same angles.

Conclusion

Gems

We added Fresnel reflection, refraction, and dispersion for more accurate lighting effects when rendering transparent objects. Many approximations were used, which could be a possible path for improvement. The biggest drawback of these lighting effects is the extra computational costs, especially with photon mapping. Our images also contain lots of noise, which could be remedied by either increasing sample sizes or using anti-aliasing, both of which would dramatically decrease performance.

Participating Media

Obviously, this algorithm cannot be used in any sort of real time application, but when accurate renderings are desired, this can produce high quality images. In terms of speed, there are a number of potential optimizations, the first of which is parallelization. Since each ray need only access the photon map and geometry, and has no need for communicating with other rays, simply running each ray's calculation on a different thread could result in a much faster rendering time. Even more ideal than CPU parallelization would be massive GPU parallelization, sharing the photon map between hundreds of GPU threads and running each ray on a thread would result in dramatic speedup. Additionally, the ray marching algorithm could be improved by doing adaptive ray marching, which starts with a large time step, and recursively halves the time step if the radiance differs too much in between measurements. In this way, a smaller timestep could be used in most situations, which would improve render times dramatically.

Rainbows

Rainbows are a complicated phenomena, which are complicated to render. The best possible solution is to simulated a single water droplet and use that data. However, in lieu of this technique it is possible to render rainbows by making use of various theories in optical physics. Each method has a trade off between quality and computing time. The quality comes by way of what phenomena can be rendered, which all require more time. Although each successive phenomena alters the overall rainbow by a small amount, its inclusion does enhance the overall rainbow, because it more accurately mimics a real rainbow.

Future Work

Gems

Speeding up up the rendering process is the biggest issue to work on. Something that would have been cool to add given more time was volume absorption, to more accurately simulate color properties of crystal structures would have made for a more brilliant diamond render. Using

different gem models would have also been great, as the properties of gems vary greatly with their shape. Getting the normal diamond model rendering was extremely difficult due to lack of triangle faces. Something else that should have been added was better support for rendering gems, participating media, and rainbows at the same time. It is possible, but many bugs exist in the code preventing the best renderings and performance issue when trying to render all at once.

Participating Media

As mentioned before, speeding up the algorithms by parallelization and using adaptive ray marching would be highly desirable additions. It would also be very interesting to vary the extinction coefficient by position to properly model heterogeneous media. This could either be a static scene with randomly generated heterogeneous media, or a smoke simulation could be used in conjunction with these algorithms to create realistic renders of the simulated substance. Produced solutions would also benefit from some method to more evenly distribute photons in case of low photon counts, to avoid “splotchy” artifacts in the final render.

Rainbows

Improving the Airy and Lorenz-Mie renders would allow for a better comparison between each of the methods. It would also possibly allow for direct comparisons and the generation of images that show exactly how each successive theory changes the render of the rainbow.

Implementing the algorithm in the Related Works section would allow for more accurate renders.

Bibliography

- Hong Du, "Mie-Scattering Calculation," Appl. Opt. 43, 1951-1956 (2004) http://www.opticsinfobase.org/ao/abstract.cfm?URI=ao-43-9-1951
- Casselman, Bill. "Feature Column from the AMS."American Mathematical Society. N.p., n.d. Web. 2 May 2013. <http://www.ams.org/samplings/feature-column/fcarc-rainbows>
- Wiscombe, W.J.. "Improved Mie Scattering Algorithms." Applied Optics 19, no. 9 (1980): 1505-1509.
- Sadeghi, Iman, Adolfo Munoz, Philip Laven, Wojciech Jarosz, Francisco Seron, Diego Gutierrez, and Henrik Wann Jensen. "Physically-Based Simulations of Rainbows." ACM Transactions on Graphics 31 (2012). http://graphics.ucsd.edu/~henrik/papers/physically_based_simulation_of_rainbows.pdf (accessed April 29, 2013).
- [1] Jensen, Henrik Wann. "A practical guide to global illumination using ray tracing and photon mapping." In ACM SIGGRAPH 2004 Course Notes, p. 20. ACM, 2004.
- Jensen, Henrik Wann, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. "A practical model for subsurface light transport." In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 511-518. ACM, 2001.
- Jensen, Henrik Wann, and Per H. Christensen. "Efficient simulation of light transport in scences with participating media using photon maps." InProceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 311-320. ACM, 1998.
- Wojciech Jarosz. Efficient Monte Carlo Methods for Light Transport in Scattering Media. Ph.D. dissertation, UC San Diego, September 2008.
- Steven Collins. “Rendering Crystal Glass,” Proc. of the 2nd Irish Workshop on Graphics, 1994.
- Yinlong Sun, F. David Fracchia, Mark S. Drew. "Rendering Diamonds," Proceedings of the 11th Western Computer Graphics Symposium, 2000.